EE 417 LAB#4 POST-LAB REPORT

GOKTUG KORKULU

27026

HOUGH TRANSFORMATION

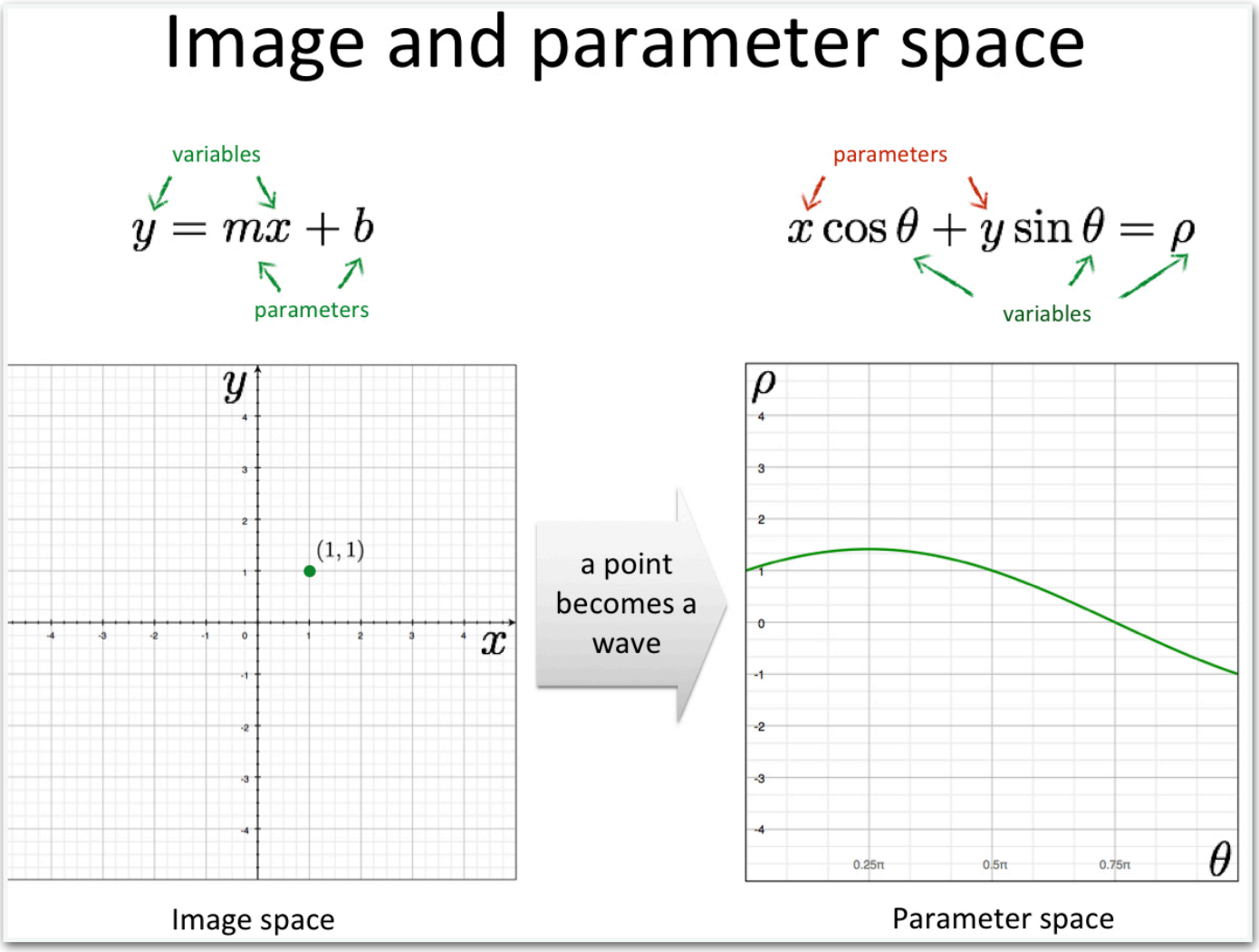
Hough transform is a method that is used to find aligned points in the images that creates lines. The original hough transform is related to transform image space lines which are having $y = a\,x + b$ representation to parameter space which are having $b = y - a\,x$ representation. In this notion, pixel points on image space are mapped into lines in parameter space; and lines in image space are mapped into points in parameter space. However, the problem of a and b values being unbounded in parameter space makes our job way harder.

For this reason, Duda and Hart propose another parameter space. Rather than representing parameter space as $b = y - a\,x$, they claim to represent using straight-line parametrization by rho and theta;

$$\rho = x * cos(\theta) + y * sin(\theta).$$

By this representation, beside the fact that we follow original Hough Transformation ideas, we now have bounded parameter space such as $0 \leq \theta < 2\pi$ and $d_m \leq \theta < 0$ where $d_m = \sqrt{(row)^2 + (col)^2}$.

This approach is known as ***Standart Hough Transformation***.



