ENGN4528 Computer Vision Clab2 Report

Zhipeng Bao u6600985

March 2018

Task 1: Implement your own Harris Corner Detector

Step 1 & Step 2: Read, understand and Complete the Matlab codes.

For this task, we are required to finish the Harris Corner Detector based on the existing codes. From the lecture, we know that there are two main process in the Harris Corner detection algorithm: *Find the points with large corner response function R* and *Find the local maximum value of the response*. By reading the half-finished codes, I found that the missing parts are exactly the two main process.

For the first part, we need to get the function R which equals:

$$R = det(M) - k * (trace(M))^2 \tag{1}$$

In the equation, k is a constant value varies from 0.04 to 0.06. We set it as a most widely used value: 0.04. And the matrix S has the following defination:

$$\begin{bmatrix} \sum (I_x(x,y))^2 & \sum I_x(x,y)I_y(x,y) \\ \sum I_x(x,y)I_y(x,y) & \sum (I_y(x,y))^2 \end{bmatrix}$$

For the next main process, we need to find the local maximum which can be calculated by Matlab inbuilt function ordfit2(). Then we need to find the ones that are larger than threshold.

I made a lot of comments in my codes. So step 3 shows the details of the whole process.

Step 3: Comment on the codes.

The codes and the comments are shown below:

```
function [rws,cols] = my_corner_detector(bw, sig, thre,sz)
    %Harris Corner Detector
    % parameters
    sigma = sig;
    thresh = thre;
    sze = sz;
```

```
7       disp = 0;
8
9       bw = double(bw); %transform image to double
10      [line,column] = size(bw);%get the size
11
12      dy = [-1 0 1; -1 0 1; -1 0 1]; % derivative mask in y direction
13      dx = dy'; % dx and dy have symmetrical structure
14
15      % image derivatives
16      Ix = conv2(bw,dx,'same'); % derivatives in x direction
17      Iy = conv2(bw,dy,'same'); % derivatives in y direction
18
19
20      %need codes here
21      g = fspecial('gaussian',max(1,fix(6*sigma)),sigma); % form a ...
            gaussian kernel
22      Ix2 = conv2(Ix.^2,g,'same');%Smoothed image derivatives and ...
            calculate M(1,1)
23      Iy2 = conv2(Iy.^2,g,'same'); %calculate M(1,2) and M(2,1)
24      Ixy = conv2(Ix.*Iy,g,'same'); %calculate M(2,2)
25
26      Ix2 = padarray(Ix2, [(sze-1)/2, (sze-1)/2]); %make it ...
            convinient for later work
27      Iy2 = padarray(Iy2, [(sze-1)/2, (sze-1)/2]); %make it ...
            convinient for later work
28      Ixy = padarray(Ixy, [(sze-1)/2, (sze-1)/2]); %make it ...
            convinient for later work
29
30      %compute the cornerness
31      k = 0.04; % range of k: 0.04¬0.06, 0.04 is a widely used ...
            value for Haris Detector
32
33      for i = 1:line
34          for j=1:column
35              xx = sum(sum(Ix2(i:i+sze-1,j:j+sze-1))); %calculate ...
                    sum of Ix^2
36              yy = sum(sum(Iy2(i:i+sze-1,j:j+sze-1))); %calculate ...
                    sum of Iy^2
37              xy = sum(sum(Ixy(i:i+sze-1,j:j+sze-1))); %calculate ...
                    sum of Ixy
38              M = [xx,xy;xy,yy]; %represent M
39              cornerness(i,j) = det(M)-k*trace(M)^2; %calculate ...
                    det(M) - k*trace(M)^2, which is the cornerness
40          end
41      end
42
43      max_value = ordfilt2(double(cornerness), sze^2, ones(sze)); ...
            %find local maximum points
44
45      cornerness = (double(cornerness) == max_value) & ...
            (double(cornerness) > thresh); %find local max points ...
            and the ones bigger than thresh
46      [rws, cols] = find(cornerness>0); %get result
47  end
```

Step 4 - Step 6: Display and compare the results.

Fig 1 shows the result on lenna with different threshold. We can see that the corners get fewer with a bigger threshold. And the sharper corners left with the bigger threshold.

