Clab-2 Report

Junlin Han

U6835134

June 27, 2020

1 Task 1 Harris Corner Detector

1.1 Read and understand the above corner detection code

For this task,I reviewed lecture slides and read it carefully.

1.2 Complete the missing parts, rewrite 'harris.m' into a Matlab function, and design appropriate function signature

1.3 Comment on line 13 and every line of your solution after line 20

Based on given code,we need to compute the harris cornerness,then perform non-maximum suppression and thresholding.I will attach some of my code here.

```
1  %My harris detector function,bw is gray image.
2  function  my_out = my_harris(bw,thresh,k)
3  [m,n]=size(bw);
4  sigma = 2;  % Parameters, add more if needed
5  sze =11;% sze is short for size.
6  % Derivative masks
7  dx = [-1 0 1;-1 0 1; -1 0 1];
8  dy = dx'; % dx is the transpose matrix of dy
9  % compute x and y derivatives of image
10 Ix = conv2(bw,dx,'same');
11 Iy = conv2(bw,dy,'same');
12 %fix is round towards 0,fspecialc create predefined 2-D ...
       gaussian filte,here
13 %it's the form of h = ...
       fspecial('gaussian',hsize,sigma).Returns a
14 %rotationally symmetric Gaussian lowpass filter of size hsize
15 %with standard deviation sigma.
16 g = fspecial('gaussian',max(1,fix(3*sigma)*2+1),sigma);
17 Ix2 = conv2(Ix.^2,g,'same'); % x and x
18 Iy2 = conv2(Iy.^2,g,'same'); % y and y
19 Ixy = conv2(Ix.*Iy,g,'same'); % x and y
```

Next part we computes the harris cornerness. R is the corneerness in my code. The formula is,for small shifts [u,v], we have a 1st order Taylor approximation:

$$E(u,v) \approx \begin{pmatrix} u,v \end{pmatrix} M \begin{pmatrix} u \\ v \end{pmatrix}$$

Matrix M has the following definition.

$$\sum_{x,y} w(x,y) \begin{pmatrix} IxIx & IxIy \\ IxIy & IyIy \end{pmatrix}$$

I found we don't need to go through every pixel since the given code already done the sum step.So we can define matrix M

$$\mathbf{M} = \begin{pmatrix} Ix2 & Ixy \\ Ixy & Iy2 \end{pmatrix}$$

Then we get the corneerness R by this formula

$$R = det(M) - k(trace(M))^2$$

```
% Task: Compute the Harris Cornerness
k=0.04;% usually 0.04-0.06,0.04 widely used.
% calculate the response of the detector.The fromula is ...
    matrix M=[Ix2,Ixy;Ixy,Iy2],R =det(M)-k*(trace(M))^2;
R=(Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2;
```

Now we can threshold on value of R,compute non-max suppression.Some code is givin in the lecture slide Week-4A page 56.We can find the local maximum by Matlab inbuilt function **ordfit2()** ,this function is equal to a maximum filter operation,find and replace the maximum of all pixels within a local region(11*11) of R.The last step is to collect data and output the result.

```
% Task: Perform non-maximum suppression and thresholding, ...
    return the N corner points  as an Nx2 matrix of x and ...
    y coordinates .
%Next 3 lines of code is from lecture slide Week-4A page ...
    56.Find local maxima,which is non maximum ...
    suppression..Here,this function is equal to a maximum ...
    filter operation,find and replace the maximum of all ...
    pixels within a local region(11*11) of R.
mx = ordfilt2(double(R),sze^2,ones(sze));
% Thresholding proccess,also find local max.
R = (double(R==mx))&(double(R>thresh));
%get result and pass it to [rows,columns].
[rows,columns] = find(R);
%get result to my_out.
```

```
9   my_out=[columns,rows];
10  end
```

Function part is done,I apply this function on test images and use matlab inbuilt function ***corner*** for comparison .Some code is quite similar so I only commented it for once.

```
% CLAB2 Task-1: Harris Corner Detector
% output Harris image 1
img1 =imread('Harris_1.jpg');%read the image
img1=rgb2gray(img1);% convert to gray image
my_out1=my_harris(img1);%apply function
imshow(img1),title('My harris result,Harris image ...
    1');%show the image
hold on;%hold on to plot the corners
plot(my_out1(:,1),my_out1(:,2), 'or');%plot corners
hold off%hold off,done
%use matlab inbuilt harris corner on Harris image 1
C1 = corner(img1);%matlab inbuilt function
imshow(img1),title('Matlab inbuilt corner,Harris image ...
    1'),%show result
hold on;%hold on to plot the corners
plot(C1(:,1), C1(:,2), 'or');%plot corners
hold off%hold off,done
```

1.4 Test this function on the provided four test images. Display your results by marking the detected corners on the input images

1.5 Compare your results with that from Matlab's built-in function corner(), and discuss the factors that affect the performance of Harris corner detection

Figure 1 to figure 4 show the result of my harris corner and matlab inbuilt corner.My harris corner detects less corner than inbuilt harris corner.Especially from figure 4 we can see the difference obviously.There are some factors may influence the result.In our method,we generate our gaussian filter with sigma value while matlab inbuilt function default gaussian filter with

$$fspecial('gaussian', [5 \quad 1], 1.5).$$

Also,we perform non-maximum suppression and thresholding while matlab inbuilt function corner only perform non-maximum suppression.For thresh value,I tried some of them and found a good balance between detect all corners and make less errors.Thresh value can significantly impact the result.I didn't find documents about the thresh value for matlab inbuilt function corner,this could be an important factor. The value of k,sensitivity factor. I choose 0.04 which is widely used.Matlab inbuilt function corner also use 0.04.

