CLAB-3 Lab Report

ENGN 4528/6528, Computer Vision 2018

College of Engineering and Computer Science

Student Name: CHAOYUN GONG

UID: u6329142

Contents

Task-1: Face Recognition using eigenface technique

1.

At the beginning of the task, the 135 face images need to be cropped to make sure the face is in middle. The code is below:

```
imagepath = 'CLab3_materials/yalefaces/testset/'; %the path for getting the orginal faces.
savepath = 'CLab3_materials/yalefaces/testset2/'; %the path for saving cropping faces.
Files = dir([imagepath '*.png']); %open the orginal    face file.
for j=1:length(Files) %figure show all the face image.
            I=imread(strcat(imagepath,Files(j).name));
      imshow(I);
      [x,y] = ginput(2);    %using ginput() function to get the cropping face's starting and ending
coordinates.
      if x(1)==x(2) && y(1)==y(2)    %if double-click image, skip this image.
            continue;
      end
      rect=[x(1),y(1),x(2)-x(1),y(2)-y(1)];    %get the new face image's coordinates and size.
      J=imcrop(I,rect);    %crop the new face.
      name=strcat(num2str(Files(j).name));    %give the same name for the new face image.
      imwrite(J,strcat(savepath,name));      %save the image in the savepath file.
end
```

2.

Reading the 135 new cropping images in savepath. And save in matrix M. The code is below:

```
trainpath = 'CLab3_materials/yalefaces/trainingset2/';
testpath = 'CLab3_materials/yalefaces/testset2/';
train_filenames = dir([trainpath '*.png']);        % return a structure with filenames
test_filenames = dir([testpath '*.png']);          % return a structure with filenames
img_num=length(train_filenames);
img_num2=length(test_filenames);

M = zeros([10000,img_num]);
M2 = zeros([10000,img_num]);
%read all train images.
for j = 1:img_num
      filename = [trainpath train_filenames(j).name];      % filename in the list
```

```
    I = imread(filename);
    I1 = imresize(I,[100 100]);% resize it to 100x100
    I1 = double(I1);
    M(:,j) = reshape(I1,10000,1);      %represent each image as a column, and make up a data matrix.
end
```

3.

PCA has four steps: [1]

a). get the average face, and minus it by big data matrix.

b). calculate the covariance matrix.

c). calculate the k eigenvectors with largest eigenvalues, then compute the eigenfaces.

d). using linear combinations of eigenfaces to represent all the training images.

The code is below:

```
MT=M';          %transpose matrix M.
M_average = mean(MT); %get the average face.
averageface = reshape(M_average,100,100);    %reshape the average face from a high dimensional vector
to a matrix.
A = uint8(averageface);
figure, imshow(A); title('average_face');    %display the average face.
%minus average face for each face.
for i = 1:img_num
    M2(:,i) = M(:,i) - M_average';
end
%perform PCA on the data matrix.
C = M2'*M2*(1/img_num); %get inner-product matrix to calculate the eigenvalues.
 [V,D] = eigs(C,k); %calculate the the k eigenvectors V with largest eigenvalues D.
%get k eigenfaces.
 eigenface = zeros(10000,k);
 for i=1:k
     eigenface(:,i)= M2*V(:,i); %calculate the eigenfaces and save in one matrix.
 end
 %show the eigenfaces.
 figure;
for i = 1:k
    image = eigenface(:,i);    %get each eigenface's high dimensional vector.
    image = reshape(image,100,100);    %reshape the vector to a image.
    subplot(2,5,i);    %show the eigenfaces in one image.
    imshow(image);title(['eigenface',num2str(i)]);
end
 %using linear combinations of eigenfaces to represent all the training images
```

```
  M3 = zeros(img_num,k);
for i = 1:img_num
      M3(i,:) = M2(:,i)' * eigenface ;      %calculate the mapping matrix and save it.
end
```

4.

The average face is in Figure 1.1

(Figure 1.1)

When k = 5, the top five eigenfaces is in Figure 1.2.

(Figure 1.2)

When k = 10, the top five eigenfaces is in Figure 1.3.

（Figure 1.3）

5.

recognize test images. It has four steps:

a). change the test image to a high dimensional vector, and minus the average face.

