ENGN4528 Computer Vision Clab3 Report

Zhipeng Bao u6600985

April 2018

Task 1: Face Recognition using eigenface technique

Step 1: Have a look at the training and test pictures

For this step, I just ran the given $viewyalefaces()$ function. The following 3 figures show the results of the training images and test images. Figure 1 shows a single training image, Figure 2 shows 2 sets of training images of the same person and Figure 3 shows the 10 test images.

Figure 1: A single image of the training set

Figure 2: Two sets of images of the training corpus

Figure 3: Test images

Step 2: Train eigenfaces

For this step, I first re-aligned and restored the training images. Figure 4 shows one of the re-aligned images. Then I read in all the images and for each single

image, I treat it as a fixed length 1-D vector. Thus, after reading all the images we can get a training matrix

$$M \in R^{l \times c}$$

in which l is the length of the vector and c is the amount of the training images.

Figure 4: One re-aligned and resized training image

The following codes show the process of getting training matrix. Besides, to make the result more accurate, I remove the DC components of the training matrix.

```
%step1: have a look at the training images
dir_name = './trainingset/';
dir_lst = dir(dir_name);
l = length(dir_lst);

top = 10;

%step2: train eigenfaces
train_matrix = [];
count = 0;
%get the train matrix
for i = 1:l
   name = dir_lst(i).name;
   if name(1)=='s'
       img = imread([dir_name,name]);
       [a,b] = size(img);
       img = reshape(img,[a*b 1]);
       img = double(img);
       train_matrix = [train_matrix,img];
       count = count+1;
   end
end

%remove DC Component
mean_face = mean(train_matrix,2);
train_matrix = train_matrix - repmat(mean_face,1,count);
```

Then goes to the most important parts of the this task: find the principal components of the training faces.

After loading images, we get a training matrix $M \in R^{45045 \times 135}$. To get the principal components of it, we need to calculate the eigen value and eigen vector

of $M * M'$. It is quite a complicated task for Matlab. As we just need several "Principal Component", we can play a trick on it. That is we can calculate the eigen values and eigen vectors of $M' * M$ instead. Next, I will prove $M' * M$ and $M * M'$ have the same eigen values and their eigen vectors have some relations.

Let's assume V is the eigen vector of MM' and thus:

$$MM'V = aV$$

Left multiply M' in both sides, then we can get:

$$M'MM'V = M'aV = aM'V$$

$$M'M(M'V) = a(M'V)$$

So that a is also the eigen value of $M'M$ and the corresponding eigen vector is $M'V$.

Based on this, I conduct the following $[eig_face] = eigenfaces(C, k)$ function to get the eigenfaces.

```
function [eig_face] = eigenfaces(C,k)
%% returns top K eigenfaces

% calculate eigenvalues
[V,D] = eig(C);
%arrange the eigen values in descend order.
[eig_val,ind] = sort(diag(D),'descend');
eig_vec = V(:,ind);
% take top k vectors
if size(eig_vec,2) ≥ k
    eig_face = eig_vec(:,1:k);
else
    error('Not enough eigenvectors - check image matrix');
end
```

Figure 5 and Figure 6 show the top 5 and top 10 eigenfaces.

Find the most similar faces for test images

For this task, I first project all the training images to the eigenface space. Then for each test image, I also project it to the same space and calculate the Euclidean distance of the test image and all the training image points. Consider the three nearest points as the most similar images from the test image. The codes are listed below:

```
%step 3: read in test images
test_name = './testset/';
test_dir = dir(test_name);
ll = length(test_dir);

%get the PCA coefficients of training images
PCA_coe = zeros(count,top);
```

Figure 5: Top 5 eigenfaces

Figure 6: Top 10 eigenfaces

```
for i = 1:count
    PCA_coe(i,:) = ...
        (train_matrix(:,i)'*eigen_face)/(eigen_face'*eigen_face);
end


%get similar faces
for i = 1:ll
    name = test_dir(i).name;
    if name(1)=='t'
        testg = imread([test_name,name]);
        figure;
        subplot(2,2,1);
        imshow(testg,[]);
        title('original imgae');
        testg = double(reshape(testg,[a*b,1]));
        testg = testg-mean_face;
        coordinates = testg'*eigen_face/(eigen_face'*eigen_face);
        distance = repmat(coordinates,count,1);
        distance = distance - PCA_coe;
        distance = distance.^2;
        distance = sum(distance,2);
        pos1 = find(distance == min(distance));
        pos1 = pos1(1);
        distance(pos1) = 9e+20;
        pos2 = find(distance == min(distance));
        pos2 = pos2(1);
        distance(pos2) = 9e+20;
        pos3 = find(distance == min(distance));
        pos3 = pos3(1);
        subplot(2,2,2);
        imshow(reshape(train_matrix(:,pos1)+mean_face,[a,b]),[]);
        title('The 1st similar image')
        subplot(2,2,3);
        imshow(reshape(train_matrix(:,pos2)+mean_face,[a,b]),[]);
        title('The 2nd similar image')
        subplot(2,2,4);
        imshow(reshape(train_matrix(:,pos3)+mean_face,[a,b]),[]);
        title('The 3rd similar image')

    end
end
```