Standart Hough Transformation visualisation. (Taken from lecture slides.)

```
%Polar Hough Transform for Lines

function HTPLine(inputimage)

%image size
[rows,columns]=size(inputimage);

%accumulator
rmax=round(sqrt(rows^2+columns^2));
acc=zeros(rmax,180);

%image
for x=1:columns
  for y=1:rows
     if(inputimage(y,x)==0)
        for m=1:180
           r=round(x*cos((m*pi)/180)
             +y*sin((m*pi)/180));
           if(r<rmax & r>0)
              acc(r,m)= acc(r,m)+1; end
        end
     end
  end
end
```

The algorithm for standart Hough Transformation (taken from lecture slides)

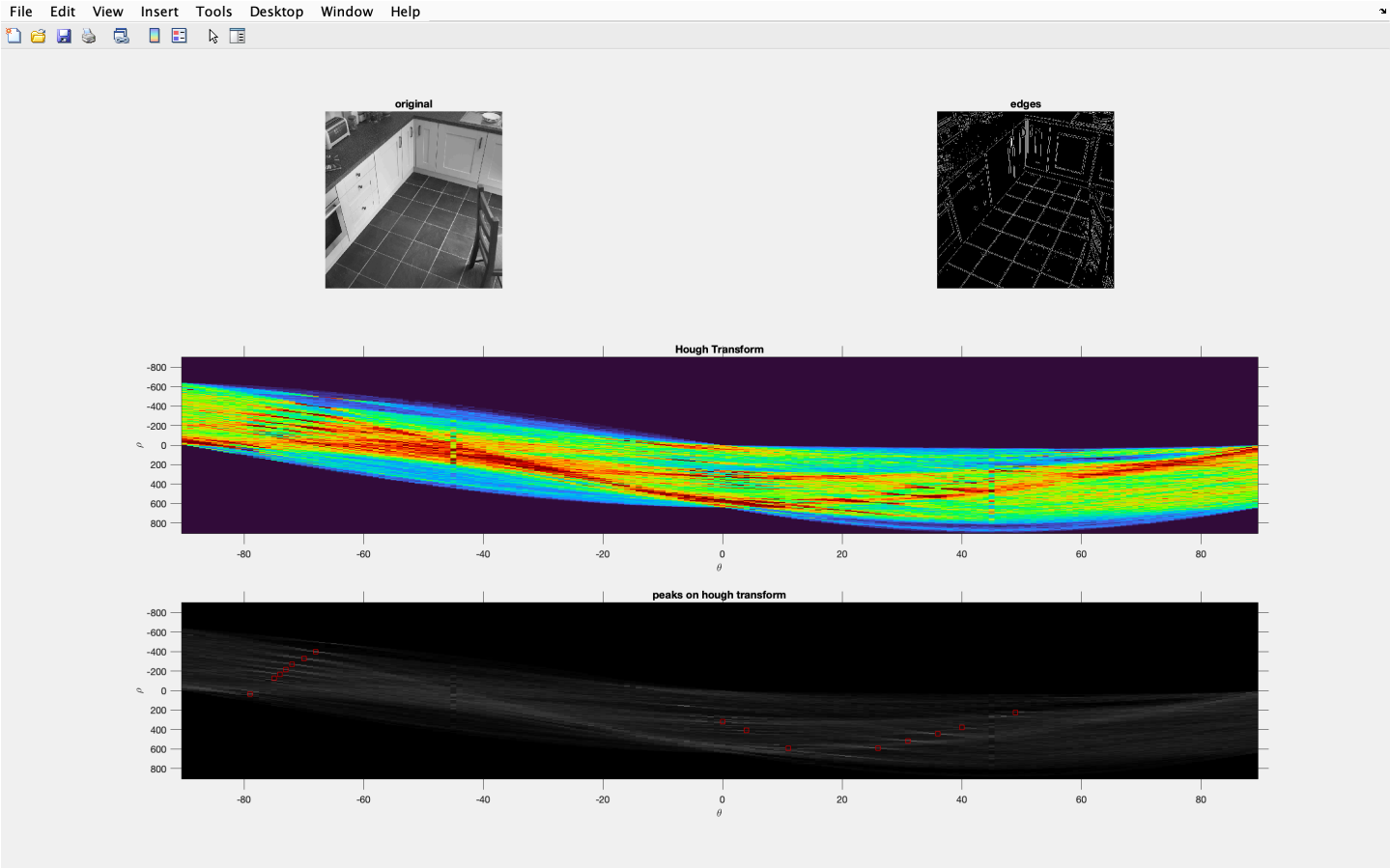
In this week's lab, I will write two programs as ***lab4houghlines.m*** and ***lab4houghcircles.m*** to detect lines and circles by using Hough Transform built-in MATLAB functions with different parameters. I will also use ***tic-toc*** commands in order to evaluate the execution time performances of each method.

