ENGN4528/6528 Computer Vision – 2015 Computer-Lab 3 (C-Lab3)

Sai Ma - u5224340

April 2015

1 Task-1: Implement Harris Corner Detector

In this question, our task is fulfill missing parts of the *Harris Corner Detector*. In the *Harris Corner Detector* algorithm, there are two main processes, one is find points with large corner response function R (R ¿ threshold), and next task is take the points of local maxima of R. Go back to the given method, it misses two parts, and they are these two main process. As a result, for the *Compute the cornerness*, it aims to get *cornerness* by image pixel values.

According to the definition, the equation to get R(cornerness) is:

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

where k is empirical constant, and from 10.4 to 0.06. For the M in the 1, the $M(x, y)$ is:

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

In the given codes, it already has code to get $I_x(x, y)$, $I_y(x, y)$:

```
Ix = conv2(im2double(bw), dx, 'same');
Iy = conv2(im2double(bw), dy, 'same');
```

As a result, the $I_x(x, y)^2$, $I_y(x, y)^2$ and $I_x(x, y)I_y(x, y)$ are:

```
g = fspecial('gaussian', max(1, fix(6*sigma)), sigma);
```

```
Ix2 = conv2(Ix.^2, g, 'same');
Iy2 = conv2(Iy.^2, g, 'same');
Ixy = conv2(Ix.*Iy, g, 'same');
```

Then, I perform two iterations of image, and construct matrix M to get R (cornerness):

```
for rowIndex = 1 : rowNum
    for columnIndex = 1 : columnNum

        M = getM(rowIndex, columnIndex, Ix2, Iy2, Ixy, sze);
        cornerness(rowIndex, columnIndex) = det(M) - ...
            0.04*(trace(M))^2;

    end
end
```

In the codes, *getM()* is:

```
function [ M ] = getM(rowIndex, columnIndex, Ix2, Iy2, ...
    Ixy, sze)
% This method used to get the M matrix based on the ...
    windoews size and bw
% image. Once the index over the range of bw, set the ...
    value is 0.

    radius = (sze - 1)/2;
    Ix2Matrix = constructImageMatrix(rowIndex, ...
        columnIndex, Ix2, radius);
    sumIx2 = sum(sum(Ix2Matrix));

    Iy2Matrix = constructImageMatrix(rowIndex, ...
        columnIndex, Iy2, radius);
    sumIy2 = sum(sum(Iy2Matrix));

    IxyMatrix = constructImageMatrix(rowIndex, ...
        columnIndex, Ixy, radius);
    sumIxy = sum(sum(IxyMatrix));

    M = [sumIx2, sumIxy; sumIxy, sumIy2];

end
```

After finish this process, the next job is to get pixel value which its cornerness are greater than *threshold*. At the same time, we also select the pixel which is its local maxima. In order to get the local maxima, we could perform Matlab method *ordfilt2()* to get each pixel maxima. Then, the code on non-maximum suppression and threshold are:

```
for rowIndex = 1 : rowNum
    for columnIndex = 1 : columnNum
        value = cornerness(rowIndex, columnIndex);
        if value > thresh && value == maxValues(rowIndex, ...
            columnIndex)
            cornerness(rowIndex, columnIndex) = 1;
        end
    end
end
```

After this, we located the pixel which is corner by:

```
[rws, cols] = find(cornerness == 1);
```

Then, the pixels in $[rws, cols]$ are our target corners, and we can display these by:

```
imshow(image);
hold on;
plot(p(:,1), p(:,2), 'or');
```

At the same time, we also perform Matlab method *corner()*, and display to compare with their results 1.

In the Matlab method *corner()*, I also set a argument of maximum corner values to make sure it can discover all corners in this figure. Then, my code version discover corner is 476, and Matlab find corner number is 523. According to there are some arguments (e.g. *sigma*, *threshold, etc.*) are different in methods, the detector results are different. It is easy to notice that Matlab version is *sensitive* to cornerness.

Then, we rotate the input image, and perform Harris Corner Detectors again to compare results 2.

In this result, my version detected 521 corners and Matlab found 799. It is easy to notice that after rotating, Matlab discovers more *corners* from figure. The reason is the black areas in rotated picture.

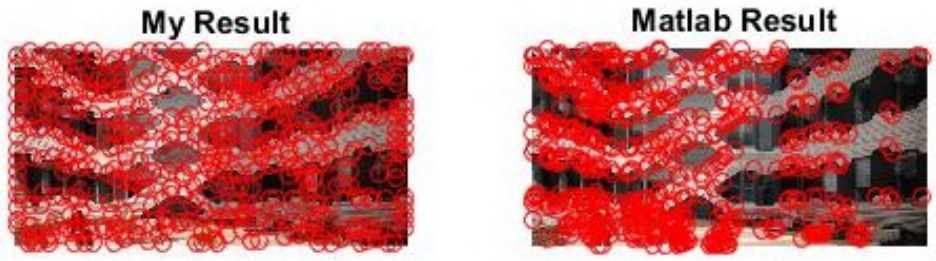

Figure 1: Harries Corner Detector Results

1.1 Compare Original Result and Rotated Result of Matlab Version

After process figure by rotated 45 degrees, Matlab method located more corners (799 ¿ 523) as same as my version (521 ¿ 476). Consider of the original figure, there are some *lines* are perpendicular to figure edges 3. During Harris Corner Detector, it will define this as a line. However, Once original figure rotated, there are some the black areas around original picture 4 and constructs new corner with these lines. This is the reason why there are more corners has been discovered in rotated figure and different from experiment data on Harris Corner Detector.

