Lab #4 Post-Lab Report

pciftcioglu@sabanciuniv.edu

November 2020

1 Introduction

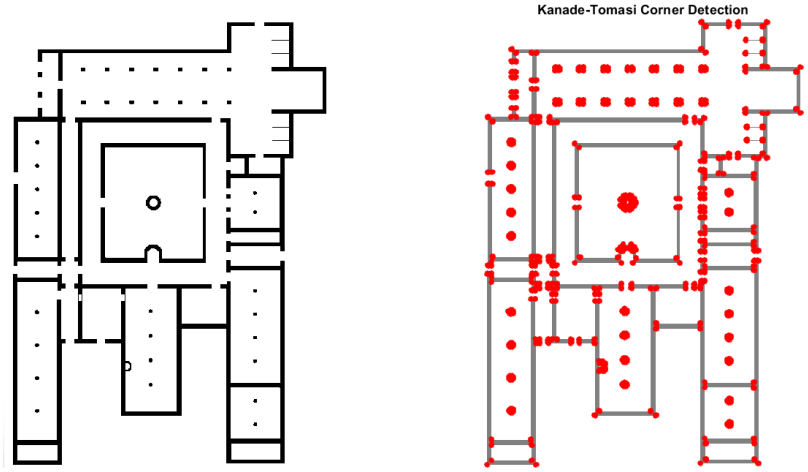
In this lab, we tried to do several operations with related to corner, line and circle detection with using Tomasi Kanade cornerness measure and Hough transformation functions. All of the operations have been performed on gray-scale images and has been checked whether if they are gray-scale before each process and if not, they have been transformed. To avoid any confusion, during the rest of the paper, the images that we will use our operations on will be referred as "Original Image", and images we will be creating after processes will be referred as "Resulting Image". Additionally, we used the **tic** and **toc** commands on the **main.m** to calculate the processing time for each function and printed them to console. Initially, the algorithms for each operation that we have done in the lab will be explained with the methods used and accordingly, further trials with different parameters (e.g. different threshold values) or different images will be discussed with my own results.

2 In Lab Implementations

2.1 Tomasi Kanade Corner Detection

Tomasi Kanade Corner detection algorithm is a method for detecting the corners in the image with getting the use of eigenvalues. For implementing Tomasi Kanade corner detection algorithm, we first initialized the sliding window size as (k = 1) 3x3 and corners array as empty. For corner detection to work better, first we smoothen the image since we wanted to get rid of the possible noises that might have occured in the original image. We either used the built-in function for Gaussian filtering (imgaussfilt(I) where I is the gray-scale version of the original image) or the Gaussian filter we implemented on the previous labs. I preferred the built in function since the Gaussian function I wrote was padding empty pixels to the edges and the corner detection was detecting them as (false) corners. Then we calculated the directional gradients using another built-in function imgradientxy(Simg) (where Simg is the smoothened version of the gray-scale image) and stored them as **Gx** and **Gy**. After that, in the nested for loops, for each window sliding among the Gradient matrices we implemented the following for each iteration:

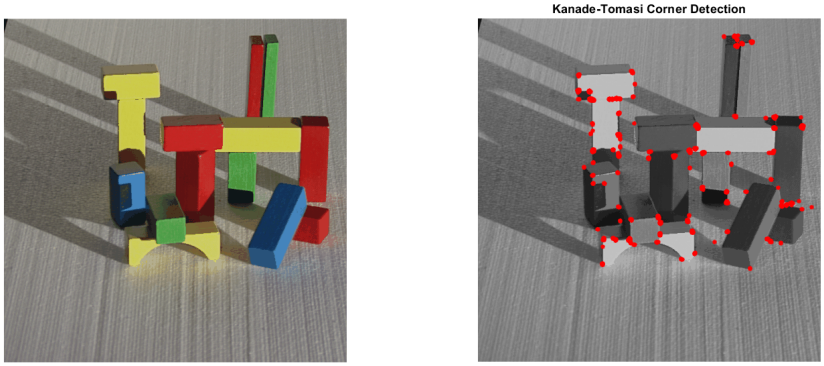
We calculated the square (elements-wise) of the window matrix for both gradients in x and y direction (Gx and Gy) and the (element-wise) multiplication of window matrices of Gx and Gy. Then we summed all the values in each matrix within self where resulting values are the indices of the co-variance matrix, stored as **H** matrix. Then we calculated the eigen-values of the H matrix (λ_1 = lambda1 and λ_2 = lambda2) with eig() command in MATHLAB. Finally, we applied a threshold method again where the minimum of the eigen values of the current pixel that are over a certain threshold are stored in the corners array. At the end, we showed all the corners by plotting them on the image and visualized them.