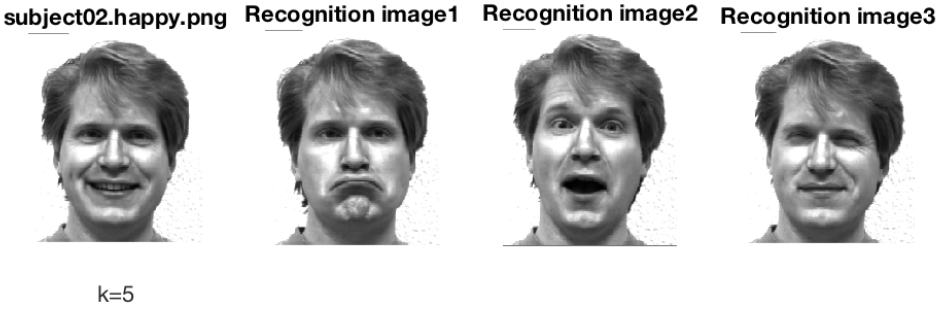
b). calculate the similarity of each training images by using eigenface.

c). using Euclidean distance to calculate the distance between test image and each training image.

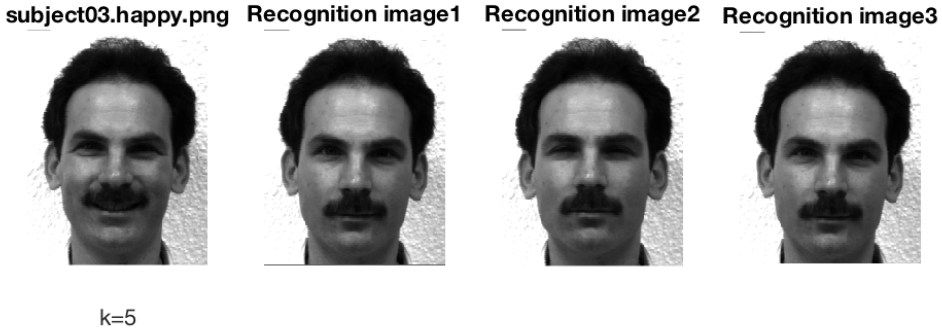
d). rank the distance and show the three training images which have the minimum distance.

When k=5, The result is below:

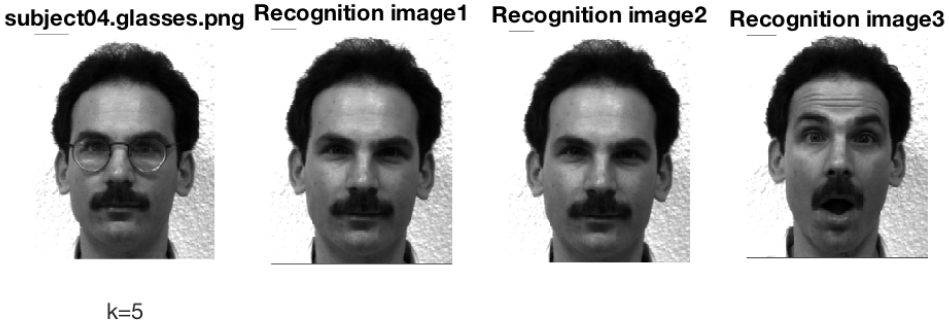
(Figure 1.4.1)



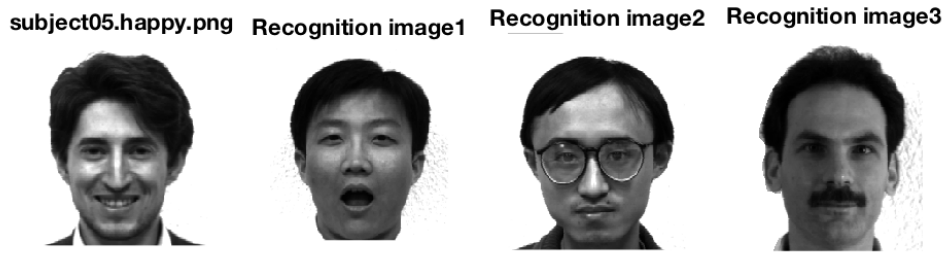
(Figure 1.4.2)