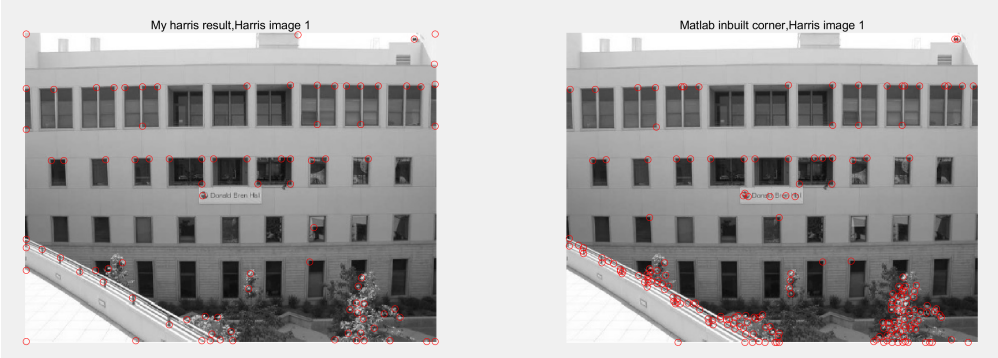


Figure 1: Comparison of results (a) My harris corner (b) Matlab inbuilt harris corner

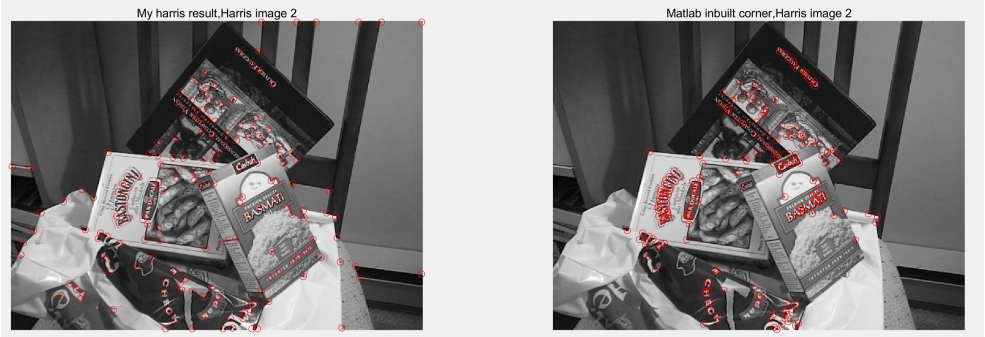


Figure 2: Comparison of results (a) My harris corner (b) Matlab inbuilt harris corner

Figure 3: Comparison of results (a) My harris corner (b) Matlab inbuilt harris corner

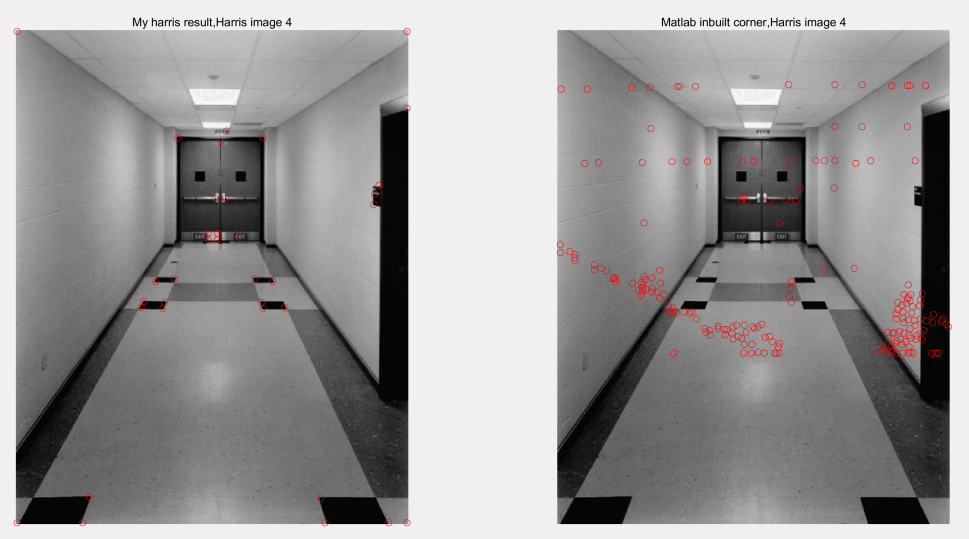


Figure 4: Comparison of results (a) My harris corner (b) Matlab inbuilt harris corner