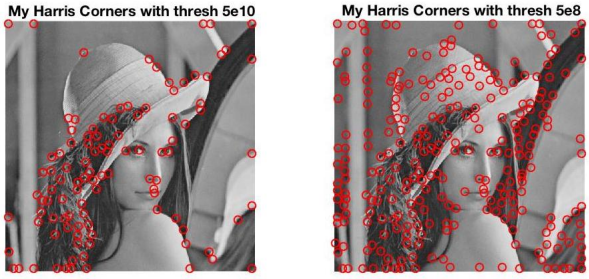

Figure 1: Result for Lenna with different threshold

Figure 2 to figure 5 show the results with $thresh = 5e10$ and compare them with Matlab inbuilt corner functions. We can see there are some different points between my function and Matlab inbuilt function. I think the reason is that the threshold and sigma are different. Besides, the inbuilt function may also use some optimal methods.

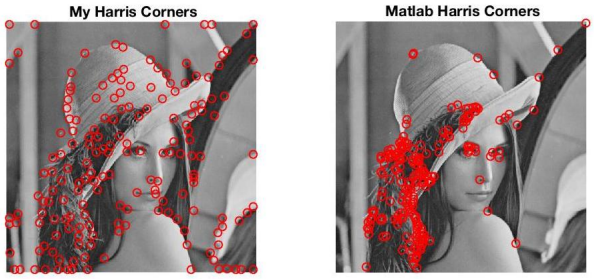

Figure 2

Taks 2: Implement your own k-means clustering function, and use it for colour image segmentation

Step 1 & Step 2: Read in and convert the image.

The codes for the first three steps are given. The only difference is for the uint 16 image. The codes are in the following.

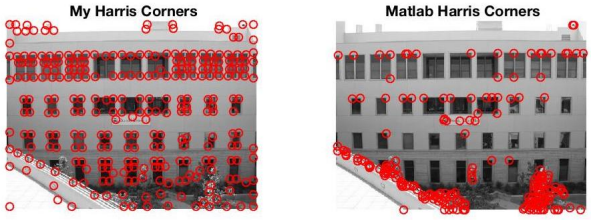

Figure 3

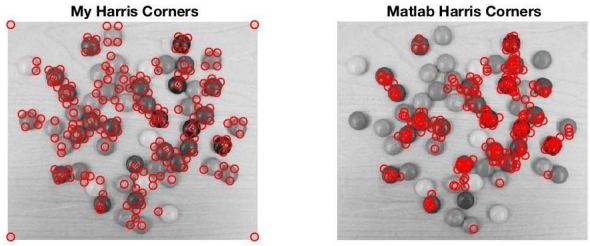

Figure 4

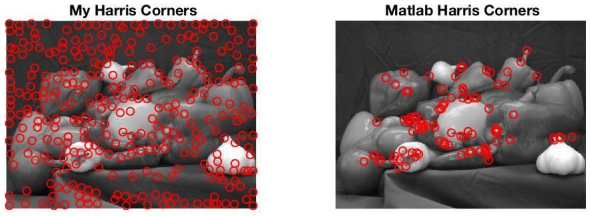

Figure 5

```
%Step 1: Read input image
path1 = './images/Peppers.png';
img1 = imread(path1);
img1 = uint8(img1/2^8);
imshow(img1);
```

Step 3: Form 5-dimensional feature vector.

I checked the given codes for this problem. It is correct.

Step 4 : Implement your own k-means algorithm.

First, the given half-finished codes have one mistake. In the 17^{th} line, the codes should be:

```
%distances = zeros(ndata,ndims); wrong!
distances = zeros(ndata,nc); % right
```

Then there are two parts of codes that I need to fulfill. The first thing is to assign each point to the nearest cluster center. The second is to update the membership assignment. Actually, these two thing is a whole part. I divide it into another two parts: Calculate the distance between each point and each cluster center. The missing codes are in the following:

```
% - calculate the distance from each point to each cluster ...
    center.
cluster_position = cluster_stats(:,2:end);
for i = 1:nc
    a = data-repmat(cluster_position(i,:),ndata,1);
    distances(:,i) = sqrt(sum(a.^2,2));
end

%%update the membership assignment, i.e., update the ...
    data_clusters with current values.
for i = 1:ndata
    data_clusters(i) = find(distances(i,:) == ...
        min(distances(i,:)),1);
end
```

Step 5: Display the results.