1.2 Compare My Method Results and Matlab Version Results

Between my version corner results is different from Matlab version results, and it caused by *arguments* in these two methods. In my version, I set threshold is 0.1, kernel size (use to get local maximum) is 9, Gaussian kernel is *fspecial('gaussian',* $max(1, fix(6*2)), 2)$;, empirical constant k is 0.4. For the Matlab version, I use default arguments, which threshold is 0.01, Gaussian kernel is *fspecial('gaussian',*$[51], 1.5)$, empirical constant k is 0.4.

In these arguments, with larger size of Gaussian Kernel, the less corners will discovers. For the empirical constant k, the larger values, the algorithm will discover more corners. For the size of kernel which used to find local

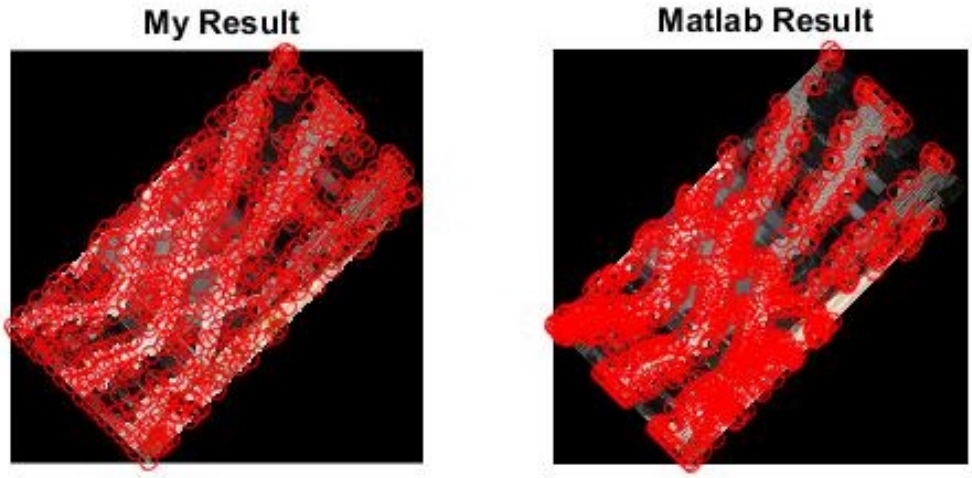

Figure 2: Harries Corner Detector Results after Rotated

Figure 3: Line in Original Figure

maximum value, increase value will find less corners. The above arguments differences cause the different Harris Corner Detector results between my version and Matlab version.

2 Task-2: SIFT Detector and SIFT Descriptors

In this task, we were asked to download and compile existing Matlab/C codes for SIFT descriptor. I download the first link `http://www.robots.`

Figure 4: New Corner in Rotated Figure

`ox.ac.uk/~vedaldi/code/sift.html` and it implemented by Vedaldi. In the code from Vedaldi, it has a main method named *sift* to get the *Scale Invariant Feature Transform* result, which includes *frames* (save keypoints information) and these keypoints SIFT *descriptors*.

2.1 Get Strongest 15 Features in Image

In our task, we only need to extract 15 strongest features via Vedali code, and overlay them in the original picture. In the *sift()* method, there is a argument named *threshold*, and it represents to strength threshold to filter features. Once the maxima of the DOG (Difference of Gaussians) scale space below this strength threshold, it will ignore. In the other words, when we set a larger threshold, only these stronger features will left in return result of method. As a result, my strategy is perform SIFT detector and get returned feature number. While the feature number is bigger than our target feature number (15), the threshold value will add θ, and perform SIFT detector again to get discovered feature number. When the feature number is smaller than our target feature number, minus threshold by half of θ, and set θ equals half of its original value. And get feature number again to compare with target value. The following is the pseudocode of approach:

The following is Matlab code to implement this approach:

Algorithm 1 Get Target Number of Stronger Features

input: $image$, T *(target feature number)*
output: $\{\overrightarrow{feature_1}, ..., \overrightarrow{feature_T}\}$

function IMPROVEDSIFT($image, T$)
  $threshold = 0.01$
  $\theta = 0.01$
  $\{\overrightarrow{feature_1}, ..., \overrightarrow{feature_N}\} \leftarrow$SIFT($image, threshold$)
  while N not equals to T **do**
    if $T < N$ **then**
      $threshold = threshold + \theta$
      $\{\overrightarrow{feature_1}, ..., \overrightarrow{feature_N}\} \leftarrow$SIFT($image, threshold$)
    else $T > N$
      $threshold = threshold - \theta$
      $\theta = \theta/2$
      $\{\overrightarrow{feature_1}, ..., \overrightarrow{feature_N}\} \leftarrow$SIFT($image, threshold$)
    end if
  end while
  return $\{\overrightarrow{feature_1}, ...\overrightarrow{feature_T}\}$
end function