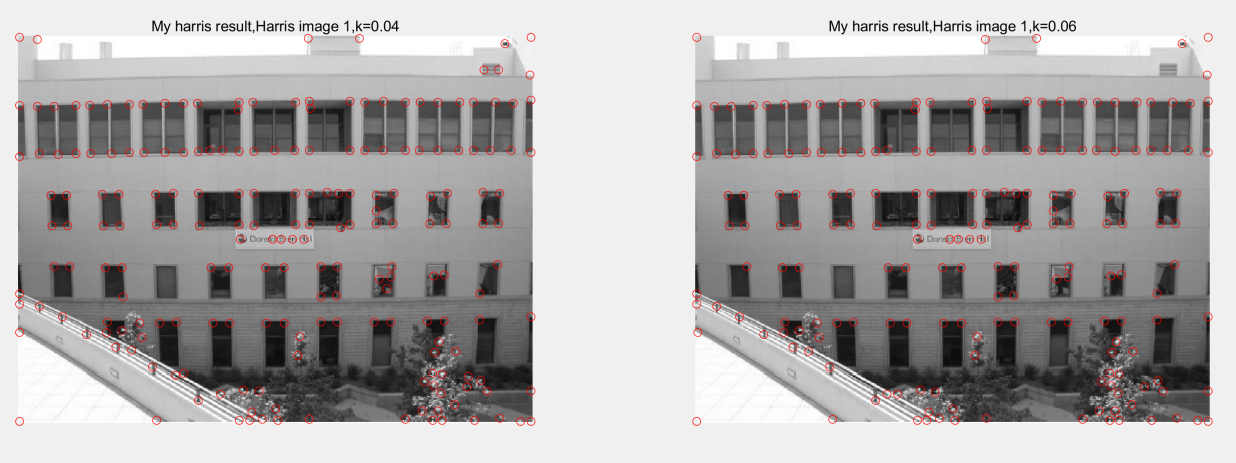


Figure 5: Comparison of results (a) K value 0.04(b) K value 0.06

Figure 6: Comparison of results (a) Thresh value 130000000(b) Thresh value 10000000

Here is some comparison with different factor values.
From figure 5 we can observe that the smaller k is,the more likely to detect sharp corners.
From figure 6 we can observe that the corners become much fewer with a bigger threshold value obviously.

2 K-Means Clustering and Color Image Segmentation

2.1 Implement your own K-means function

K means clustering is implemented with following steps 1.Initialize cluster centers 2.By computing the distance, assign each data point to the closest center. 3.Recompute centers after assignment. 4.Iterate until no points are reassigned.
Fro step 2,the distance is Euclidean distance. We can calculate it with this formula

$$D(Xi, Cj) = \sqrt{\sum_{i=1}^{N}(Xi - Cj)^2}$$

where x is the points,c is the cluster central.We don't need to find the square root in practice.
Our goal is color image segmentation,so I assume the input is 3-dimension.If we just input some data in 2-dimension,we need to adjust [a, b, c] = **size**(Img) to [a, b] = **size**(Img). I reshape the data first in my kmeans function for convenience,this idea is similar to PCA.The disadvantage is that the output need to be reshaped back to show the result.

```
function [C, label] = mykmeans(Img, k)
[a, b, c] = size(Img);
%reshape data input to X with [a*b,c] size
X = reshape(double(Img), a*b, c);
% X size [k,c].C is clusters,assign points randomly.
C = X(randperm(a*b, k), :);
% store value
temp = inf;
while true
    %use sum to reshape the size,sum(X.^2,2) will output a ...
       [a*b,1] size
    %matrix,sum(C.^2, 2) will output a [k,1] size matrix.
    %calculate the distance.D with size[a*b,k]
    D = sum(X.^2, 2)*ones(1, k) + (sum(C.^2, 2)*ones(1, ...
       a*b))' - 2*X*C';
    %assign each point to the closest center
    [¬, label] = min(D, [], 2) ;
    for i = 1:k
        %recalculate cluster central then reassign points ...
           to cluster.
```

```
        C(i, :) = mean(X(label == i , :));
    end
    %calculate the distance.
    dist = sum(sum((X - C(label, :)).^2, 2));
    %Calculate the difference between iterations,if it's ...
       small enough then
    %break.
    if abs(dist-temp) < 0.000001
        break;
    end
    %store last dist value in temp.
    temp = dist;
end
end
```

2.2 Apply your K-means function to color image segmentation

One of the test image is in uint16,so we have to convert it to uint8 first.

```
img2=uint8(img2/2^8);
```

We need to convert RGB image to Lab image,then encode it as a 5-D vector instead of 3-D vector.I wrote a function to realize this.

```
function dataset =makedataset(img)
[m, n, p] = size(img);
dataset = zeros(m,n,5);
for i=1:m
    for j=1:n
        dataset(i,j, 1) =img(i,j,1);
        dataset(i,j, 2)  =img(i,j,2);
        dataset(i,j, 3) = img(i,j,3);
        dataset(i,j, 4) = i;
        dataset(i,j, 5)  = j;
    end
end
```

I tried some different k values for comparison.From figure 7 and figure 8.We can observe that segmentation results works well with different k values.And the k means with coordinates result have a good relationship with coordinates and it's more likely to assign pixels together if pixels are close.