(a) Original Image (b) Corners, T = 40000

Figure 1: Tomasi Kanade Corner Detection

The threshold value is given by us as a parameter which we used as T = 40000 for this example and as can be seen from the Figure 1 the corner detection algorithm successfully detected all the corners. On the given image, additional to all of the rectangles' corners the dots' edges are detected as well which is also correct.



(a) Original Image (b) Corners, T = 40000

Figure 2: Tomasi Kanade Corner Detection

As it was also given in the lab document and provided us as an image, I also tried this method on another image during the lab which resulted as the Figure 1b with again T = 40000. After looking closely, even though it succeeded to detect most of the corners, there are still few missing that it failed to identify. It might be the case that it is related to threshold value which we will be seeing in the further trials.

2.2 Hough Lines

Here, we implemented a line detection function with using the Hough transform where the points on the image space are mapped into another space called the parameter space. Since the methods we used for edge detection in the previous labs mostly resulted in a disconnection and called as edgels', this time we want to detect the lines where there is not interruption of the lines.

To do so, first I chose an edge detector for extracting the b and white version of the image with the detected edges and since my observations were in the favor Canny Edge detector as successful, I the built-in function edge() with 'Canny' and stored it **img _edges** (see Appendix lab4houghlines.m) and visualized it with the original image. After that, I used the **hough()** built-in function and calculated the H matrix, θ and ρ values with it which is the new values in the parameter space extracted by using the trigonometric parametric equation of the line in 2D. Then I calculated the Hough peaks and Hough lines with again the built-in functions **houghpeaks(H,20,'threshold',Thresh)** and **houghlines(img_edges,theta,rho,peaks)** where H is the Hough transform matrix and 20 is for returning 20 peak values, and Thresh is for the threshold value we determined (also the default value by MATH LAB). H, theta and rho is returned by the hough() function while peaks is returned by the houghpeaks() function. After that I visualized the peaks and showed the lines on the original image. Additionally, I calculated the max and min length with determining the line lengths for each element in the lines (using norm() function) and comparing with the current max and min line value. Finally I visualized the selected lines with the stored end points of them from xy_long and xy_short matrices (see Appendix lab4houghlines.m).