Algorithm 2 SIFT

input: $image, threshold$
output: $\{\overrightarrow{feature_1}, ..., \overrightarrow{feature_N}\}$

function SIFT($image, threshold$)
  codeS from `http://www.robots.ox.ac.uk/~vedaldi/code/sift.html`
  return $\{\overrightarrow{feature_1}, ..., \overrightarrow{feature_N}\}$
end function

```
function [ frames, descr, gss, dogss ] = improvedSIFT( ...
    img, descriptNum )
% This method used to get SIFT with defined descriptor number

    img = img-min(img(:)) ;
    img = img/max(img(:)) ;

    fprintf('Computing frames and descriptors.\n') ;

    Threshold = 0.01; % default threhold of SIFT is 0.01
    theta = 0.01; % set theta is 0.01 to used to modify ...
        threshold value

    while 1 % run while loop until find a correct ...
        threshold to find given feature number

        [frames, descr, gss, dogss] = sift( img, ...
            'Verbosity', 0, 'Threshold', Threshold ) ;
        [¬ , descrNum] = size(descr);

        if descrNum == descriptNum
            break;
        elseif descrNum < descriptNum
            Threshold = Threshold - theta; % go back to ...
                last step threshold number
            theta = theta/2;
        end

        fprintf('Try threshold is %d\n', Threshold);
        fprintf('Get features are %d\n', descrNum);
        Threshold = Threshold + theta;

    end

    fprintf('When set threshold is %d\n', Threshold);
    fprintf('Find features is %d, and equals to our ...
        defined\n', descrNum);

end
```

2.2 Overlay Discovered Features in Image

In order to overlay features in original image, we can use *plotsiftframe* with defined plot style, e.g. circles. The following is its Matlab code (please look at Appendix to read full codes) and display its results 5. For each circle, it has a line in it. This line defines the canonical orientation. At the same time, the radius of the circle define the optimal scale of this keypoint, which calculated by σ of Gaussian kernel. For some circle, there are 2 lines in it, and looks like a diameter, which means there are two features, have same optimal scale, and have opposite canonical orientation.

```
[frames, descr, gss, dogss] = improvedSIFT(img, descriptNum);
figure('name', 'SIFT Results'); clf ; colormap gray ;
imagesc(img), title('Before Rotate'); hold on ; axis equal ;
featureResult1 = plotsiftframe(frames, 'style', 'circle') ;
```

2.3 Display 15 SIFT Descriptors

For each SIFT descriptor, it is a $128D$ SIFT vector, and we can display it as a $1D$ discrete function curve with 128 entries. Then, I use Matlab method *plot* to display these SIFT vectors. The following is code 6

```
% display SIFT descriptors
figure('name', 'SIFT Descriptors before Rotate Display');
subplot(2,2,1);
plot(descr(:,1:4)), title('SIFT descriptors from #1 to #4');
legend('#1', '#2', '#3', '#4');
subplot(2,2,2);
plot(descr(:,5:8)), title('SIFT descriptors from #5 to #8');
legend('#5', '#6', '#7', '#8');
subplot(2,2,3);
plot(descr(:,9:12)), title('SIFT descriptors from #9 to #12');
legend('#9', '#10', '#11', '#12');
subplot(2,2,4);
plot(descr(:,13:15)), title('SIFT descriptors from #13 to ...
    #15');
legend('#13', '#14', '#15');
```

And its result is :