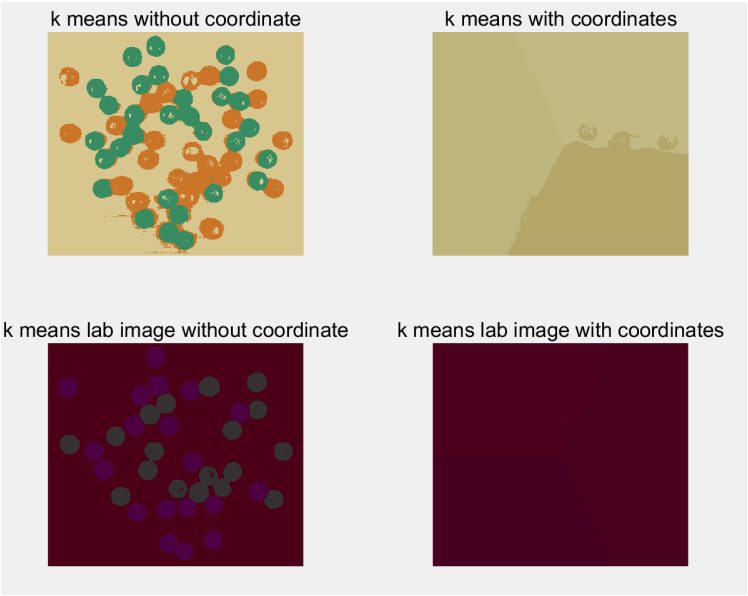


Figure 7: K =3 results,mandm

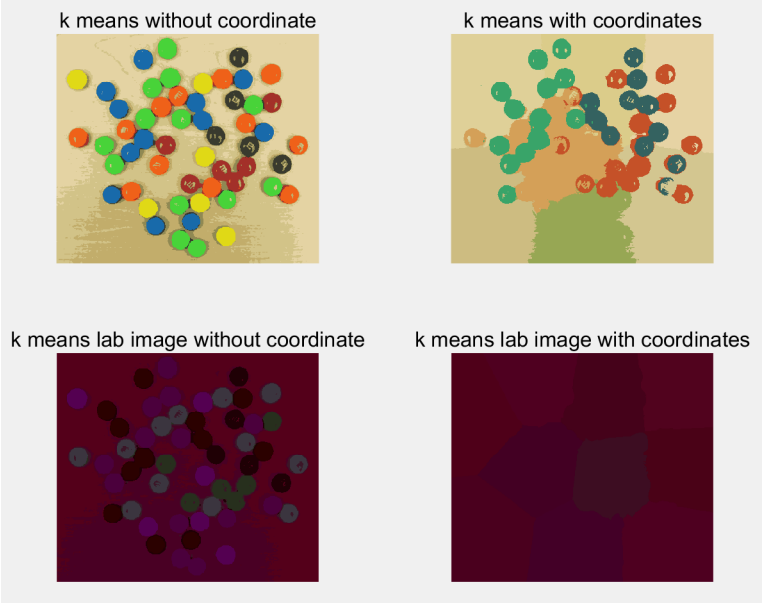


Figure 8: K =10 results,mandm

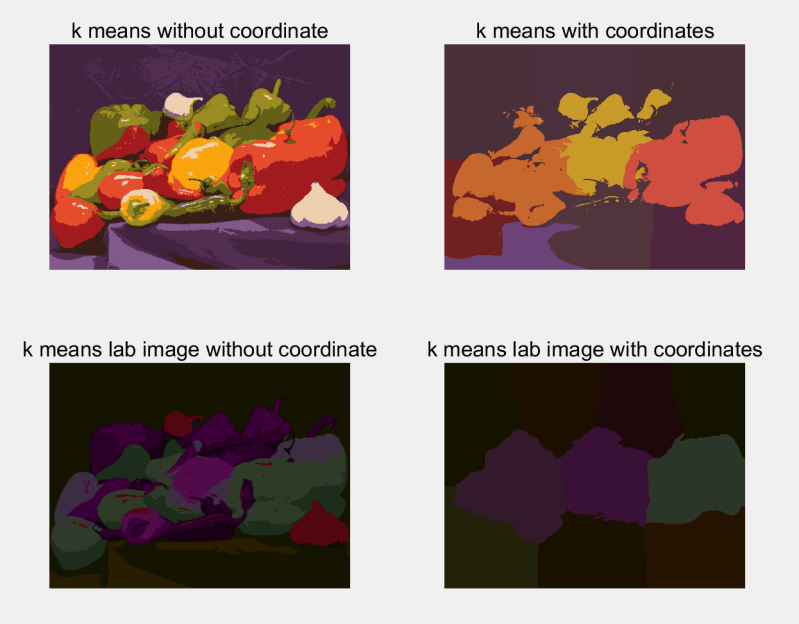


Figure 9: K =10 results,peppers

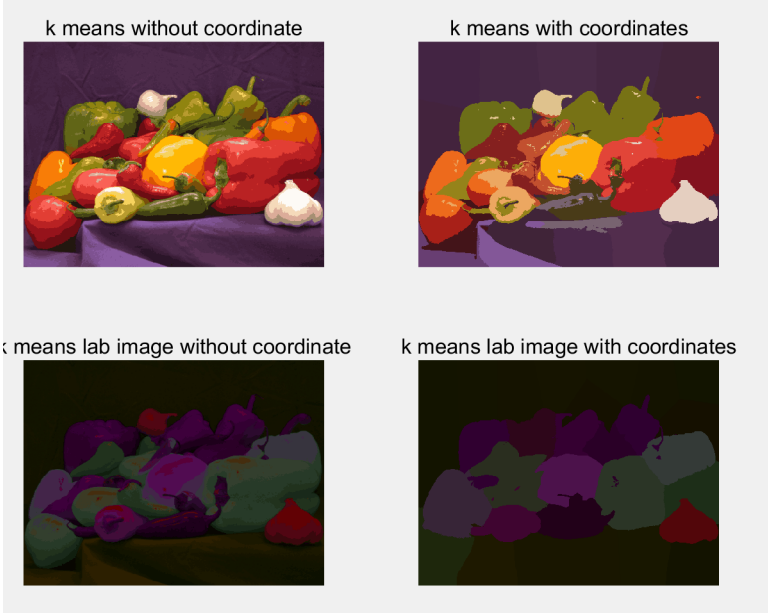


Figure 10: K =30 results,peppers

From figure 9 and figure 10.We can draw the same conclusion as before.And I choose a large k value here,we can see the larger k is,the more similar to original image.

2.3 K-means++

summarize the key steps in the report

The difference between k-means and k-means ++ is about initialization.While k-means choose k centers of clusters randomly,k-means ++ will follow these steps.