For the result of my k-mean algorithm, I firstly set k to 5. The result with different factor for different images are shown in Figure 6 to Figure 15.

From the result I found no difference among the different factors. However, I read the codes carefully and think about the function of the factor. In my opinion, the factor is used to times coordinates of each feature vector, and make the values bigger than before. When using a bigger factor, more values will be set as maximum, then the boundary will be influenced.

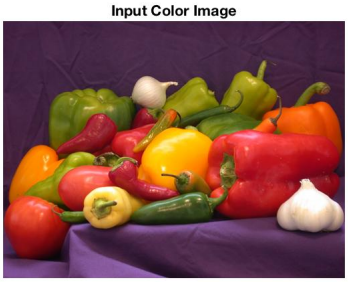

Figure 6: The first color image.

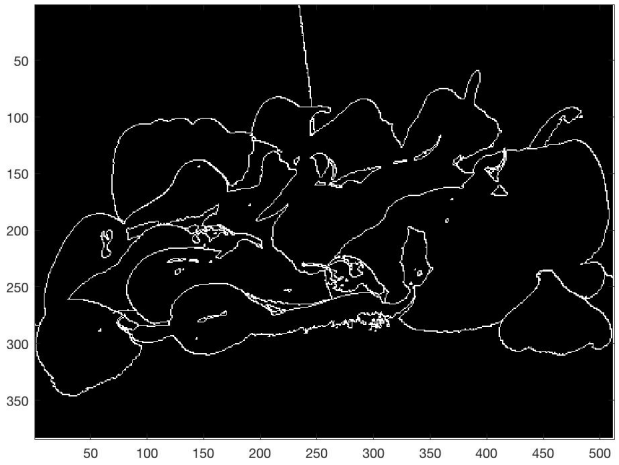

Figure 7: The boudnary of the first image.

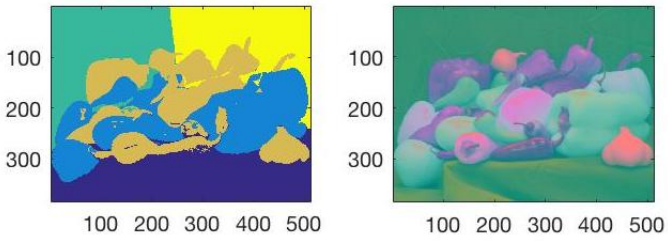

Figure 8: The 5-means result with factor = 1

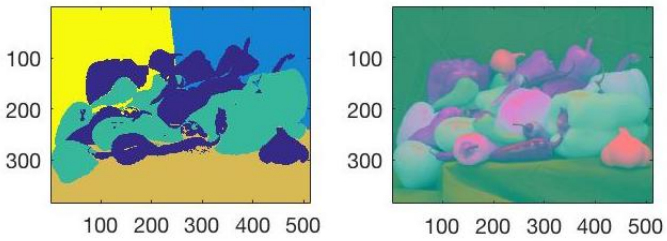

Figure 9: The 5-means result with factor = 0.1

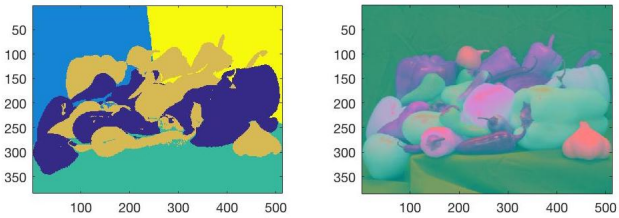

Figure 10: The 5-means result with factor = 10

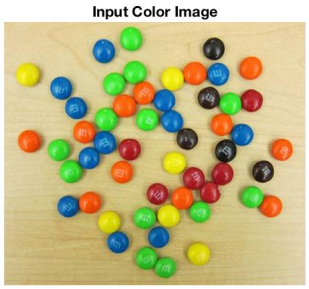

Figure 11: The second image.