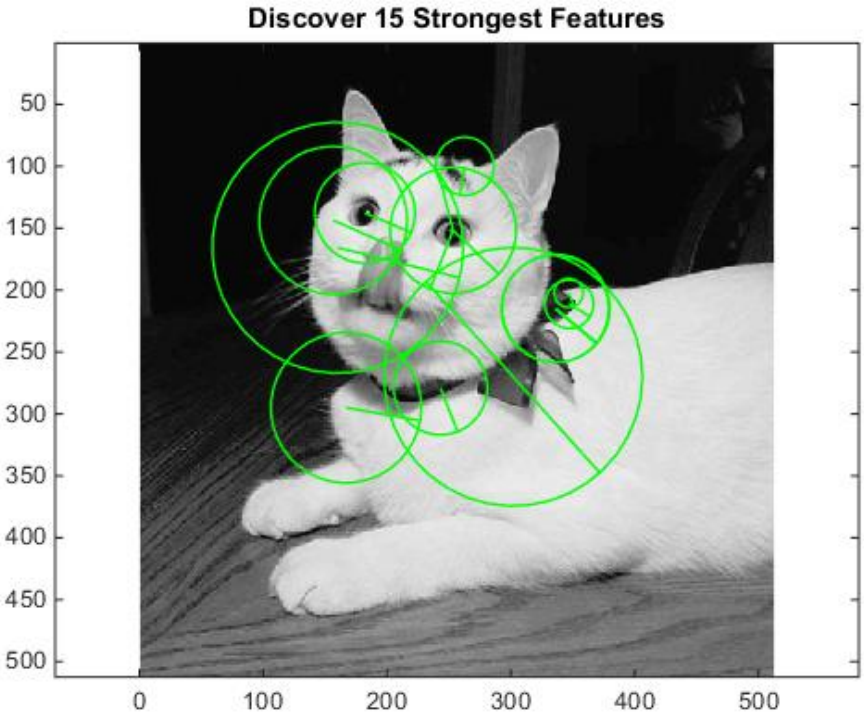

Figure 5: 15 Strongest Features Result

2.4 Compare Two SIFT Results

After get 15 strongest features and SIFT descriptors, the next task is resize image into 768×768 pixels, rotate it by 45 degrees, and then crop out the central part to form a 512×512 subimage. The following is Matlab code:

```
% resize to 768*768, rotate 45
newImg = imresize(img, [768, 768]);
newImg = imrotate(newImg, 45);

% then, crop out central part with 512*512
[newRows, newColumns] = size(newImg);
rect = [fix(newColumns/2) - 255, fix(newRows/2) - 255, ...
    511, 511];
newImg = imcrop(newImg, rect);
```

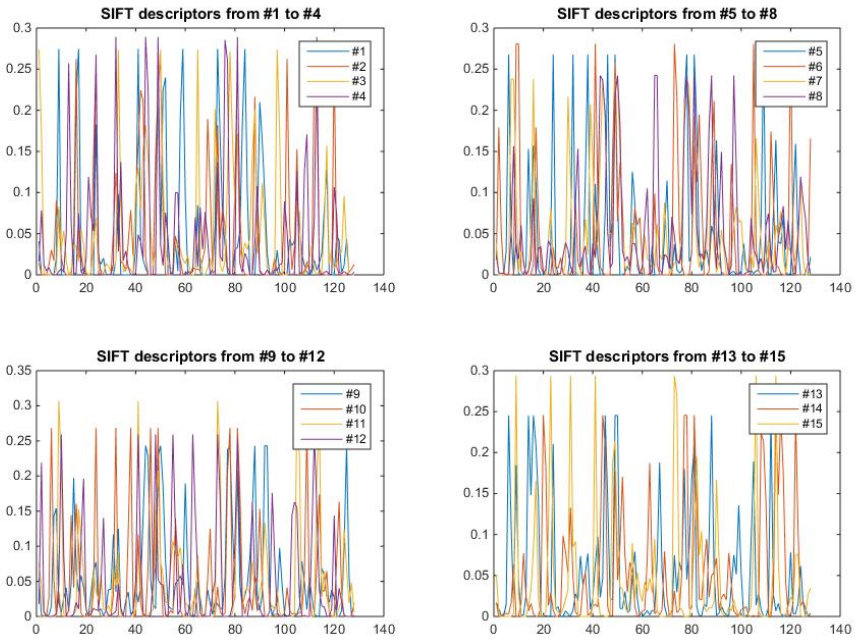

Figure 6: 15 Strongest SIFT Descriptors Result

Then, perform improved SIFT approach to this new image, then display their results 7 8.

As we known, in the features matching approach, its main task is to find two feature with similar SIFT descriptors, and then I perform method from Vedali to test two figures matching results 9.

In this 9, it detected 12 features matching, and the majority is correct. The reason why not match 15 features is that some features in original picture has not crop out to rotated image.

This matching result validates SIFT approach is a scale invariant detection approach, and $128D$ SIFT descriptor vectors also perform well in features matching.