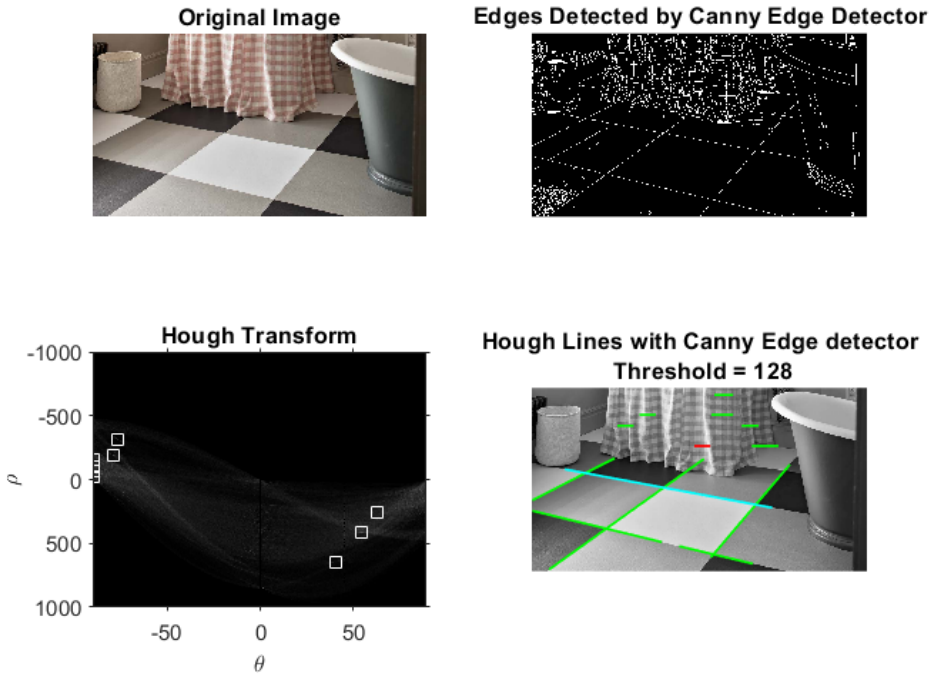


Figure 3: Hough Lines

As it can seen from the Figure 3 that the Hough Transform resulted in sinusoidal waves which represents the every possible line candidates in the image. And as we selected the peak values that are pointed out with boxes, the corresponding lines are the following depicted with green color and also longest and shortest lines are signified with cyan and red respectively.

2.3 Hough Circles

For this last experiment, we used Hough transformation again but this time with circle equation for circle detection. To do so, with the given values for minimum and maximum radius I initialized the radius range as rmin, rmax = [20,60]. Since the functions we used and their parameters are all explained in the lab document I won't be explaining the details but I used the circle detection function imfindcircles() 4 times to try it with different sensitivity parameters and also for detecting the circles darker or brighter than the background. Then I visualized all four different trials (see Appendix lab4houghcircles.m).