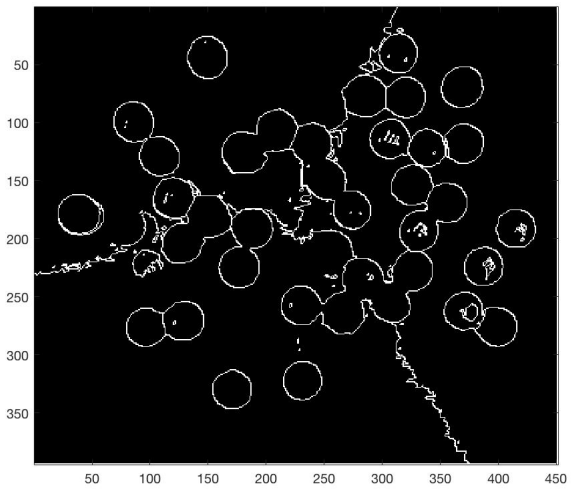

Figure 12: The boudnary of the second image

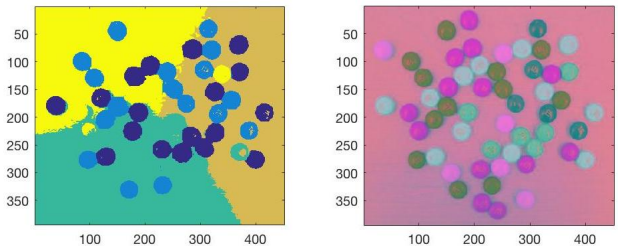

Figure 13: The 5-means result with factor = 1

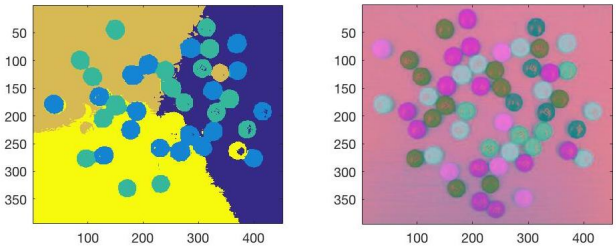

Figure 14: The 5-means result with factor = 10

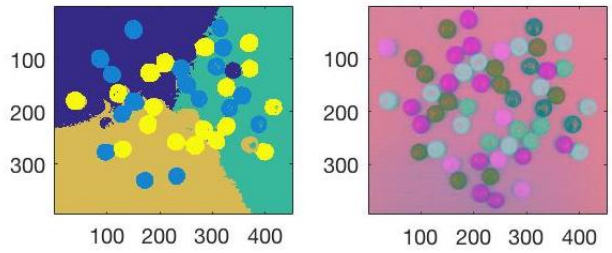

Figure 15: The 5-means result with factor = 0.1

I think maybe the reason is that the number of the classes is too small to show the difference. So I change the k to 10 and redo the experiment.

The result of the MMS are shown as follows:

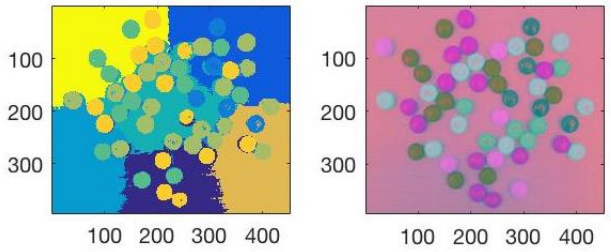

Figure 16: The 10-means result with factor = 10

This time, we can see the clear difference between different factor. It also prove my assumption of the factor.

Task 3: Play with SIFT code, and implement an image match algorithm

Step 1: Read in image and rotate it.

This task is quite simple. Here I show the results. The codes can be found in the next part (attached codes.)

Step 2: Read, understand and test SIFT function.

I read the codes of $sift.m$ carefully and understand how it works. The first step is to restore the file in a pgm form. Then, use external sift binary file to calculate the sift values (in tmp.key file). The final step is to transform the values to descriptors and locs. The first two dimensions of locs are the column and line number of the points. Then the scale and the ori.