Figure 7 to Figure 16 show the result of top 5 PC Analysis and Figure 17 to Figure 26 top 10 PC Analysis.

For the top 5 analysis, the overall recognition rate is **90%**. All the images have the correct corresponding images except test image 8 which has totally wrong corresponding images. For the top 10 analysis, the overall recognition rate is **93.3%** The two mismatch pairs happened also on test image 8. Having a look at that specific result, we can see that the reason is that the two person is kind of similar in hairstyle and age. Even I saw them the first time, I thought they were the same person.

So personally speaking, I think the codes and results reflect the real ability of PCA in face recognition.

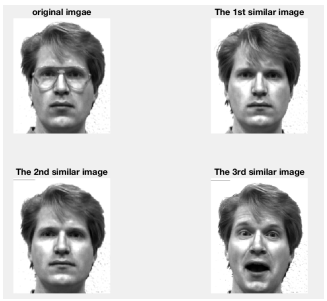

Figure 7: Top 5 PCA on test 1

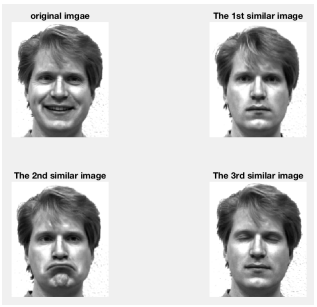

Figure 8: Top 5 PCA on test 2

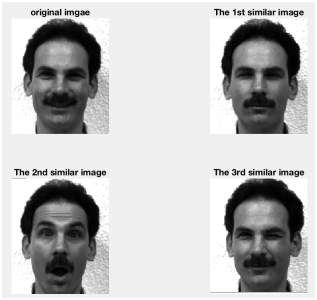

Figure 9: Top 5 PCA on test 3

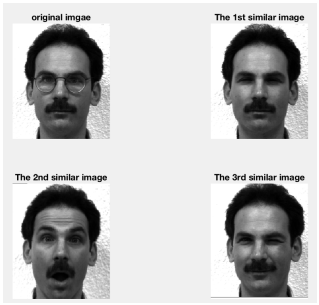

Figure 10: Top 5 PCA on test 4

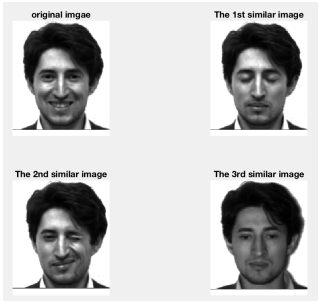

Figure 11: Top 5 PCA on test 5

Figure 12: Top 5 PCA on test 6

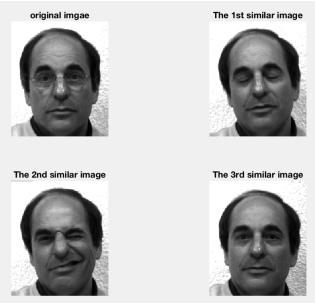

Figure 13: Top 5 PCA on test 7

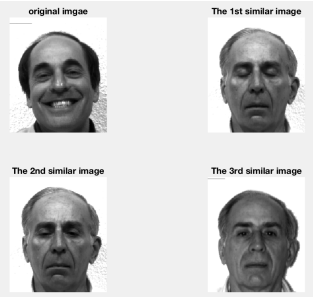