(Figure 1.4.3)

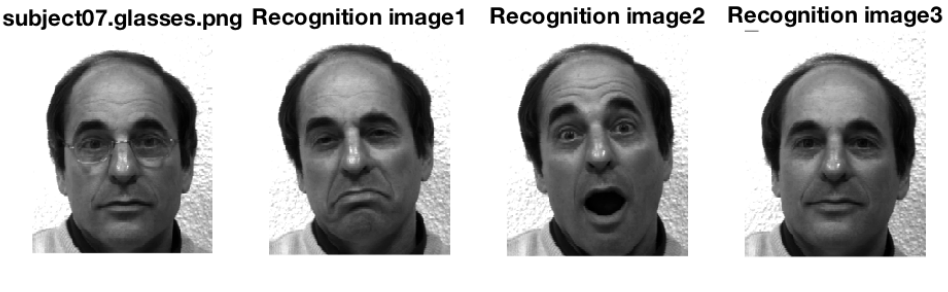


(Figure 1.4.4)

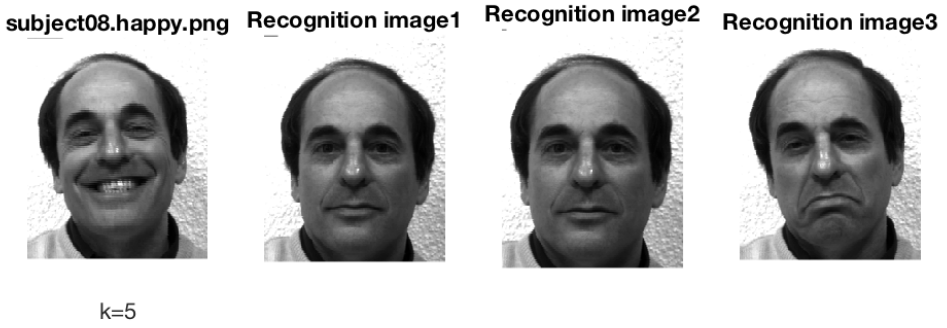


(Figure 1.4.5)

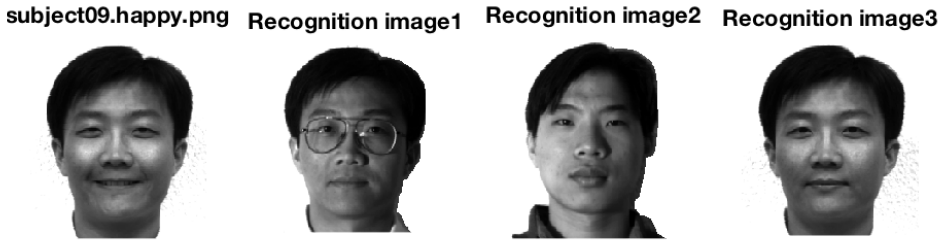
(Figure 1.4.6)



(Figure 1.4.7)

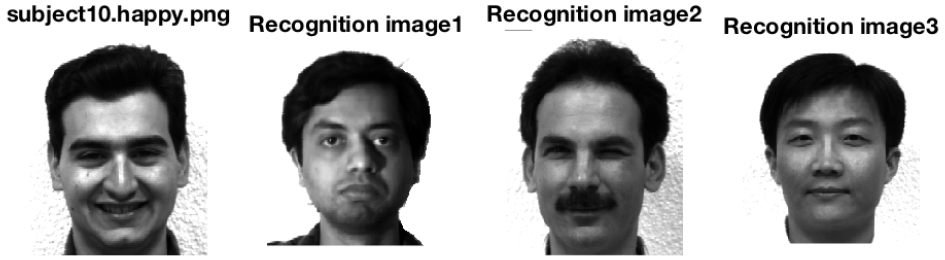


(Figure 1.4.8)