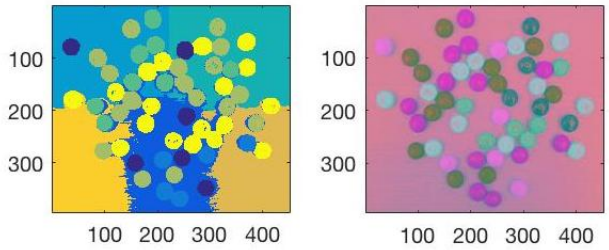

Figure 17: The 10-means result with factor = 0.1

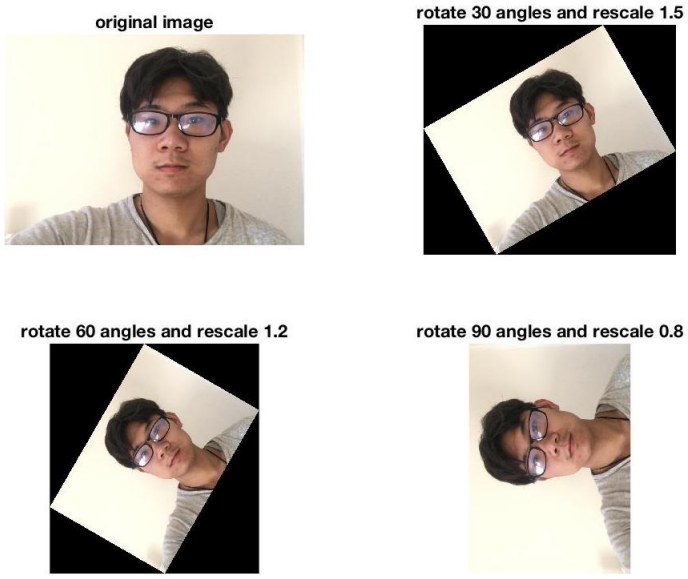

Figure 18: The original images and rotated and rescaled images.

However, although I have two operation systems myself, I cannot run the codes. For my mac system, it shows the binary file cannot be run and for my windows system, win32 file also failed. So I changed the codes a little and use a ubuntu server to finish the sift file. I divide the codes for two parts: use external method to calculate the related features.

The first part of the codes are as follows:

```
1       fin = imread('face_02_u6600985.jpg');
2       imwrite(fin,'im0.pgm');
```

For the second part, I rewrite the function below:

```
1  function [descriptors, locs] = my_sift(image,keys)
2  % Summarize and reduce some parts of the given sift function.
3  % The reason is that my mac cannot do the sift and my windows ...
       does not
4  % support win32.
5  % Thus I use a ubuntu server to finish sift and this function is ...
       used to
6  % get the sift points.
7  %
8  % image: XXX.jpg, original image.
9  % keys: the XXX.key file got from the sift.
10 [rows, cols] = size(image);
11
12 % Open tmp.key and check its header.
13 g = fopen(keys, 'r');
14 if g == -1
15     error('Could not open file tmp.key.');
16 end
17 [header, count] = fscanf(g, '%d %d', [1 2]);
18 if count ≠ 2
19     error('Invalid keypoint file beginning.');
20 end
21 num = header(1);
22 len = header(2);
23 if len ≠ 128
24     error('Keypoint descriptor length invalid (should be 128).');
25 end
26
27 % Creates the two output matrices (use known size for efficiency)
28 locs = double(zeros(num, 4));
29 descriptors = double(zeros(num, 128));
30
31 % Parse tmp.key
32 for i = 1:num
33     [vector, count] = fscanf(g, '%f %f %f %f', [1 4]); %row col ...
           scale ori
34     if count ≠ 4
35         error('Invalid keypoint file format');
36     end
37     locs(i, :) = vector(1, :);
38
39     [descrip, count] = fscanf(g, '%d', [1 len]);
```

```
    if (count ≠ 128)
        error('Invalid keypoint file value.');
    end
    % Normalize each input vector to unit length
    descrip = descrip / sqrt(sum(descrip.^2));
    descriptors(i, :) = descrip(1, :);
end
fclose(g);
```

The following codes are attached in the next part. Figure 19 to Figure 22 show the results of sift functions.