Figure 14: Top 5 PCA on test 8

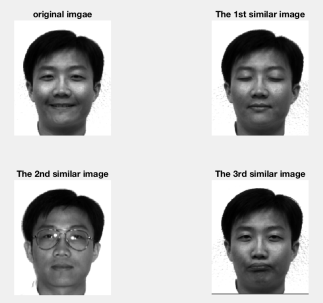

Figure 15: Top 5 PCA on test 9

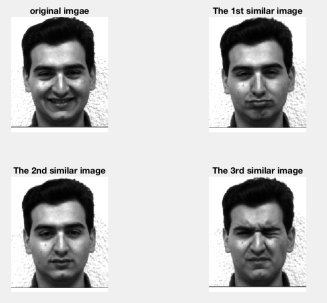

Figure 16: Top 5 PCA on test 10

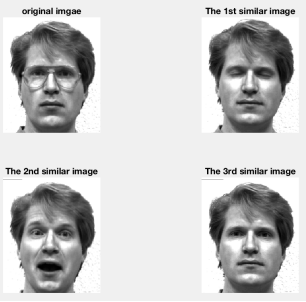

Figure 17: Top 10 PCA on test 1

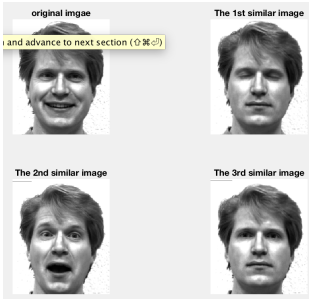

Figure 18: Top 10 PCA on test 2

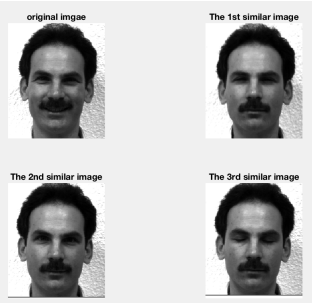

Figure 19: Top 10 PCA on test 3

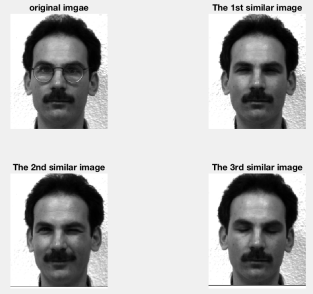

Figure 20: Top 10 PCA on test 4

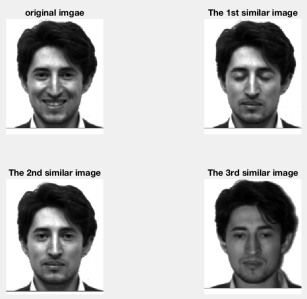

Figure 21: Top 10 PCA on test 5

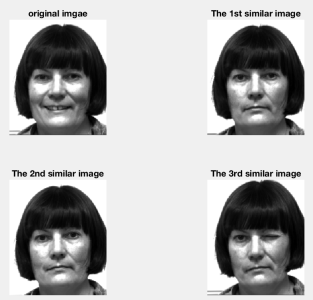

Figure 22: Top 10 PCA on test 6

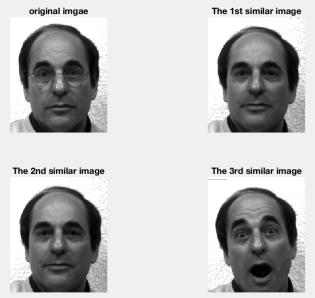

Figure 23: Top 10 PCA on test 7

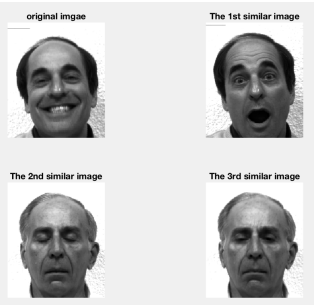

Figure 24: Top 10 PCA on test 8

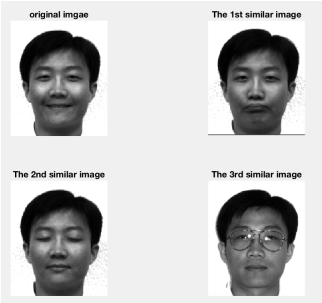

Figure 25: Top 10 PCA on test 9

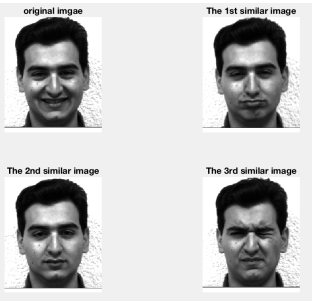

Figure 26: Top 10 PCA on test 10

By the way, the most important thing in this task is to re-align the images. When I align the data by myself, the recognition rate stays at about 70% to 80%. When I align them with a Matlab script which means each image has the same standard format. In this situation, the recognition rate goes up to more than 90%.