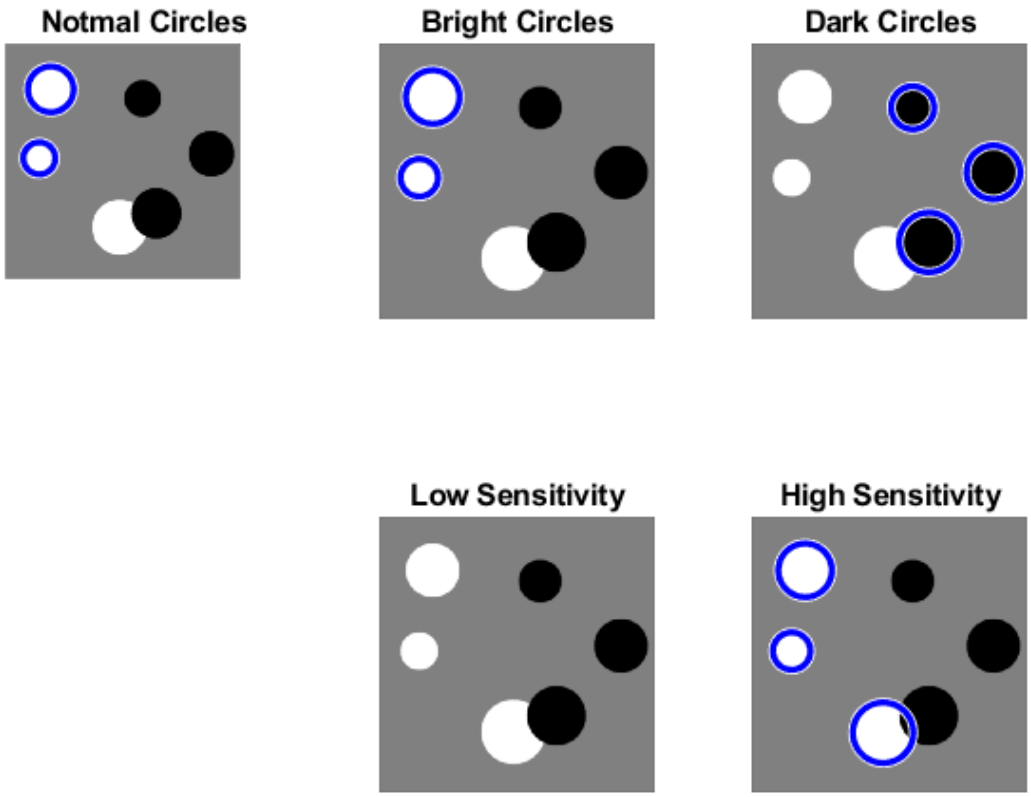


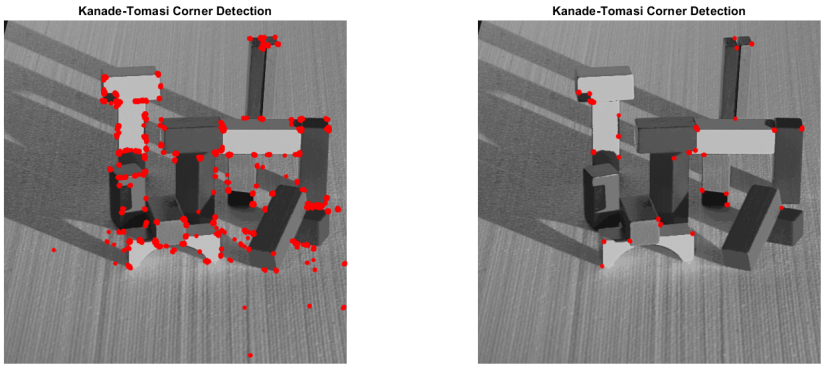
Figure 4: Hough Circles

As can be seen in the Figure 4 "Normal Circles" when I called the function without specifying any parameters with only the radius range it only detected the left 2 circles. However, when I called the function with specifying 'Object Polarity' parameter , it successfully found the circles which are darker and brighter. On the other hand, low sensitivity with 0.4 failed to detect any of the circles but high sensitivity with value 0.9 only detected the brighter circles.

3 Post Lab Experiments

3.1 Tomasi Kanade Corner Detection

Since for the image in Figure 1 we achieved to detect all the corners, I will now try to find a better detection for the second image that we tried which was on the Figure 2 to see whether changing the threshold value will affect the results or not.



(a) Corners, T = 20000 (b) Corners, T = 100000

Figure 5: Tomasi Kanade Corner Detection

As can be seen from the Figure 5a, when I tried a smaller threshold value, the corners appeared to be even less than the actual one so I also tried another high threshold value which can also be seen from the Figure 5b. However, this time, wrong corners started to appear on the places where our eyes wouldn't see as corners which also ended up in a worse result. Thus, it is possible to say that either it should have tried many more values to see the optimum threshold value and might expect more corners to be detected or put the blame on the detection algorithm and accept the fact that it didn't turn out to be that successful for this image.

3.2 Hough Lines

For the Hough lines, as the resulting lines were successful enough I wanted to try another image where the lines are not that clear (e.g. checker on the floor) compared to the image in the Figure 3. And I kept all the other parameters the same such as the threshold.