1 Choose one center C1 randomly among the data points.

2 Calculate the distance D between data points and C1.

$D(Xi, C1) = \sqrt{\sum_{i=1}^{N} \|Xi - C1\|^2}$

3 Choose second center C2, with probability proportional to D^2.

Pr=D(Xi,C1)$^2 / \sum_{j=1}^{N} D(Xj, C1)^2$

4 Recompute the distance D.

$D(Xi, C2) = \sqrt{\sum_{i=1}^{N} \|Xi - C2\|^2}$

5 Compute the distance vector $D = min(D(Xi, C1), D(Xi, C2)...D(Xi, Cj))$

6 Choose new centers,and do steps 2-5 until get k centers.

7 The rest of algorithm is same as k-means.

implement K-means++

```
C = Img(:,1+round(rand*(size(Img,2)-1)));%first center
    L = ones(1,size(Img,2));
    for i = 2:k
        D = Img-C(:,L);
        D = cumsum(sqrt(dot(D,D,1))); %add the sum of distance
        if D(end) == 0
            C(:,i:k) = Img(:,ones(1,k-i+1));
            return;
        end
        C(:,i) = Img(:,find(rand < D/D(end),1)); %The ...
            second parameter of find indicates the number ...
            of indexes
        [¬,L] = ...
            max(bsxfun(@minus,2*real(C'*Img),dot(C,C,1).')); ...
            %assign points
    end
```

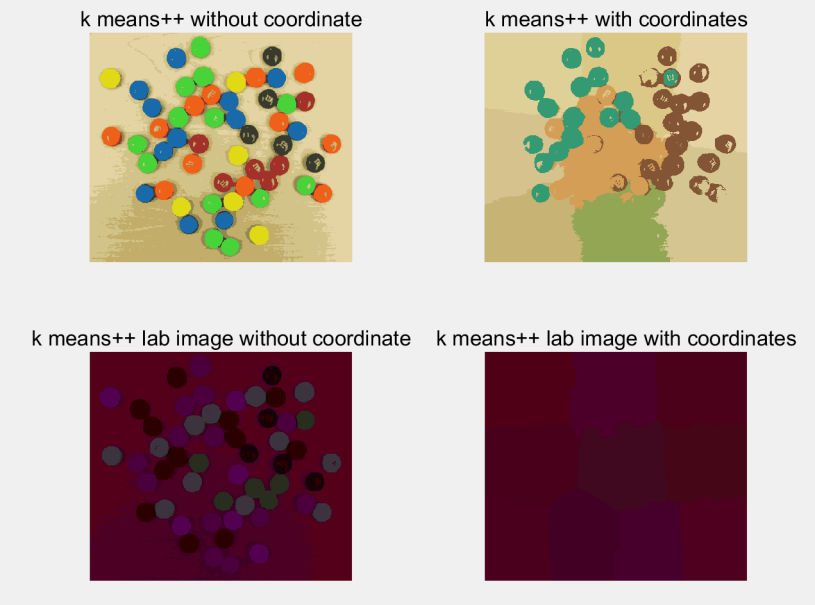


Figure 11: K =10 results,mandm,with k means ++

Compare the image segmentation performance

I choose to use 5-D vector mandm lab image with k=10 for comparison.
Figure 8 shows the segmentation result of normal k means algorithm.
Figure 11 shows the segmentation result of k means ++ algorithm.
We can observe that the segmenatation results are same.
To find the convergence speed,I tested them independently with 5-D vector lab image..I use iteration times to show the speed. I tested each one for 5 times.

Algorithm	K value	average	min	max
Kmeans	3	21	19	24
Kmeans++	3	20	17	27
Kmeans	5	76	47	122
Kmeans++	5	53	41	92
Kmeans	10	102	53	215
Kmeans++	10	81	66	127

From this table we can observe K-means ++ will converge faster in average,and my results may not show the true performance of K-means ++ due to the limit of test times.

3 Face Recognition using Eigenface

3.1 Unzip the face images and get an idea what they look like

3.1.1 Why alignment is important?

W must make sure all the test/train images are aligned image.If not,lots of noise will occur and damage our recognition.The consequence,for instance,the central part of the image should be the nose.But unaligned image's central could be mouth/eyes.Hence unaligned image impact our recognition system.

3.2 Train an Eigen-face recognition system

3.2.1 Read all the 135 training images from Yale-Face,collect all the data points into a big data matrix

I will attach my code here,it shows how I read the text and the big data matrix. I noticed that the length of **traningDataDir** is 137,but 1-2 are some hidden file which are useless.So I start for loop with initial count as 3 to ignore the hidden file.I also resize those images to [100,100] since the limited memory.

```
traindir_name = 'F:\Task3\trainingset\';
trainimgDataDir = dir(traindir_name);
train_data_matrix=[];
for i = 3:length(trainimgDataDir)
    %135 images but length137,the first/second is ...
       unnecessary,i start from
    %3 to igore them.
    name = trainimgDataDir(i).name;
    img = imread([traindir_name,name]);
    img=imresize(img,[100 100]);
    [a,b] = size(img);
    img = reshape(img,[a*b,1]);
    img = double(img);
    train_data_matrix= [train_data_matrix,img];
end
```

3.2.2 Perform PCA on the data matrix,display the mean face, explain the reason,implement it in your code.

To perform PCA on the data matrix,we need to follow 2 steps.
Step 1 calculating the data covariance matrix.
We first find the mean matrix of train data.Then use our big data matrix to subtract the mean matrix.My big data matrix with size [10000,135] which comes from [height of image*width of image,numbers of images].And my mean matrix with size [10000,1].So I use the **repmat** function to make sure every train image in the big data matrix subtract the mean matrix.
I use inbuilt function **std** to find the standard deviation of the big data matrix.Then apply inbuilt function **bsxfun** .It will save memory and it's a more graceful way than using inbuilt function **repmat**.

```
%find  standard deviation
sig = std(train_data_matrix);
%appliesthe element-by-element binary operation right ...
    array divide to arrays A and B
%every element in data_matrix divide corresponding ...
    elements in sig.
train_data_matrix_divided = ...
    bsxfun(@rdivide,train_data_matrix,sig);
```