Task 2: DLT for 2-view homography estimation

Find 6 pairs of corresponding points

Using Matlab inbuilt function *ginput*(), we can get 6 pairs of corresponding points. Then I restore the data in order to make it convenient to debug. the locations of points 1 to points 6 are from upper left to the lower right.

The codes and results are listed below.

Codes:

```
close all;
clear all;
clc;

im_left = imread('Left.jpg');
im_right = imread('Right.jpg');
figure;
subplot(1,2,1);
imshow(im_left);
title('Left Image');
subplot(1,2,2);
imshow(im_right);
title('Right Image');
hold on;
[x,y] = ginput(12);
```

Results:

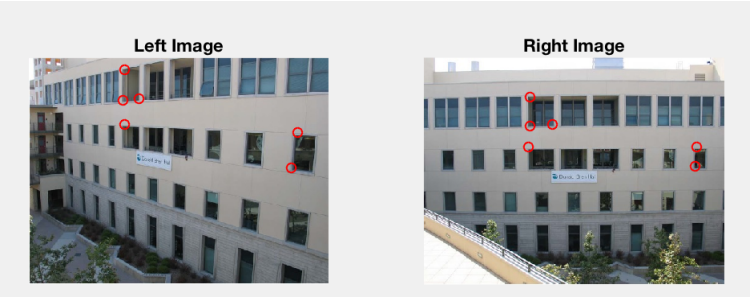

Figure 27: Corresponding points of given images

Step 2: Implement DLT.m

The most important of DLT is to solve H from the lecture we learned.

First we know:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

After expose this equation, we can get following equations:

$$x'_i \times (h_{20} \times x_i + h_{21} \times y_i + h_{22}) = h_{00} \times x_i + h_{01} \times y_i + h_{02}$$
$$y'_i \times (h_{20} \times x_i + h_{21} \times y_i + h_{22}) = h_{10} \times x_i + h_{11} \times y_i + h_{12}$$

Then, the problem become to get the eigenvector that belongs to the smallest eigenvalue of matrix A.

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ \dots & & & & & & & & \\ x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{bmatrix}$$

Based on these knowledge from the lecture, I implement the DLT function. The codes are listed below:

```
1
2 function H=DLT(u2Trans, v2Trans, uBase, vBase)
3 % Computes the homography H applying the Direct Linear ...
      Transformation
4 % The transformation is such that
5 % p = H p' , i.e.,:
6 % (uBase, vBase, 1)'=H*(u2Trans , v2Trans, 1)' %
7 % INPUTS:
8 % u2Trans, v2Trans - vectors with coordinates u and v of the ...
      transformed image point (p')
```

```
% uBase, vBase - vectors with coordinates u and v of the ...
    original base image point p
%
% OUTPUT
% H - a 3x3 Homography matrix %
% Zhipeng Bao 27th Apr, 2018
l = length(uBase);
% assume the length of u2Tran, v2Tran, uBase, vBase are the ...
    same, or we need
% to write some codes for anomaly detection.
A = zeros(2*l,9);
%Form the matrix A;
for i = 1:l
    x = u2Trans(i);
    y = v2Trans(i);
    xp = uBase(i);
    yp = vBase(i);
    A(2*i-1,:) =  [x,y,1,0,0,0,-xp*x,-xp*y,-xp];
    A(2*i,:) = [0,0,0,x,y,1,-yp*x,-yp*y,-yp];
end
%svd decompose
[S,D,V] = svd(A);
[w,h] = size(V);
%get the last column of V matrix which is H
H = V(:,end);
%reshape H to 3X3
H = reshape(H,[3 3])';
%Normalize H
H = H/H(3,3);
```

Step 3: Test results

By running the DLT function, we can get the result of H. It displays as the following:

$$H = \begin{bmatrix} 0.2572 & 0.0712 & 75.1127 \\ -0.1244 & 0.9802 & -21.5698 \\ -0.0013 & 0.0004 & 1.0000 \end{bmatrix}$$

To verify the H matrix, I find the projective image from the right image based on H and show the result below.

The first two images shared almost the same perspective which proves the correctness of the codes.

how many points are minimally required in order to apply DLT to estimate a Homography

At list 4 pairs of points are needed to apply DLT algorithm. The reason is that the matrix H has 8 degrees of freedom and each pair of points can provide two linear independent equations. Thus, to solve all the 8 degrees, at least 4 pairs of points are needed.