3 Appendix: Matlab Code

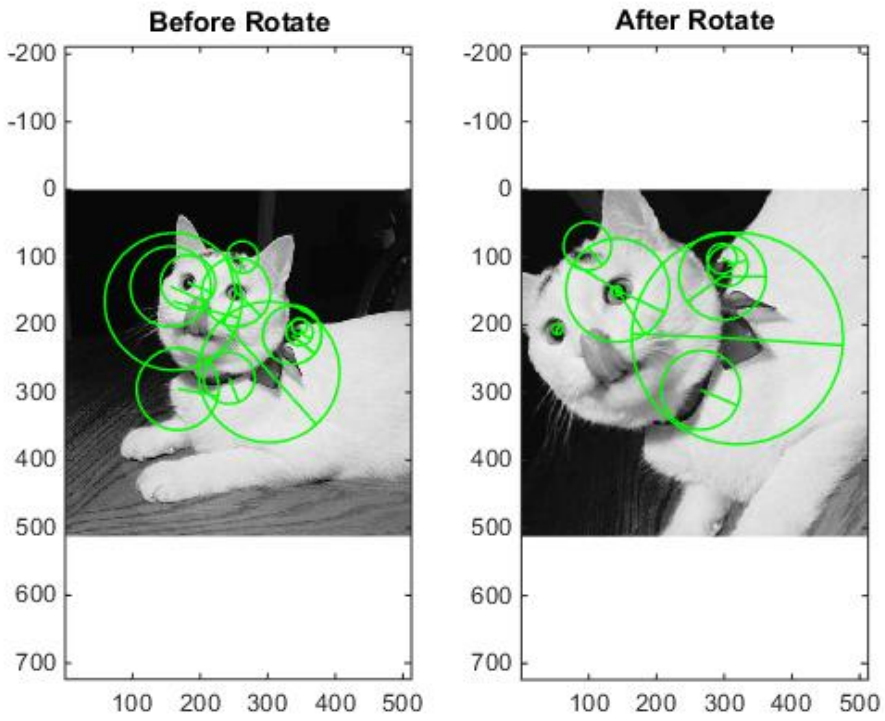

Figure 7: 15 Strongest Features in Two Figures

```
% for task 1: implement your own Harris Corner Detector

% clean un-relatived items
clc;
clear;

% set relative arguments
sigma = 2; thresh = 0.1; sze = 13; disp = 0;

% read image and transform to bw format
image = imread('ANUbuilding2.jpg');
bw = rgb2gray(image);
[rows, columns] = size(bw); % get pixel number to set max ...
    possible corners
raidus = (sze - 1) / 2;
```

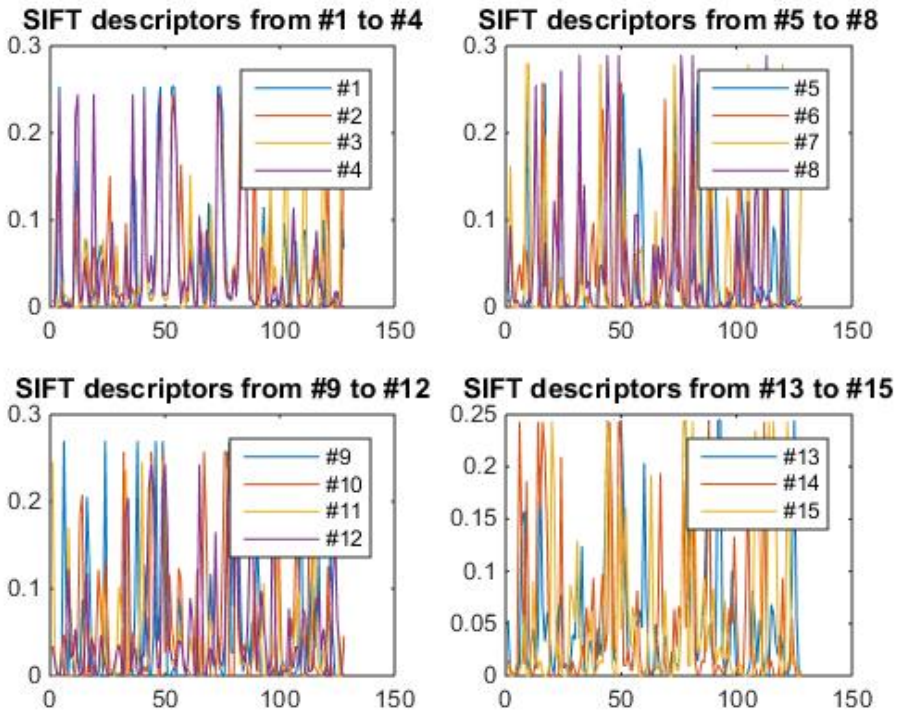

Figure 8: 15 Strongest SIFT Descriptors Result after Rotated

```
fourImageCorners = double([[1 + raidus, 1 + raidus];[1 + ...
    raidus, columns - raidus];[rows - raidus, 1 + ...
    raidus];[rows - raidus, columns - raidus]]); % delete them

% get harrize corner detector results
[rws, cols] = getHarrisCornerDetector(bw, sigma, thresh, ...
    sze, disp);

[newrws, newclos] = getPureCorners(rws, cols, ...
    fourImageCorners);

p = [newclos, newrws];
[cornersNum, ¬] = size(newclos);
fprintf('Before Rotated, My Method Detects Corners Num is ...
    %d\n', cornersNum);

% then, perform Matlab corner method to get result
```