k=5

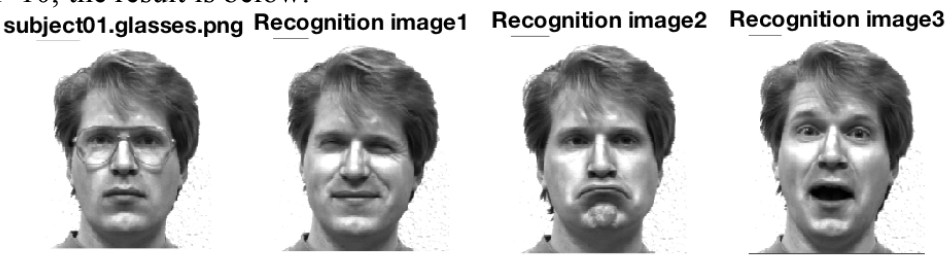
(Figure 1.4.9)



k=5

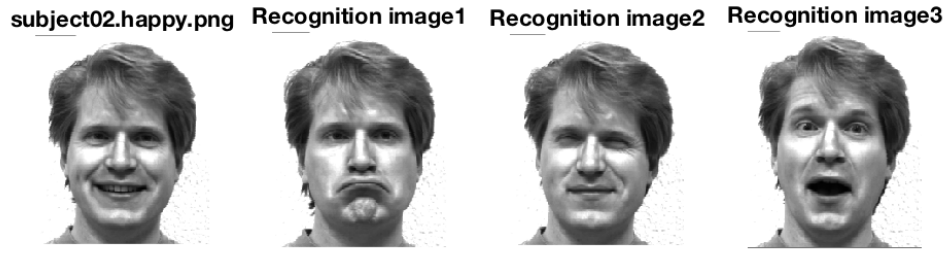
(Figure 1.4.10)

When k=10, the result is below:



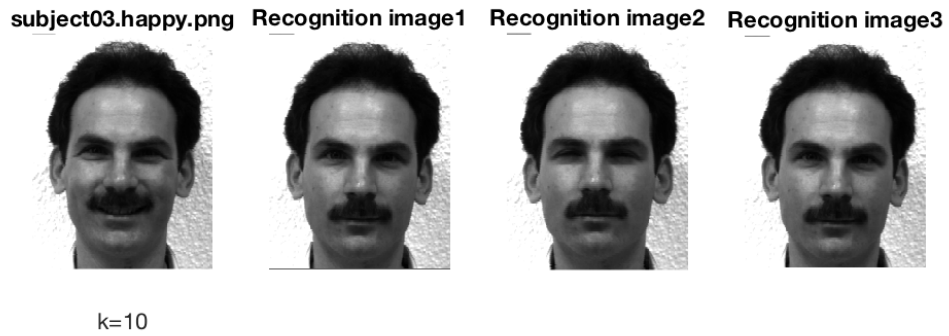
k=10

(Figure 1.5.1)

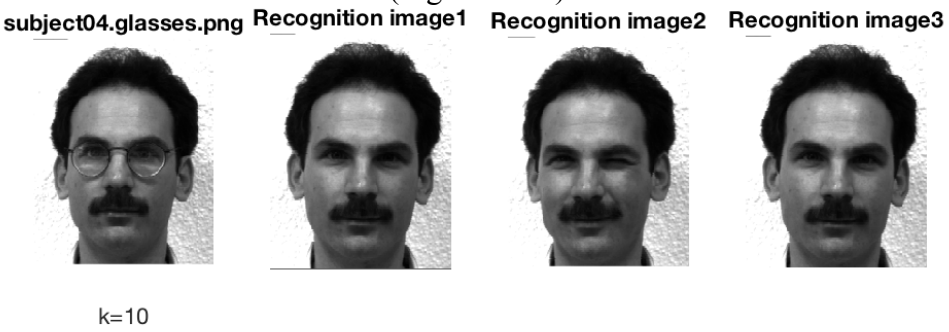


k=10

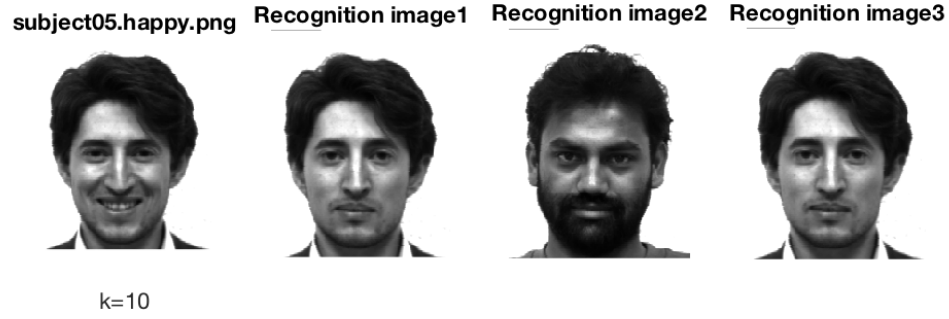
(Figure 1.5.2)