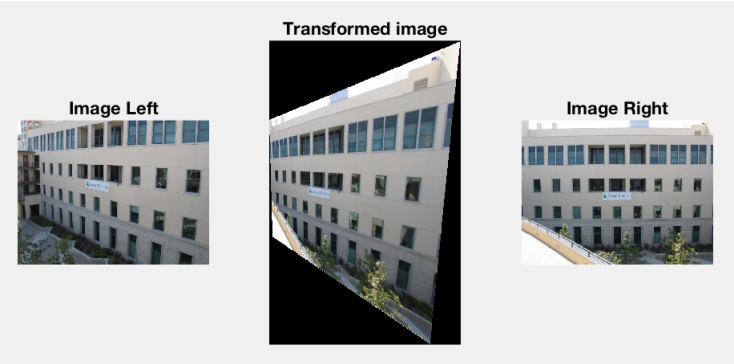

Figure 28: Transformed result

I also test the 4 points in my Matlab experiments, I firstly select point 1,3,4,6 to get the result. It shows below:

$$H = \begin{bmatrix} 0.2765 & -0.0097 & 76.7986 \\ -0.1067 & 0.8509 & -20.9564 \\ -0.0012 & 0.0002 & 1.0000 \end{bmatrix}$$

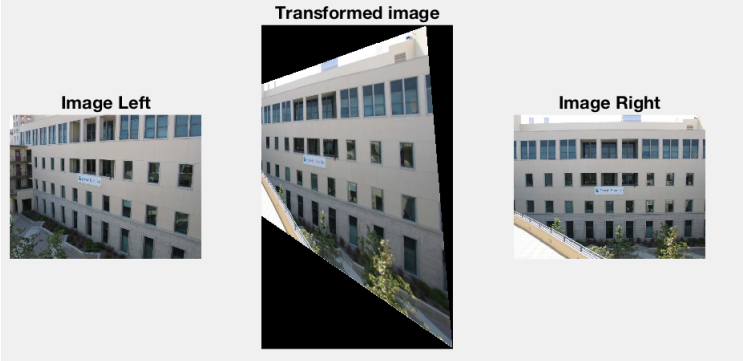

Figure 29: Result based on pairs 1,3,4,5

The result is quite similar to the 6-pair one. So it is proved 4 pairs of points is enough to estimate matrix H.

However, to keep the DLT method robust, we need to give more points. For example, if I use the pairs [1,2,3,5], in which points 1,2,3 are concentrating on the same area, to estimate matrix H, the result is not that good. Here are the results.

$$H = \begin{bmatrix} 0.3939 & 0.2485 & 50.3797 \\ -0.1054 & 1.0403 & -32.1165 \\ -0.0015 & 0.0019 & 1.0000 \end{bmatrix}$$

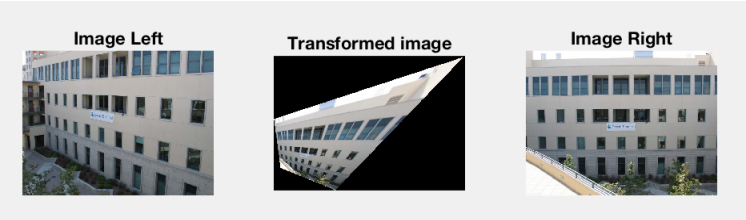

Figure 30: Result based on pairs 1,2,3,5

In conclusion, we need at least 4 pair of points to apply DLT method. But to get an effective DLT result, the locations of the pairs should be decentralized. With more pairs of points, the result will be more accurate.

Attached codes

For this part, I attach the main codes of each task and some other supportive codes here.