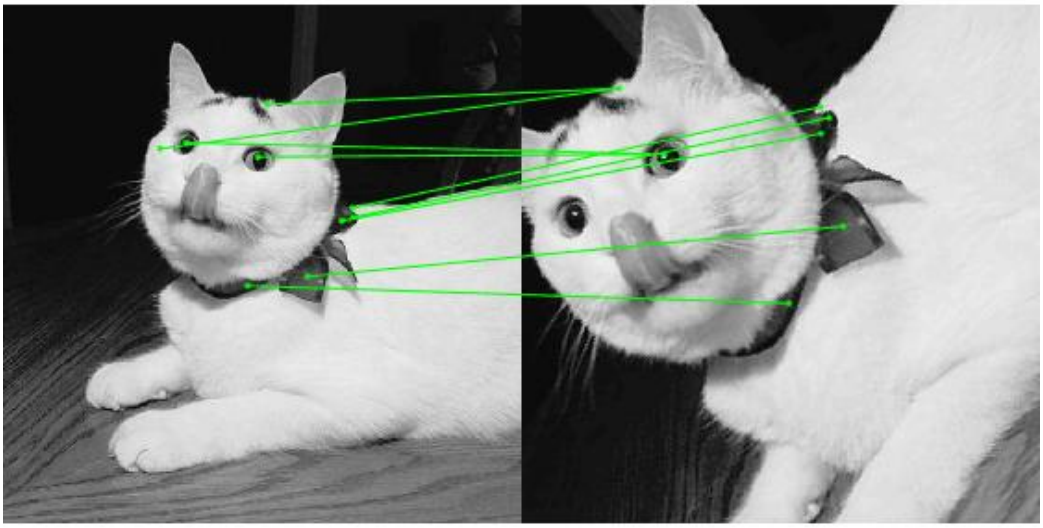

Figure 9: SIFT Matching between Two Images

```
MatlabC = corner(bw, 'Harris', rows*columns);
[cornersNum, ¬] = size(MatlabC(:,1));
fprintf('Before Rotated, Matlab Method Detects Corners Num ...
    is %d\n', cornersNum);

% then, show the display image and add corner data
figure('name', 'Harris Corner Dectector Results: Before ...
    Rotate');
subplot(1,2,1);
imshow(image);
hold on;
plot(p(:,1), p(:,2), 'or');
title('\bf My Result')
hold off

subplot(1,2,2);
imshow(image);
```

```
42 hold on
43 plot(MatlabC(:,1), MatlabC(:,2), 'or')
44 title('\bf Matlab Result')
45 hold off
46
47 % then, rotate the image
48 imgRotate = imrotate(image, 45);
49
50 % then perform Harris Corner Dectector Algorithm again, ...
      display results
51 bwRotate = rgb2gray(imgRotate);
52 [rows, columns] = size(bwRotate);
53
54 % get harrize corner detector results
55 [rws, cols] = getHarrisCornerDetector(bwRotate, sigma, ...
      thresh, sze, disp);
56 pRotate = [cols, rws];
57
58 [cornersNum, ¬] = size(cols);
59 fprintf('After Rotated, My Method Detects Corners Num is ...
      %d\n', cornersNum);
60
61 % then, perform Matlab corner method to get result
62 MatlabCRotate = corner(bwRotate, 'Harris', rows*columns);
63 [cornersNum, ¬] = size(MatlabCRotate(:,1));
64 fprintf('After Rotated, Matlab Method Detects Corners Num ...
      is %d\n', cornersNum);
65
66 % then, show the display image and add corner data
67 figure('name', 'Harris Corner Dectector Results: Before ...
      Rotate');
68 subplot(1,2,1);
69 imshow(imgRotate);
70 hold on;
71 plot(pRotate(:,1), pRotate(:,2), 'or');
72 title('\bf My Result')
73 hold off
74
75 subplot(1,2,2);
76 imshow(imgRotate);
77 hold on
78 plot(MatlabCRotate(:,1), MatlabCRotate(:,2), 'or')
79 title('\bf Matlab Result')
80 hold off
```

```
function [ rws, cols ] = getHarrisCornerDetector( bw, ...
    sigma, thresh, sze, disp )
% This method will use to detect the corn of an image ...
    based on Harris
% Approach.

[rowNum, columnNum] = size(bw);

dy = [-1 0 1; -1 0 1; -1 0 1];
dx = dy';

Ix = conv2(im2double(bw), dx, 'same');
Iy = conv2(im2double(bw), dy, 'same');

g = fspecial('gaussian', max(1, fix(6*sigma)), sigma);

Ix2 = conv2(Ix.^2, g, 'same');
Iy2 = conv2(Iy.^2, g, 'same');
Ixy = conv2(Ix.*Iy, g, 'same');

radius = (sze - 1)/2;
Ix2 = padarray(Ix2, [radius, radius]);
Iy2 = padarray(Iy2, [radius, radius]);
Ixy = padarray(Ixy, [radius, radius]);

cornerness = zeros(size(bw));

for rowIndex = 1 : rowNum
    for columnIndex = 1 : columnNum

        M = getM(rowIndex, columnIndex, Ix2, Iy2, Ixy, sze);
        cornerness(rowIndex, columnIndex) = det(M) - ...
            0.04*(trace(M))^2;

    end
end

% Now we need to perform non-maximum suppression and threshold
% get the local max value within sze
maxValues = ordfilt2(cornerness, sze^2, ones(sze));

for rowIndex = 1 : rowNum
    for columnIndex = 1 : columnNum
        value = cornerness(rowIndex, columnIndex);
```

```
42          if value > thresh && value == maxValues(rowIndex, ...
                columnIndex)
43              cornerness(rowIndex, columnIndex) = 1;
44          end
45      end
46  end
47
48  [rws, cols] = find(cornerness == 1); % find row, col ...
        coords. clf
49
50  if disp == 1
51
52      imshow(bw);
53      hold on;
54      p = [cols, rws];
55      plot(p(:,1), p(:,2), 'or');
56      title('\bf Harris Corners');
57
58  end
59
60  end
```

```
1   function [ M ] = getM(rowIndex, columnIndex, Ix2, Iy2, ...
        Ixy, sze)
2   % This method used to get the M matrix based on the ...
        windoews size and bw
3   % image. Once the index over the range of bw, set the ...
        value is 0.
4
5       radius = (sze - 1)/2;
6       Ix2Matrix = constructImageMatrix(rowIndex, ...
            columnIndex, Ix2, radius);
7       sumIx2 = sum(sum(Ix2Matrix));
8
9       Iy2Matrix = constructImageMatrix(rowIndex, ...
            columnIndex, Iy2, radius);
10      sumIy2 = sum(sum(Iy2Matrix));
11
12      IxyMatrix = constructImageMatrix(rowIndex, ...