We normalized our data matrix,now we can find the covariance matrix.

```
covmat=cov(train_data_matrix_divided);
```

Step 2 performing eigenvalue decomposition on the covariance matrix.
We just use inbuilt function **eig** to find the eiginvector matrix we need.
Display the mean face
Figure 12 shows the mean face image.
Explain the reason
If we run the code following the PCA algorithm we implemented above,it take some time. One alternative way is directly apply svd decomposition on normalized data matrix,this way will save some memory and runs faster since we don't need to implement covariance matrix by ourselves.The result is same.
There are one more way to perform PCA.M is the data matrix after subtracting mean matrix.If we use MM^T to compute the eigin value and eigin vectors.One general solution for this case is,we can form a covariance matrix with smaller

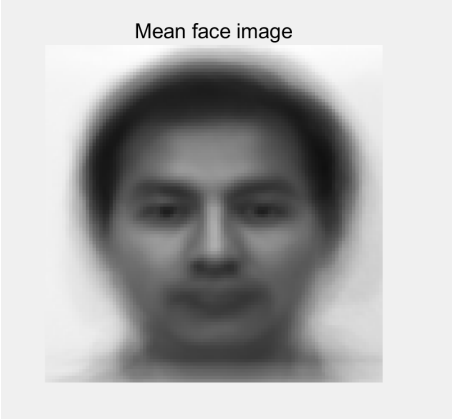


Figure 12: mean face

size.For our data matrix M,instead of calculating eiginvectors and eigin values from$MM^T(45045 * 45045)$,we can compute them from $M^T * M(135 * 135)$.
Let's assume V is the eiginvector of MM^T.
We have $MM^TV = \alpha V$,where α is just some relationship between them.
We multiply M^T in both sides.
$M^TMM^TV = M^T\alpha V = \alpha M^TV$
So we can conclude that α is a eigin value of M^TM and the corresponding eigin vector is M^TV.
Implement it in your code
We just simply use inbuilt function **svd**,since I didn't use MM^T for computation there's no need to implement this method here.

```
%U is the eigin vector matrix ,S is the eigin value matrix
[U,S,V] = svd(train_data_matrix_divided);
```

3.2.3 Determine the top k principal components

I choose k=10.We find the top 10 eiginvector and reshape it back then display the results.
Figure 13 show the gray image of top 10 eigin faces.

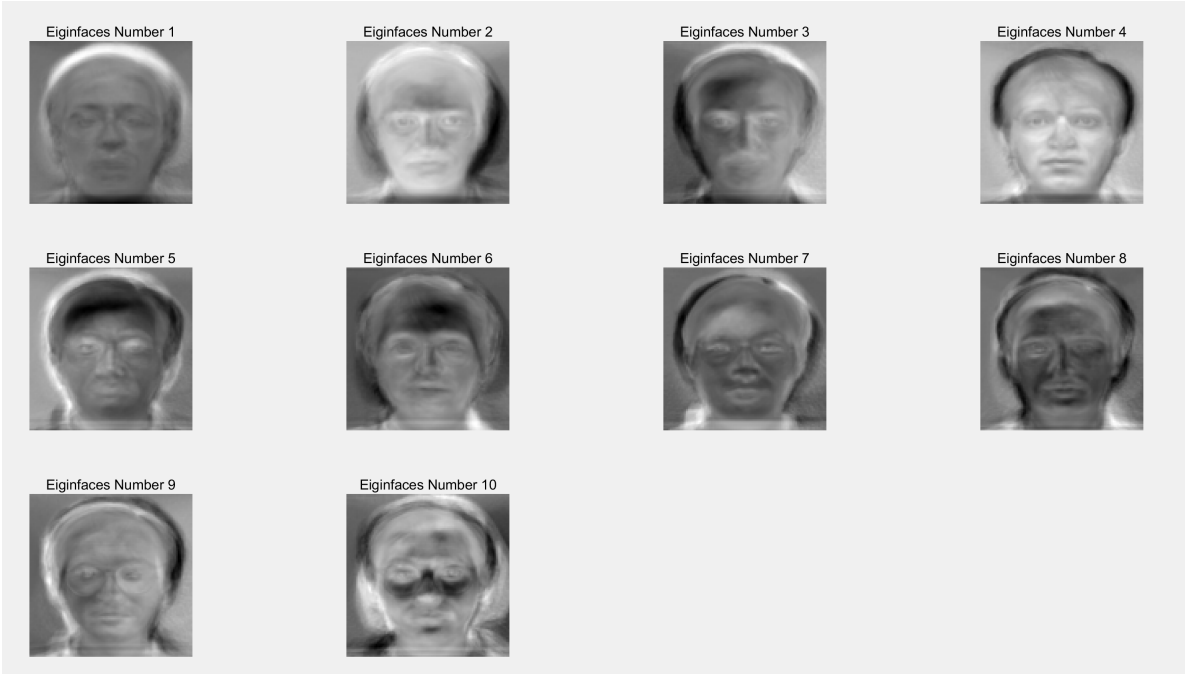


Figure 13: Top 10 eigin faces

3.2.4 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

We do the same process of reading images and store all test data in to one matrix with size [10000,10].We find projection then apply nearest-neighbour search to find the most similar faces.Matlab inbuilt function **norm** will calculate the Euclidean norm of vector,then we can apply inbuilt function **sort** and find the top 3 most similar images.

```
test_data_matrix=test_data_matrix - ...
    repmat(train_mean_matrix,1,(length(testimgDataDir)-2));
train= ((U * U)\(U' * train_data_matrix));
test = ((U * U)\(U' * test_data_matrix));
distance = zeros(10,135);
matchs = zeros(1,30);
for i = 1:10
    for j = 1:135
        distance(i,j) = norm(test(:,i)-train(:,j));
    end
end
for i=1:10
    [¬,order] = sort(distance(i,:));
    matchs(3*i-2) = order(1);
    matchs(3*i-1) = order(2);
    matchs(3*i) = order(3);
end
```

Show these top 3 faces

Figure 14-18 show the results of top 3 faces.