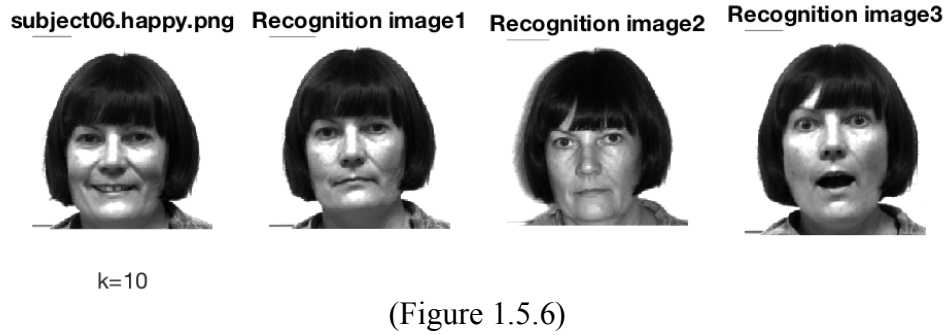
(Figure 1.5.3)



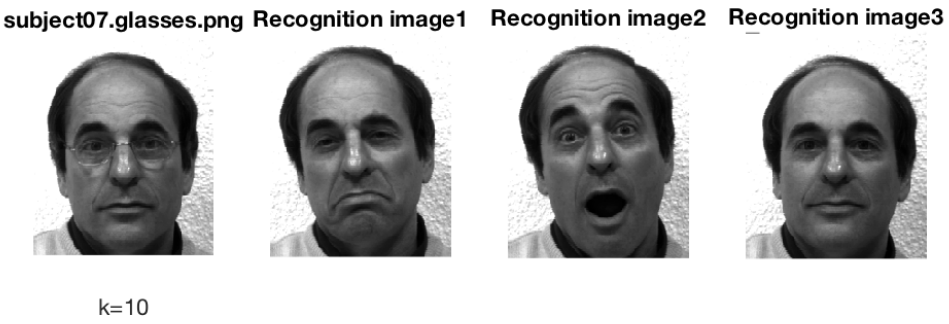
(Figure 1.5.4)



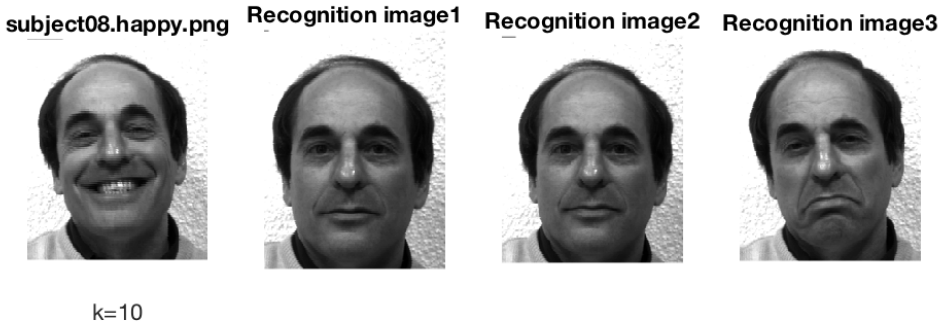
(Figure 1.5.5)



(Figure 1.5.6)