            columnIndex, Ixy, radius);
13      sumIxy = sum(sum(IxyMatrix));
14
15      M = [sumIx2, sumIxy; sumIxy, sumIy2];
16
```

```
17 end
```

```
1  function [ imageMatrix ] = constructImageMatrix( ...
       sourceRowIndex, sourceColumnIndex, valueImg, radius )
2  % This method used to return a matrix based on the given ...
     image and kernel
3  % radius. Once the wanted matrix over the range of source ...
     image, set the
4  % value to 0.
5
6      matrixRowIndexUp = sourceRowIndex;
7      matrixRowIndexDown = sourceRowIndex + 2 * radius;
8
9      matrixColumnIndexLeft = sourceColumnIndex;
10     matrixColumnIndexRight = sourceColumnIndex + 2 * radius;
11
12     imageMatrix = ...
           double(valueImg(matrixRowIndexUp:matrixRowIndexDown, ...
           matrixColumnIndexLeft:matrixColumnIndexRight));
13
14 end
```

```
1  function [ purerws, purecols ] = getPureCorners( rws, ...
     cols, fourImageCorners )
2  % This method used to ignore the corners which located in ...
     imageEdge. This
3  % method will perform with assumption there is no corners ...
     in image edges.
4
5      lengths = length(rws);
6      purerws = [];
7      purecols = [];
8
9      for cornerIndex = 1 : lengths
10         element = [cols(cornerIndex,1), rws(cornerIndex,1)];
11
12         if element == [fourImageCorners(1,2), ...
               fourImageCorners(1,1)]
13             continue
14         elseif element == [fourImageCorners(2,2), ...
               fourImageCorners(2,1), ]
15             continue
```

```
        elseif element == [fourImageCorners(3,2), ...
            fourImageCorners(3,1)]
            continue
        elseif element == [fourImageCorners(4,2), ...
            fourImageCorners(4,1)]
            continue
        else
            purerws = [purerws; rws(cornerIndex)];
            purecols = [purecols; cols(cornerIndex)];
        end
    end

end
```

```
% for task 2: perform with SIFT detector and SIFT descriptors

% clean un-relatived items
clc;
clear;

descriptNum = 15; % save 15 strongest description

% read a image, convert to gray, then resize it to 512 * 512
% img = imreadbw('data/img3.jpg');
img = imreadbw('cat.jpg');
img = imresize(img, [512, 512]);

% get SIFT related information
[frames, descr, gss, dogss] = improvedSIFT(img, descriptNum);

% % then, draw features in the photo
% figure('name', 'SIFT Result before Rotated'); clf ; ...
    colormap gray ;
% imagesc(img) ; hold on ; axis equal ;
% featureResult = plotsiftframe(frames, 'style', 'circle') ;

% resize to 768*768, rotate 45
newImg = imresize(img, [768, 768]);
newImg = imrotate(newImg, 45);

% then, crop out central part with 512*512
[newRows, newColumns] = size(newImg);
rect = [fix(newColumns/2) - 255, fix(newRows/2) - 255, ...
    511, 511];
```

```
29 newImg = imcrop(newImg, rect);
30
31 % then, get SIFT related information again
32 [newFrames, newDescr, newGss, newDogss] = ...
       improvedSIFT(newImg, descriptNum);
33
34 figure('name', 'SIFT Results'); clf ; colormap gray ;
35 subplot(1,2,1);
36 imagesc(img), title('Before Rotate'); hold on ; axis equal ;
37 featureResult1 = plotsiftframe(frames, 'style', 'circle') ;
38 subplot(1,2,2);
39 imagesc(newImg), title('After Rotate') ; hold on ; axis ...
       equal ;
40 featureResult2 = plotsiftframe(newFrames, 'style', ...
       'circle') ;
41
42 % display SIFT descriptors
43 figure('name', 'SIFT Descriptors before Rotate Display');
44 subplot(2,2,1);
45 plot(descr(:,1:4)), title('SIFT descriptors from #1 to #4');
46 legend('#1', '#2', '#3', '#4');
47 subplot(2,2,2);
48 plot(descr(:,5:8)), title('SIFT descriptors from #5 to #8');
49 legend('#5', '#6', '#7', '#8');
50 subplot(2,2,3);
51 plot(descr(:,9:12)), title('SIFT descriptors from #9 to #12');
52 legend('#9', '#10', '#11', '#12');
53 subplot(2,2,4);
54 plot(descr(:,13:15)), title('SIFT descriptors from #13 to ...
       #15');
55 legend('#13', '#14', '#15');
56
57 figure('name', 'SIFT Descriptors after Rotate Display');
58 subplot(2,2,1);
59 plot(newDescr(:,1:4)), title('SIFT descriptors from #1 to ...