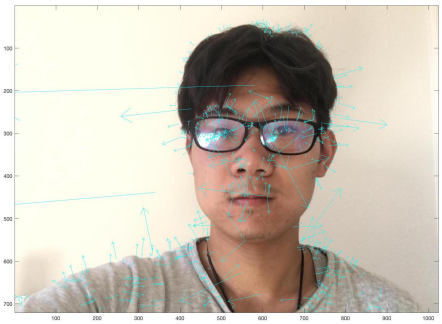

Figure 19: SIFT result of the original image.

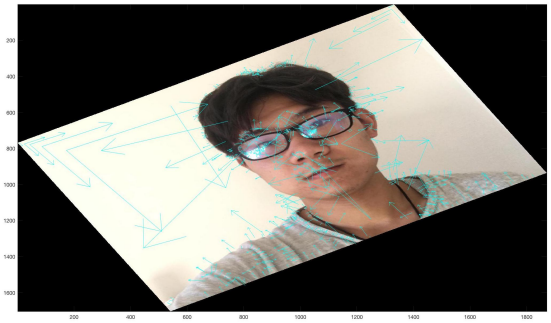

Figure 20: SIFT result of the first processed image.

Step 3: Display two images, find the SIFT feature points and match them.

For this subtask, I first choose two images. Let us name them im1 and im2. The algorithm to find the matching points is easy. I just calculate the distance between each two sift point of the two images. If two points are matched, then the distance between them are much too smaller than others. So I conduct the following method:

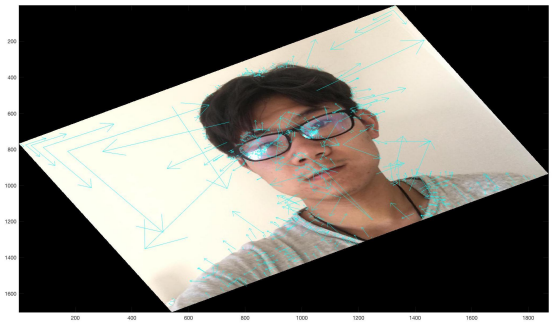

Figure 21: SIFT result of the second processed image.

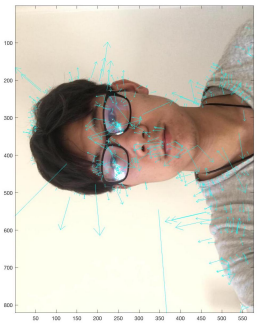

Figure 22: SIFT result of the third processed image.

For each point(vector) in im1, calculate all the distance from it to all the points in im2. Then sort the distance and select the smallest two distances. If the two values are s_1 and $s_2(s_1 < s_2)$. Then if $s_1 < k \times s_2$, I assume they are matched. I set k as 0.6 here.

The following codes shows the algorithm:

```
%Step 3: Match features
% Matlab inbuilt function matchFeatures work well for the this ...
    problem.
% But I still write my own codes to practice the algorithm.
%
%chosen images: im0 and im3

min_locations = [];
[a0,b0] = size(des_ori);
[a1,b1] = size(des_im3);
k = 0.6;
for i = 1:a0
    orin2im3 = repmat(des_ori(i,:),a1,1);
    %map one line in the original image to a map.
    % like (1,2,3)-->(1,2,3;1,2,3;1,2,3)
    distance = sum((orin2im3 - des_im3).^2,2);
    distance = sqrt(distance);
    sort_distance = sort(distance);
    if sort_distance(1)<k*sort_distance(2)
        min_locations = [min_locations;i,find(distance == ...
            sort_distance(1))];
    end

end

matchpoints1 = locs0(min_locations(1:10,1),1:2);
matchpoints2 = locs3(min_locations(1:10,2),1:2);


figure;
showMatchedFeatures(fin, ...
    im3,[matchpoints1(:,2),matchpoints1(:,1)], ...
    [matchpoints2(:,2),matchpoints2(:,1)], 'montage', ...
    'parent',axes);
```

Figure 23 shows the matching points from the original image and the third image. It matches well! To test the robustness of the codes, I also test the matching point from image 1 to image 3. Figure 24 shows the result. These two results show that my algorithm is powerful and robust.