Task1.m

```
clear all;
close all;
clc;

%step1: have a look at the training images
dir_name = './trainingset/';
dir_lst = dir(dir_name);
l = length(dir_lst);

top = 10;

%step2: train eigenfaces
train_matrix = [];
count = 0;
%get the train matrix
for i = 1:l
   name = dir_lst(i).name;
   if name(1)=='s'
       img = imread([dir_name,name]);
       [a,b] = size(img);
       img = reshape(img,[a*b 1]);
       img = double(img);
       train_matrix = [train_matrix,img];
       count = count+1;
   end
end

%remove DC Component
mean_face = mean(train_matrix,2);
```

```
train_matrix = train_matrix - repmat(mean_face,1,count);

%tricky: A'A and AA' have the same eigen value
%and their eigen vector can be transformed.
A = train_matrix'*train_matrix;
eigen_face = eigenfaces(A,top);

%actual eigen vectors(faces).
eigen_face = train_matrix*eigen_face;

%show eigen faces
figure;
for i = 1: top
   subplot(2,ceil(top/2),i);
   imshow(reshape(eigen_face(:,i),[a,b]),[]);
end

%step 3: read in test images
test_name = './testset/';
test_dir = dir(test_name);
ll = length(test_dir);

%get the PCA coefficients of training images
PCA_coe = zeros(count,top);
for i = 1:count
    PCA_coe(i,:) = ...
        (train_matrix(:,i)'*eigen_face)/(eigen_face'*eigen_face);
end


%get similar faces
for i = 1:ll
    name = test_dir(i).name;
    if name(1)=='t'
        test_img = imread([test_name,name]);
        figure;
        subplot(2,2,1);
        imshow(test_img,[]);
        title('original imgae');
        test_img = double(reshape(test_img,[a*b,1]));
        test_img = test_img-mean_face;
        coordinates = test_img'*eigen_face/(eigen_face'*eigen_face);
        distance = repmat(coordinates,count,1);
        distance = distance - PCA_coe;
        distance = distance.^2;
        distance = sum(distance,2);
        pos1 = find(distance == min(distance));
        pos1 = pos1(1);
        distance(pos1) = 9e+20;
        pos2 = find(distance == min(distance));
        pos2 = pos2(1);
        distance(pos2) = 9e+20;
        pos3 = find(distance == min(distance));
        pos3 = pos3(1);
        subplot(2,2,2);
        imshow(reshape(train_matrix(:,pos1)+mean_face,[a,b]),[]);
        title('The 1st similar image')
```

```
86          subplot(2,2,3);
87          imshow(reshape(train_matrix(:,pos2)+mean_face,[a,b]),[]);
88          title('The 2nd similar image')
89          subplot(2,2,4);
90          imshow(reshape(train_matrix(:,pos3)+mean_face,[a,b]),[]);
91          title('The 3rd similar image')
92
93      end
94  end
```

Task2.m

```
1   close all;
2   clear all;
3   clc;
4
5   im_left = imread('Left.jpg');
6   im_right = imread('Right.jpg');
7   %{
8   figure;
9   subplot(1,2,1);
10  imshow(im_left);
11  title('Left Image');
12  subplot(1,2,2);
13  imshow(im_right);
14  title('Right Image');
15  hold on;
16  [x,y] = ginput(12);
17  %}
18  %load data
19  load t2.mat;
20  a = [1,3,5,7,9,11];
21  b = [2,4,6,8,10,12];
22  x1 = x(a);
23  x2 = x(b);
24  y1 = y(a);
25  y2 = y(b);
26  H = DLT(x2,y2,x1,y1);
27  disp('H = ');
28  disp(H);
29
30
31  figure;
32  tform = projective2d(H');
33  imageTrans = imwarp(im_right,tform);
34  subplot(1,3,1);
35  imshow(im_left);
36  title('Image Left');
37  subplot(1,3,2);
38  imshow(imageTrans,[]);
39  title('Transformed image');
40  subplot(1,3,3);
41  imshow(im_right);
42  title('Image Right');
```

DLT.m

```
function H=DLT(u2Trans, v2Trans, uBase, vBase)
% Computes the homography H applying the Direct Linear ...
    Transformation
% The transformation is such that
% p = H p' , i.e.,:
% (uBase, vBase, 1)'=H*(u2Trans , v2Trans, 1)' %
% INPUTS:
% u2Trans, v2Trans - vectors with coordinates u and v of the ...
    transformed image point (p')
% uBase, vBase - vectors with coordinates u and v of the ...
    original base image point p
%
% OUTPUT
% H - a 3x3 Homography matrix %
% Zhipeng Bao 27th Apr, 2018
l = length(uBase);
% assume the length of u2Tran, v2Tran, uBase, vBase are the ...
    same, or we need
% to write some codes for anomaly detection.
A = zeros(2*l,9);
%Form the matrix A;
for i = 1:l
    x = u2Trans(i);
    y = v2Trans(i);
    xp = uBase(i);
    yp = vBase(i);
    A(2*i-1,:) =  [x,y,1,0,0,0,-xp*x,-xp*y,-xp];
    A(2*i,:) = [0,0,0,x,y,1,-yp*x,-yp*y,-yp];
end
%svd decompose
[S,D,V] = svd(A);
[w,h] = size(V);
%get the last column of V matrix which is H
H = V(:,end);
%reshape H to 3X3
H = reshape(H,[3 3])';
%Normalize H
H = H/H(3,3);
```