       #4');
60 legend('#1', '#2', '#3', '#4');
61 subplot(2,2,2);
62 plot(newDescr(:,5:8)), title('SIFT descriptors from #5 to ...
       #8');
63 legend('#5', '#6', '#7', '#8');
64 subplot(2,2,3);
65 plot(newDescr(:,9:12)), title('SIFT descriptors from #9 to ...
       #12');
66 legend('#9', '#10', '#11', '#12');
```

```
67 subplot(2,2,4);
68 plot(newDescr(:,13:15)), title('SIFT descriptors from #13 ...
       to #15');
69 legend('#13', '#14', '#15');
70
71 fprintf('Computing matches.\n') ;
72 % By passing to integers we greatly enhance the matching ...
       speed (we use
73 % the scale factor 512 as Lowe's, but it could be greater ...
       without
74 % overflow)
75 descr1=uint8(512*descr) ;
76 descr2=uint8(512*newDescr) ;
77 tic ;
78 matches=siftmatch( descr1, descr2 ) ;
79 fprintf('Matched in %.3f s\n', toc) ;
80
81 figure('name', 'Matched Results') ; clf ;
82 plotmatches(img, newImg, frames(1:2,:), newFrames(1:2,:), ...
       matches) ;
83 drawnow ;
84
85 figure('name', 'Detected Features Result'); clf ; colormap ...
       gray ;
86 imagesc(img), title('Discover 15 Strongest Features'); ...
       hold on ; axis equal ;
87 featureResult1 = plotsiftframe(frames, 'style', 'circle') ;
```

```
1  % for task 1: implement your own Harris Corner Detector
2
3  % clean un-relatived items
4  clc;
5  clear;
6
7  % set relative arguments
8  sigma = 2; thresh = 0.1; sze = 13; disp = 0;
9
10 % read image and transform to bw format
11 image = imread('ANUbuilding2.jpg');
12 bw = rgb2gray(image);
13 [rows, columns] = size(bw); % get pixel number to set max ...
       possible corners
14 raidus = (sze - 1) / 2;
```

```
15 fourImageCorners = double([[1 + raidus, 1 + raidus];[1 + ...
       raidus, columns - raidus];[rows - raidus, 1 + ...
       raidus];[rows - raidus, columns - raidus]]); % delete them
16
17 % get harrize corner detector results
18 [rws, cols] = getHarrisCornerDetector(bw, sigma, thresh, ...
       sze, disp);
19
20 [newrws, newclos] = getPureCorners(rws, cols, ...
       fourImageCorners);
21
22 p = [newclos, newrws];
23 [cornersNum, ¬] = size(newclos);
24 fprintf('Before Rotated, My Method Detects Corners Num is ...
       %d\n', cornersNum);
25
26 % then, perform Matlab corner method to get result
27 MatlabC = corner(bw, 'Harris', rows*columns);
28 [cornersNum, ¬] = size(MatlabC(:,1));
29 fprintf('Before Rotated, Matlab Method Detects Corners Num ...
       is %d\n', cornersNum);
30
31 % then, show the display image and add corner data
32 figure('name', 'Harris Corner Dectector Results: Before ...
       Rotate');
33 subplot(1,2,1);
34 imshow(image);
35 hold on;
36 plot(p(:,1), p(:,2), 'or');
37 title('\bf My Result')
38 hold off
39
40 subplot(1,2,2);
41 imshow(image);
42 hold on
43 plot(MatlabC(:,1), MatlabC(:,2), 'or')
44 title('\bf Matlab Result')
45 hold off
46
47 % then, rotate the image
48 imgRotate = imrotate(image, 45);
49
50 % then perform Harris Corner Dectector Algorithm again, ...
       display results
51 bwRotate = rgb2gray(imgRotate);
```

```
52 [rows, columns] = size(bwRotate);
53
54 % get harrize corner detector results
55 [rws, cols] = getHarrisCornerDetector(bwRotate, sigma, ...
       thresh, sze, disp);
56 pRotate = [cols, rws];
57
58 [cornersNum, ¬] = size(cols);
59 fprintf('After Rotated, My Method Detects Corners Num is ...
       %d\n', cornersNum);
60
61 % then, perform Matlab corner method to get result
62 MatlabCRotate = corner(bwRotate, 'Harris', rows*columns);
63 [cornersNum, ¬] = size(MatlabCRotate(:,1));
64 fprintf('After Rotated, Matlab Method Detects Corners Num ...
       is %d\n', cornersNum);
65
66 % then, show the display image and add corner data
67 figure('name', 'Harris Corner Dectector Results: Before ...
       Rotate');
68 subplot(1,2,1);
69 imshow(imgRotate);
70 hold on;
71 plot(pRotate(:,1), pRotate(:,2), 'or');
72 title('\bf My Result')
73 hold off
74
75 subplot(1,2,2);
76 imshow(imgRotate);
77 hold on
78 plot(MatlabCRotate(:,1), MatlabCRotate(:,2), 'or')
79 title('\bf Matlab Result')
80 hold off
```