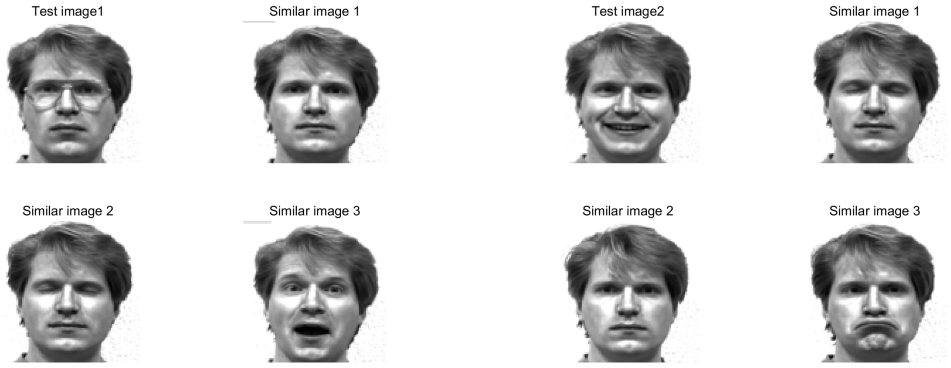
Report and analyze the recognition accuracy of your method

For all test images and top 3 faces, the overall recognition rate is 28/30,which is 93.3%.

The two mismatch faces all happen in test image 8.

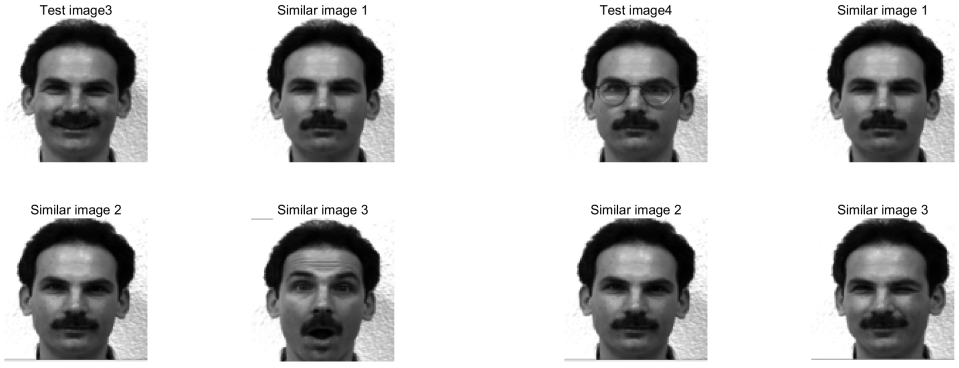
We can observe from figure 17(b),the second and third similar image are wrong.I think it's plausible to have these two mismatch faces since these 2 man have similar hairstyle.

In conclusion,the accuracy of face recognition using eigenface is acceptable with good training data.It's a traditional algorithm for face recognition without deep learning so we can't expect it to be perfect.



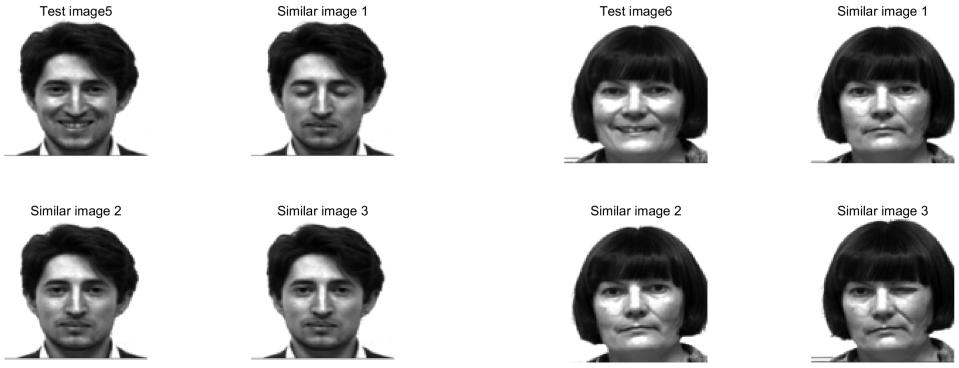
(a) (b)