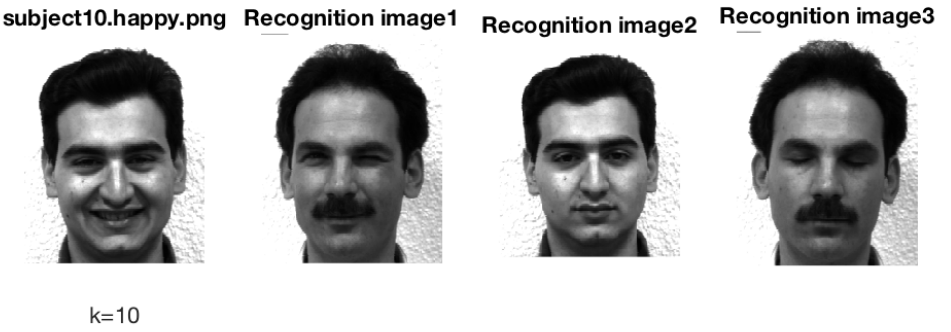


(Figure 1.5.7)



(Figure 1.5.8)

(Figure 1.5.9)



(Figure 1.5.10)

k=5, the recognition rate is 80%. k=10, the recognition rate is 90%. Within limits, more eigenfaces can recognize the test face more accurately.

The code is below:

```
%read the test images.
for n = 1:img_num2
    filename2 = [testpath test_filenames(n).name];
    t = imread(filename2);
    t1 = imresize(t,[100 100]); % resize it to 100x100
    t1 =   double(t1);
    t2 = reshape(t1,10000,1); %translate the test image to a high dimensional vector.
    t3 = t2 - M_average'; %minus the average face for each test face.
    feature_vec = t3' * eigenface ; %calculate the similarity of the input to each training image
    %using Euclidean distance to calculate the distance between test image and each training image.
    for j = 1:img_num
        gap = feature_vec - M3(j,:);
        d(1,j)= sqrt(sum(gap.^2));
    end
    %compare the distances and find the three minimum values.
    [val,indx] = sort(d); %range the distances.
    min_distance = indx(1); %find the minimum distance image number.
    sec_distance = indx(2); %find the second smallest distance image number.
    thi_distance = indx(3); %find the third smallest distance image number.
    %get the original images for the test image and similar training images.
    image_test = imread([testpath test_filenames(n).name]);
    image_1 = imread([trainpath train_filenames(min_distance).name]);
    image_2 = imread([trainpath train_filenames(sec_distance).name]);
    image_3 = imread([trainpath train_filenames(thi_distance).name]);
    %show the four images.
    figure
    subplot(1,4,1); imshow(image_test); title(test_filenames(n).name); xlabel('k=10');
    subplot(1,4,2); imshow(image_1); title('Recognition image1');
    subplot(1,4,3); imshow(image_2); title('Recognition image2');
    subplot(1,4,4); imshow(image_3); title('Recognition image3');
end
```

Hint (3):

we can use matrix (A'*A) to replace inner-product matrix (A*A'). the covariance matrix (A'*A) is 10000x10000, it is too large. Matrix (A*A') is just 135x135. Because the number of training image is less than the vector dimensional (M<N^2). And the useful eigenvectors are M-1 (other eigenvectors' eigenvalues are zero). The result is the same. The calculation is below:

$$
\begin{aligned}
& C' \cdot e_i = \lambda \cdot e_i && | \quad C' = \Phi^T \cdot \Phi \\
\Leftrightarrow \quad & \Phi^T \cdot \Phi \cdot e_i = \lambda \cdot e_i && | \quad \text{multiply from the left with } \Phi \\
\Leftrightarrow \quad & \Phi \cdot \Phi^T \cdot \Phi \cdot e_i = \lambda \cdot \Phi \cdot e_i && | \quad C = \Phi \cdot \Phi^T \\
\Leftrightarrow \quad & C \cdot \Phi \cdot e_i = \lambda \cdot \Phi \cdot e_i && | \quad \text{define } v_i = \Phi \cdot e_i \\
\Leftrightarrow \quad & C \cdot v_i = \lambda \cdot v_i
\end{aligned}
$$

(Figure 1.6) [2]

Task-2: DLT for 2-view homography estimation

1.