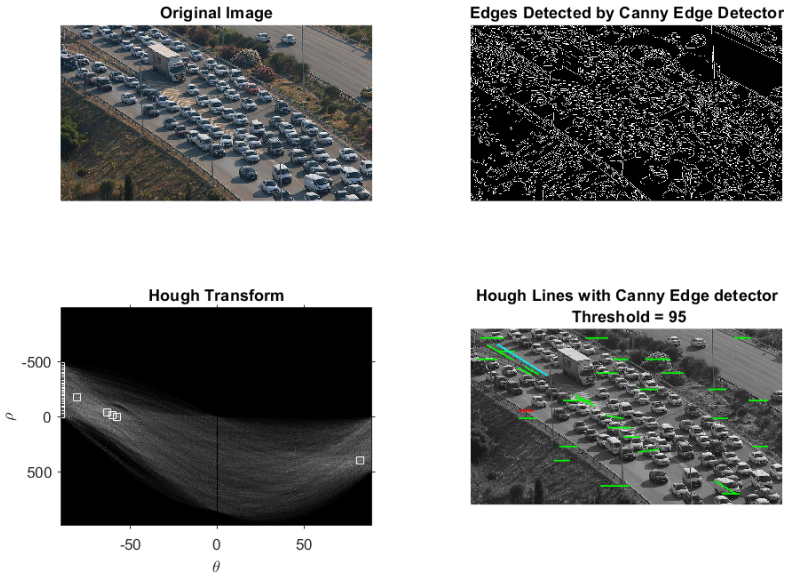


Figure 6: Hough Lines

Since one of the places that the line detection might be seen and be useful in real life is detecting the lines on the road for the automated cars to follow, I wanted to see its performance on a road. However, as can be seen from the Figure 6 that it failed to detect any of the lines of the road but rather detected the barriers on where the road broke apart into two. And also some of the lines were arbitrary when I observed it with my eyes. While the longest line indicated with blue was actually line but the actual longest, red line was barely significant on the original image to be a line. Thus, I tried another image where there is clearly lines in the seen that has been captured.

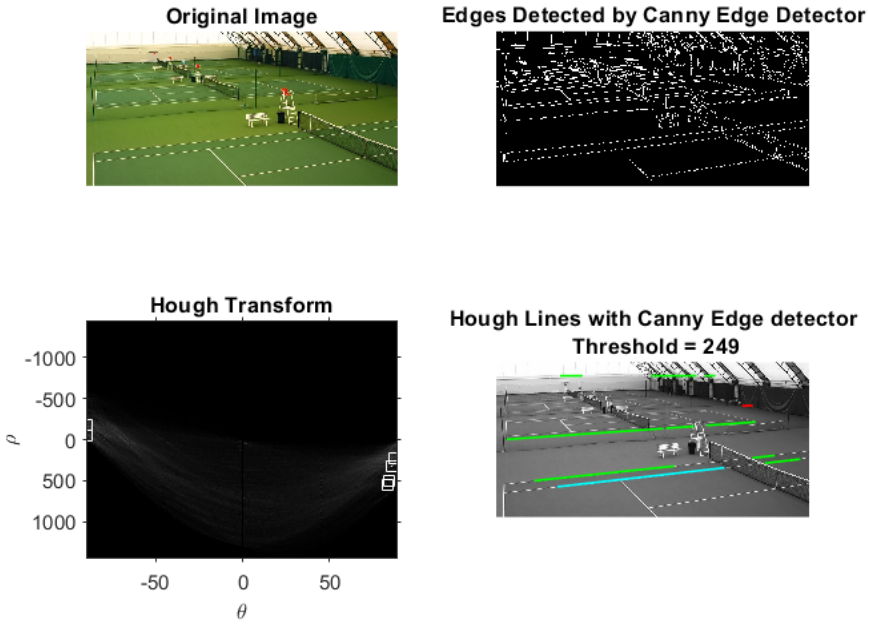


Figure 7: Hough Lines

As it can be seen from the Figure 7 the transform was able to detect few lines as the lines of the borders of the tennis court but it failed to detect all of them. It might be related to the threshold value that has been used or other parameters initialized so to achieve better results a deeper analysis should be done with trying to find the optimal values for each unique image.

Finally, when compared the results of the graphs for Hough transforms from Figure 3, 6, 7, it is possible to conclude that when the lines are more visible and easier to detect by the algorithm the resulting lines are less visible since the candidates are lower. When looked at the resulting graph on 6, the lines and the waves are quite complicated.

3.3 Hough Circles

Since the algorithm did not detect all the circles when I didn't insert any parameters and failed to detect any circles when the sensitivity is low (0.4) I wanted to do more trial with experimenting on the sensitivity parameter to see how it affects the results.