Some other supported codes

Main codes for task 1:

```
close all;
```

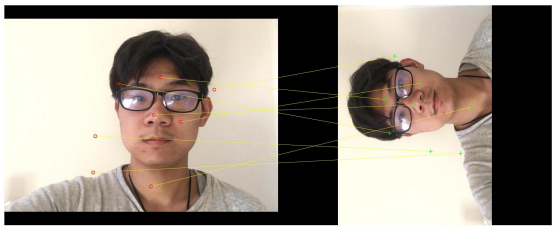

Figure 23: Matching points from original image to image 3.

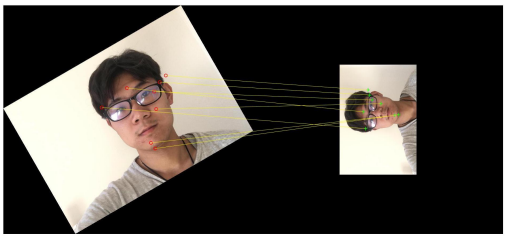

Figure 24: Matching points from image 1 to image 3.

```
clear all;
clc;

im = imread('./images/Lenna.png');
bw = rgb2gray(im);
figure;
subplot(1,2,1);
imshow(bw);

[cols,rws] = my_corner_detector(bw,2,50000000000,11);

hold on;
plot(rws,cols,'or');
title('\bf My Harris Corners with thresh 5e10');

subplot(1,2,2);
imshow(bw);

[cols,rws] = my_corner_detector(bw,2,500000000,11);

hold on;
plot(rws,cols,'or');
title('\bf My Harris Corners with thresh 5e8');


%{
subplot(1,2,2);
imshow(bw);
hold on;
CRNs = corner(bw);
plot(CRNs(:,1),CRNs(:,2),'or');
```

```
33 title('\bf Matlab Harris Corners');
34 %}
```

Main codes for task 2:

```
1  clear all;
2  close all;
3  clc;
4
5  %Step 1: Read input image
6  path1 = './images/Peppers.png';
7  img1 = imread(path1);
8  img1 = uint8(img1/2^8);
9  %imshow(img1);
10
11
12 path2 = './images/mandm.png';
13 img1 = imread(path2);
14 imshow(img1);
15
16
17 title('Input Color Image');
18
19 %Step 2: Convert the image to lab channel
20 cform = makecform('srgb2lab');
21 lab = applycform(img1,cform);
22
23 %Step 3: Form 5-Dimension feature vector.
24 features = im2feature(lab);
25
26 %Step 4: Implement your own k-mean algorithm
27
28 %Step 5: display the segmentation result, by calling the following
29 %supporting function.
30 [data_clusters, cluster_stats] = my_kmeans( features, 10 );
31 displayclusters(lab,data_clusters);
```