The translate equation is:

$$\begin{matrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & 1 \end{matrix} * \begin{matrix} u \\ v \\ 1 \end{matrix} = \begin{matrix} u' \\ v' \\ 1 \end{matrix}$$

$$\begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} \Leftrightarrow \begin{cases} uh_1 + vh_2 + h_3 = u' \\ uh_4 + vh_5 + h_6 = v' \\ uh_7 + vh_8 + h_9 = 1 \end{cases} \Leftrightarrow \begin{bmatrix} 0 & 0 & 0 & -u & -v & -1 & v'u & v'v & v' \\ u & v & 1 & 0 & 0 & 0 & -u'u & -u'v & -u' \\ -v'u & -v'v & -v' & u'u & u'v & u' & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow \mathbf{Ah} = \mathbf{0}$$

(Figure 2.1) [3]

The code for the DLT function is below:

```
function H=DLT(u2Trans, v2Trans, uBase, vBase)
% Computes the homography H applying the Direct Linear Transformation
% The transformation is such that
% Hp = p'
% H*(uBase, vBase, 1)'=(u2Trans , v2Trans, 1)' %
% CHAOYUN GONG, 22/Apr/2018
A = zeros(12,9);
for i=1:6 %identify the caculation matrix.
A((i-1)*2+1:(i-1)*2+2,1:9) = [0, 0, 0, -uBase(i), -vBase(i), -1, v2Trans(i)*uBase(i), v2Trans(i)*vBase(i),
v2Trans(i);
uBase(i), vBase(i), 1, 0, 0, 0, -u2Trans(i)*uBase(i), -u2Trans(i)*vBase(i), -u2Trans(i)];
end

[U,S,V]=svd(A); %Orthogonal decomposition, using svd()function to calculate the singular value.
h= V(:,9); %the the last column of singular value matrix is the matrix H.
h=h./h(9); %Normalized, the last element of matrix H is 1.
H= reshape(h,3,3)'; %change the one column to 3x3 matrix.
```

The main code is below:

```
left = imread('Left.jpg');
right = imread('Right.jpg');
figure; imshow(left); title('left');
[uBase,vBase] = ginput(6);    %get the 6 input points for left image.
figure;imshow(right);title('Right');
[u2Trans,v2Trans] = ginput(6); %get the corresponding 6 input points for right image.
H=DLT(u2Trans, v2Trans, uBase, vBase); %call function DLT()
%test the matrix H.
figure(1);imshow(left);hold on;plot(uBase,vBase,'r.'); %show the six points in left image.
[Px, Py] = ginput(1);    %input one new point for test.
P = [Px, Py];hold on;plot(P(1),P(2),'gp','MarkerSize', 12); %show the test point in left image.
x1 = P(:, 1); %get the test point' coordinates.
y1 = P(:, 2);
p1 = [x1'; y1'; ones(1, length(x1))]; %change the Two-dimensional coordinates to Homogeneous
coordinates.
q1 = H*p1;    %use matrix H to translate the point.
q1 = q1./[q1(3, :); q1(3,:); q1(3, :)]; %Normalized.
p2 = q1'; %change from one column to one row.
figure(2)
imshow(right);hold on;plot(u2Trans,v2Trans,'r.'); %show the six input points in right image.
hold on;
plot(p2(1), p2(2), 'gp', 'MarkerSize', 12); %show the corresponding point for the test point in right image.
```

2.

Figure 2.2 is the left image, the six red point is the input points, the green pentagram is the input test point. Figure 2.3 is the right image, the six red point is the corresponding input points, the green pentagram is the output of the corresponding input test point by using matrix H.

(Figure 2.2)

(Figure 2.3)

The test shows that the matrix H is correct.

3.

3x3 homography matrix H is below:

16.483568498983605	-4.579924634469490	-1.570808294858930e+03
4.424622979354007	2.683353578780927	-4.094150616731278e+02
0.028382288348058	-0.012379785332735	1

The translation is from left image to right image, equation is p'(right) = H*p(left).

The minimally required points are four. Because each corresponding point have two equations. There are 8 unknow elements in matrix H, four points have eight equations. And eight equations can solve eight unknow elements.

Reference list

[1] Turk, M. & Pentland, A. (1991) Face Recognition using Eigenfaces. IEEE.

[2] zouxy09 (2015). Face recognition-eigenface. CSDN. Link: https://blog.csdn.net/zouxy09/article/details/45276053

[3] Opsahl, T. Lecture 4.3 Estimating homographies from feature correspondences. http://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/forelesninger/lecture_4_3-estimating-homographies-from-feature-correspondences.pdf