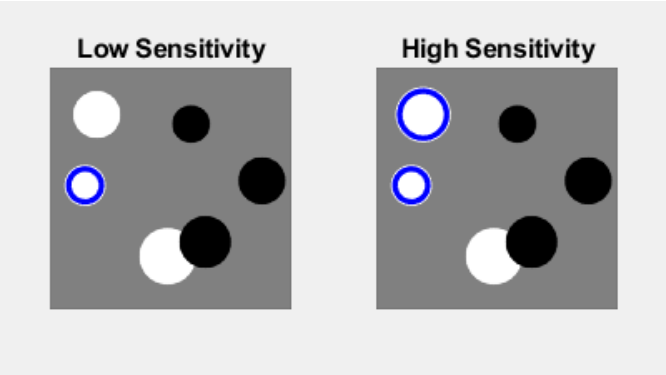


Figure 8: Hough Circles with Sensitivity 0.6 and 0.8

Since the default value for sensitivity is 0.85 (according to MATH LAB documentation) and max is 0.9 and we already tried both of them, I tried two different values in between the values I already tried (0.4 and 0.9). And as can be seen from the Figure 8, the resulting detecting were not any better than the previous trials and they both only found the brighter circles. Thus, based on my observations, we can conclude than the function performs better with higher sensitivity (0.9 optimal) on this image and detect bright circles. However, it is important to note that, it has been achieved to detect all the circles with specifically using the parameters 'bright' or 'dark' and can be combined to achieve a total result of detected circles as can be seen from Figure 9, so it might be a better idea to use two functions instead of one and merge them.

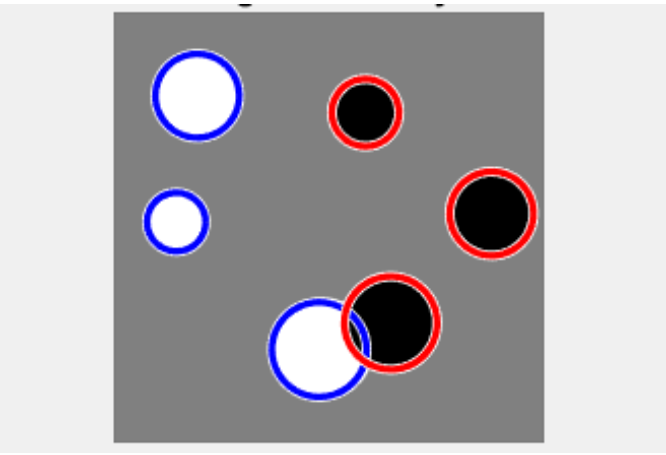


Figure 9: Hough Circles

Appendix

3.4 main.m

```
%% Corner detection
clear all; close all; clc;
tic

img = imread('blocks.png');
imshow(img);
Corners_Filtered = lab4ktcorners(img);
toc

%% Hough Lines
clear all; close all; clc;
tic
img = imread('checker.jpg');
[H,theta,rho] = lab4houghlines(img);
toc
%% Hough Circles
clear all; close all; clc;
tic
img = imread('circlesBrightDark.png');
[centersBright, radiiBright, centersDark, radiiDark] = lab4houghcircles(img);
toc
```

3.5 lab4ktcorners.m

```
function [corners] = lab4ktcorners(I)

[r,c,ch] = size(I);
if (ch ==3)
    I = rgb2gray(I);
end

img = I;

% Corners_Filtered = zeros(size(img));

% Transforming to double (preparation for calculations)
img = double(img);
%Corners_Filtered = double(Corners_Filtered);

% Initializations of corners, threshold, window size
corners = [];
T = 40000;
k = 1;

% Smoothing
Simg = imgaussfilt(img);

%  Finding H for each Pixel using windows
[Gx,Gy] = imgradientxy(Simg);

for i = k+1:1:r-k-1
    for j=k+1:1:c-k-1
        Gx_window = Gx(i-k:i+k,j-k:j+k);
        Gy_window = Gy(i-k:i+k,j-k:j+k);

        I_x2 = Gx_window .* Gx_window;
        I_xy = Gx_window .* Gy_window;
        I_y2 = Gy_window .* Gy_window;

        I_x2= sum(I_x2(:));
        I_xy= sum(I_xy(:));
        I_y2= sum(I_y2(:));
```

```
39
40          H = [I_x2, I_xy;
41               I_xy, I_y2];
42
43          e = eig(H);
44
45          lambda1 = e(1);
46          lambda2 = e(2);
47
48          % Finding eigen values and thresholding to get corners
49
50          if (min(lambda1,lambda2) > T)
51              corners = [corners; i,j];
52          end
53
54      end
55  end
56
57  % Visualization
58  img = uint8(img);
59
60  figure
61  imshow(I);
62  hold on;
63  plot(corners(:,2),corners(:,1),'r*', 'MarkerSize', 4, 'Linewidth', 1);
64  title('Kanade-Tomasi Corner Detection');
65  end
```