Whole codes of k-means function for task 2:

```
1  function [data_clusters, cluster_stats] = my_kmeans( data, nc )
2  % This function performs k-means clustering on data
3  %given (nc) = the number of clusters.
4  %  Random Initialization
5  data = double(data);
6  ndata = size(data,1);
7  ndims = size(data,2);
8
9  random_labels = floor(rand(ndata,1) * nc) + 1;
10
11 data_clusters = random_labels;
```

```
12
13 cluster_stats = zeros(nc,ndims+1);
14
15 distances = zeros(ndata,nc);
16
17 while(1)
18
19     pause(0.03);
20
21     % Make a copy of cluster statistics for
22     % comparison purposes.  If the difference is very small, the ...
           while loop will exit.
23     last_clusters = cluster_stats;
24
25     % For each cluster
26     for c=1:nc
27
28         % Find all data points assigned to this cluster
29         [ind] = find(data_clusters == c);
30         num_assigned = size(ind,1); %number of
31
32         % some heuristic codes for exception handling.
33         if( num_assigned < 1 )
34             disp('No points were assigned to this cluster, some ...
                   special processing is given below');
35
36             % Calculate the maximum distances from each cluster
37             max_distances = max(distances);
38
39             [maxx,cluster_num] = max(max_distances);
40             [maxx,data_point] = max(distances(:,cluster_num));
41
42             data_clusters(data_point) = cluster_num;
43
44             ind = data_point;
45             num_assigned = 1;
46         end   %% end of exception handling.
47
48         % Save number of points per cluster,  plus the mean vectors.
49         cluster_stats(c,1) = num_assigned;
50         if( num_assigned > 1 )
51             summ = sum(data(ind,:));
52             cluster_stats(c,2:ndims+1) = summ / num_assigned;
53         else
54             cluster_stats(c,2:ndims+1) = data(ind,:);
55         end
56
57     end
58
59     % Exit criteria
60     diff = sum(abs(cluster_stats(:) - last_clusters(:)));
61     if( diff < 0.00001 )
62         break;
63     end
64
65     % - Set each cluster center to the average of the points ...
           assigned to it.
```

```
66
67      % - calculate the distance from each point to each cluster ...
            center.
68      cluster_position = cluster_stats(:,2:end);
69      for i = 1:nc
70          a = data-repmat(cluster_position(i,:),ndata,1);
71          distances(:,i) = sqrt(sum(a.^2,2));
72      end
73
74      %%update the membership assignment, i.e., update the ...
            data_clusters with current values.
75      for i = 1:ndata
76          data_clusters(i) = find(distances(i,:) == ...
                min(distances(i,:)),1);
77      end
78      % Display clusters for the purpose of debugging.
79      %pause;
80  end
```

Main codes for task 3:

```
1  clear all;
2  close all;
3  clc;
4  %Step 1: read in and rotate images
5  fin = imread('face_02_u6600985.jpg');
6  %rotate 30,60,90, rescale: 1.5,1.2,0.8
7  im1 = imrotate(imresize(fin,1.5),30);
8  im2 = imrotate(imresize(fin,1.2),60);
9  im3 = imrotate(imresize(fin,0.8),90);
10 %show images
11 figure;
12 subplot(2,2,1);
13 imshow(fin);
14 title('original image');
15 subplot(2,2,2);
16 imshow(im1);
17 title('rotate 30 angles and rescale 1.5');
18 subplot(2,2,3);
19 imshow(im2);
20 title('rotate 60 angles and rescale 1.2');
21 subplot(2,2,4);
22 imshow(im3);
23 title('rotate 90 angles and rescale 0.8');
24 %save images
25 imwrite(im1,'im1.jpg');
26 imwrite(im2,'im2.jpg');
27 imwrite(im3,'im3.jpg');
28
29 %Step 2: show sift images
30 %draw sift images
31 [des_ori, locs0] = my_sift(fin,'origin.key');
32 %showkeys(fin,locs0);
33 [des_im1, locs1] = my_sift(im1,'im1.key');
```

```
%showkeys(im1,locs1);
[des_im2, locs2] = my_sift(im2,'im2.key');
%showkeys(im2,locs2);
[des_im3, locs3] = my_sift(im3,'im3.key');
%showkeys(im3,locs3);

%Step 3: Match features
% Matlab inbuilt function matchFeatures work well for the this ...
    problem.
% But I still write my own codes to practice the algorithm.
%
%chosen images: im0 and im3

min_locations = [];
[a0,b0] = size(des_ori);
[a1,b1] = size(des_im3);
k = 0.6;
for i = 1:a0
    orin2im3 = repmat(des_ori(i,:),a1,1);
    %map one line in the original image to a map.
    % like (1,2,3)-->(1,2,3;1,2,3;1,2,3)
    distance = sum((orin2im3 - des_im3).^2,2);
    distance = sqrt(distance);
    sort_distance = sort(distance);
    if sort_distance(1)<k*sort_distance(2)
        min_locations = [min_locations;i,find(distance == ...
            sort_distance(1))];
    end

end

matchpoints1 = locs0(min_locations(1:10,1),1:2);
matchpoints2 = locs3(min_locations(1:10,2),1:2);


figure;
showMatchedFeatures(fin, ...
    im3,[matchpoints1(:,2),matchpoints1(:,1)], ...
    [matchpoints2(:,2),matchpoints2(:,1)], 'montage', ...
    'parent',axes);
```