Figure 14: Results of top 3 faces



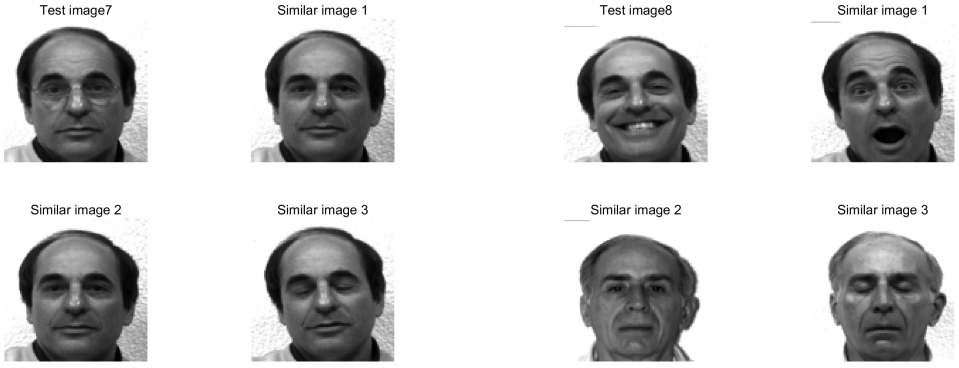
(a) (b)

Figure 15: Results of top 3 faces



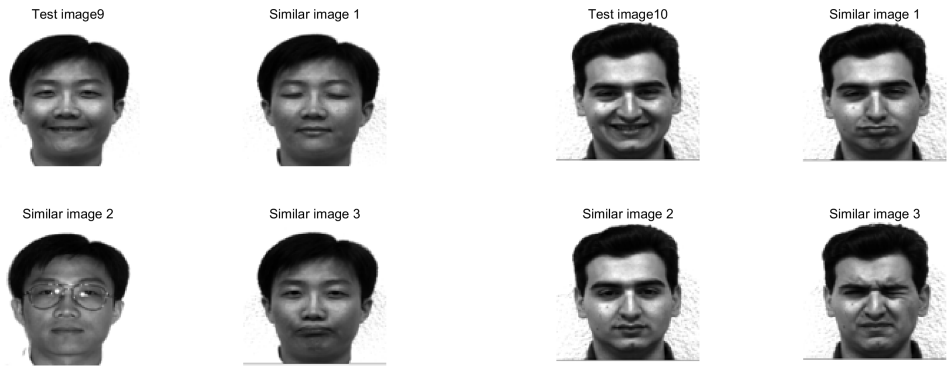
(a) (b)

Figure 16: Results of top 3 faces



(a) (b)

Figure 17: Results of top 3 faces



(a) (b)

Figure 18: Results of top 3 faces

3.2.5 Read in one of your own frontal face images.

I took 10 photos and split them into 2 groups.3 photos are taken before cutting hair.Others are taken after cutting hair.I align them by manually crop.They're roughly aligned.
I use one of 'before cutting hair' image as test image.If eiginfaces algorithm works well,we anticipate the other 2 'before cutting hair' are the most 2 similar faces.
From figure 19,we can observe that our face recognizer won't work if we don't train it with some data first.

3.2.6 Repeat the previous experiment by pre-adding the other 9 of your face images into the training set

From figure 20,we can observe that our face recognizer works well if we train it first.The most 2 similar images are all 'before cutting hair' image.

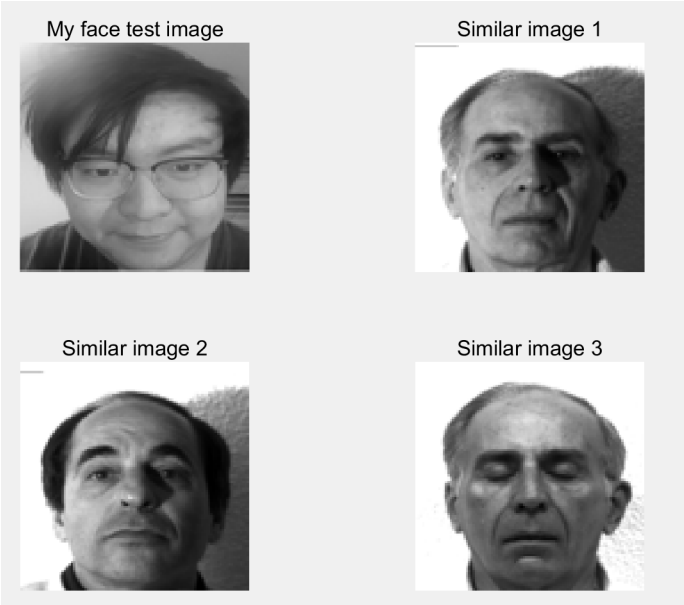


Figure 19: Results of top 3 faces without adding into the training set

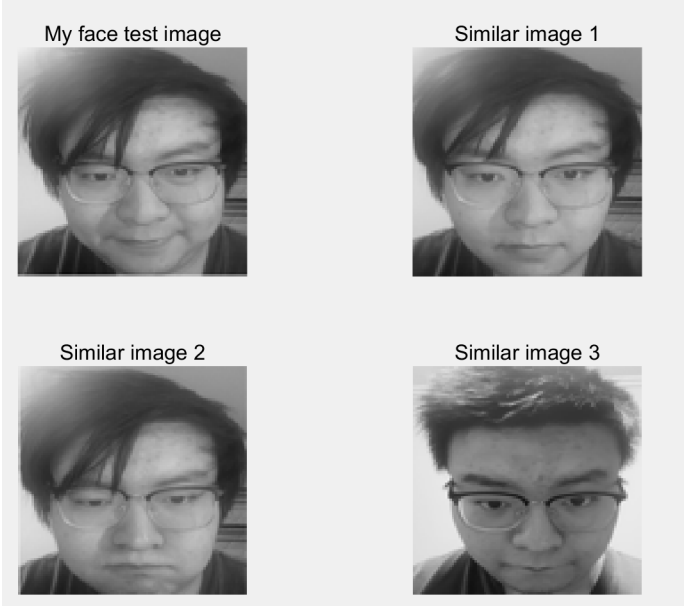


Figure 20: Results of top 3 faces with adding into the training set