3.6 lab4houghlines.m

```
1   function [H,theta,rho] = lab4houghlines(I);
2
3   [r,c,ch] = size(I);
4   if (ch ==3)
5       img = rgb2gray(I);
6   end
7
8      % Edge Detection: the selection of the edge detector is very significant!
9      % Hint use edge built-in function
10     img_edges = edge(img,'Canny');
11
12  % Hough Transform + Display
13      [H,theta,rho] = hough(img_edges, 'RhoResolution',0.5,'Theta',-90:0.5:89);
14
15      figure('Name','Hough Transform','NumberTitle','off');
16          subplot(2,2,1)
17          imshow(I);
18          title('Original Image');
19
20          subplot(2,2,2)
21          imshow(img_edges);
22          title(sprintf('Edges Detected by %s Edge Detector', 'Canny'));
23
24          subplot(2,2,3)
25
26          imshow(H,[],'XData',theta,'YData',rho,'InitialMagnification','fit');
27          title('Hough Transform');
28          xlabel('\theta'), ylabel('\rho');
29          axis on, axis normal, hold on;
30
31  Thresh = ceil(0.5*max(H(:)));
32
33  %Hough peak points and hough lines, find 20 peaks and threshold value is Thresh
34  peaks = houghpeaks(H,20,'threshold',Thresh);
35  x = theta(peaks(:,2)); y = rho(peaks(:,1));
36  plot(x,y,'s','color','white');
37
```

```
% Hough Lines
lines = houghlines(img_edges,theta,rho,peaks,'FillGap',10,'MinLength',40);

% Plotting All Lines and Highlighting Longest and Shortest Lines

subplot(2,2,4);
    imshow(img),title(sprintf('Hough Lines with %s Edge detector\n Threshold = %d', 'Canny',
    Thresh)), hold on

    max_length = 40;
    min_length = 2000;

%% Plotting the lines

for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','green');

   len = norm(lines(k).point1 - lines(k).point2);
   % for determining the longest line segment
   if ( len > max_length)
      max_length = len;
      xy_long = xy;
   elseif (len < min_length)
      min_length = len;
      xy_short = xy;
   end
end

plot(xy_long(:,1),xy_long(:,2),'LineWidth',1,'Color','cyan');
plot(xy_short(:,1),xy_short(:,2),'LineWidth',1,'Color','red');
```

3.7 lab4houghcircles.m

```
function [centersBr, radiiBright, centersD, radiiDark] = lab4houghcircles(I)

[r,c,ch] = size(I);
if (ch ==3)
    img = rgb2gray(I);
end

% Radius Range
rmin = 20;
rmax = 60;

% Circle detection with different parameters

[centerN, radiiNormal] = imfindcircles(I,[rmin rmax]);
[centersBr, radiiBright] = imfindcircles(I,[rmin rmax],'ObjectPolarity','bright');
[centersD, radiiDark] = imfindcircles(I,[rmin rmax],'ObjectPolarity','dark');
[centerL, radiiLow] = imfindcircles(I,[rmin rmax],'Sensitivity',0.6);
[centerH, radiiHigh] = imfindcircles(I,[rmin rmax],'Sensitivity',0.9);

%Visualisation

figure
    subplot(2,3,1)
    hold
    imshow(I)
    viscircles(centerN, radiiNormal,'Color','b');
    title("Notmal Circles")

    subplot(2,3,2)
    imshow(I)
    hold
    viscircles(centersBr, radiiBright,'Color','b');
    title("Bright Circles");
```

```
34
35      subplot(2,3,3)
36      imshow(I)
37      hold
38      viscircles(centersD, radiiDark,'Color','b');
39      title("Dark Circles");
40
41      subplot(2,3,5)
42      imshow(I)
43      hold
44      viscircles(centerL, radiiLow,'Color','b');
45      title("Low Sensitivity")
46
47      subplot(2,3,6)
48      imshow(I)
49      hold
50      viscircles(centerH, radiiHigh,'Color','b');
51          viscircles(centersD, radiiDark,'Color','r');
52
53      title("High Sensitivity")
```