LINE DETECTION

```
1  function [H,theta,rho] = lab4houghlines(img)
2
3  %convert to gray-scale
4  [~,~,ch] = size(img);
5  if (ch==3)
6      img = rgb2gray(img);
7  end
8
9  %detect edges
10 edge_detector = 'LoG';
11 img_edges = edge(img,edge_detector);
12 figure;
13 subplot(3,2,1), imshow(img), title('original');
14 subplot(3,2,2), imshow(img_edges), title('edges');
15
16 %hough transform of the edge image
17 [H,theta,rho] = hough(img_edges);
18 subplot(3,1,2), imshow(imadjust(rescale(H)),'XData',theta, 'YData', rho, 'InitialMagnification','fit'), title('Hough Transform');
19 xlabel('\theta'), ylabel('\rho');
20 axis on, axis normal, hold on;
21 colormap(gca,"turbo");
22
23 %peak Hough points. I tried and saw that there are 15 pixels that are above
24 %threshold so that's why I set the number of peaks to be found as 15.
25 peaks = houghpeaks(H,15, 'Threshold',0.5*max(H(:)));
26 subplot(3,1,3), imshow(H, [], 'XData', theta, 'YData', rho, 'InitialMagnification', 'fit'), title('peaks on hough transform');
27 xlabel('\theta'), ylabel('\rho');
28 axis on, axis normal, hold on;
29 plot(theta(peaks(:,2)), rho(peaks(:,1)), 's', 'color', 'red');
30
```

- This functions takes an arbitrary image as input and returns Hough matrix (***H***), ***theta*** and ***rho*** values. Before the integral parts of this function, we check whether the input image is gray-scaled or not. If not, we convert it to gray-scaled by checking the channel number of that image and proceeding accordingly. (Shown 1,2).
- First of all, before applying the Hough transforms, we should convert the image to edge image. In the line 10, we choose the edge detection method, in this specific example I chose Laplacian of Gaussian (LoG). Afterwards, we proceed and obtain edge image by using ***edge()*** built-in function. ***edge(I, method)*** detects edges in image *I* using edge-detection algorithm specified by *method*. Right after we get edge image, we plot original and edge images side by side. (Shown in 3).
- Now, it is time to do the most important operations of this lab. We are applying ***hough()*** built-in function to edge image which computes the Standart Hough Transform of the binary edge image. The hough function is designed to detect lines. What this function returns is;
  - H: a parameter space matrix whose rows and columns correspond to rho and theta values respectively.
  - Rho: the distance from the origin to the line along a vector perpendicular to the line.
  - Theta: the angle in degrees between the x-axis and this vector.
- Right after we gather H, rho and theta values, we plot the hough transform result. rescale(A) scales the entries of an array to the interval [0,1]. imadjust() maps the intensity values in gray-scaled image to new values by saturating the bottom and the top 1% of all pixel values.we label x-axis as theta, y-axis as rho and color the graph with colormap name turbo. (Shown in 4).
- In part 5, we get peak values of Hough transform which pass the threshold value that is half of the maximum value of hough matrix ***H***. We do this process by utilizing ***houghpeaks()*** function. ***houghpeaks(H, numpeaks, Name, Value)*** locates peaks in the Hough transform matrix, H, generated by hough function. n*umpeaks* specifies the maximum number of peaks to identify. *Name:'threshold'* implies that minimum value to be considered a peak. In this function, 'peaks = houghpeaks(H,15, 'Threshold',0.5*max(H(:)));' means determine the maximum 15 amount of peaks which passes the threshold value of half of the maximum peak value in Hough matrix.

- After that, we plot the Hough Transform graph again but this time without coloring. Then we label x and y-axis as same as before and plot peak values on top of the hough transform graph so that we see the peak value which passes the value of half of the maximum value of Hough matrix. (Shown in 5).



- Here we see the gray-scaled version of input image, the edge detected version of gray-scaled image, the hough transform graph colored as ‘turbo’ and the 15 peak values who passes the given threshold value.

```
30
31  %hough lines.
32  lines = houghlines(img_edges, theta, rho, peaks, 'FillGap',10,'MinLength',40);
33
34  pause(1)
35  close all
36  figure;
37  imshow(img), hold on;
38  max_len = 0;
39  min_len = 300;
40
41
42  for k = 1:length(lines)
43      xy = [lines(k).point1; lines(k).point2];
44      plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
45
46      %plot beginnings and ends of lines
47      plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
48      plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');
49
50      %determine the endpoints of the longest and shortest line segments
51      len = norm(lines(k).point1 - lines(k).point2);
52
53      if ( len > max_len)
54        max_len = len;
55        xy_long = xy;
56      end
57
58      if(len < min_len)
59          min_len = len;
60          xy_short = xy;
61      end
62
63  end
64
65      plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan');
66      plot(xy_short(:,1),xy_short(:,2),'LineWidth',2,'Color','red');
67      %display('minimum length : ' + min_len + ', max length: '+ max_len);
68      xlabel({['minimum length: ',num2str(min_len)],['maximum length: ',num2str(max_len)]});
69      %xlabel({'maximum length',max_len});
70
71  end
```

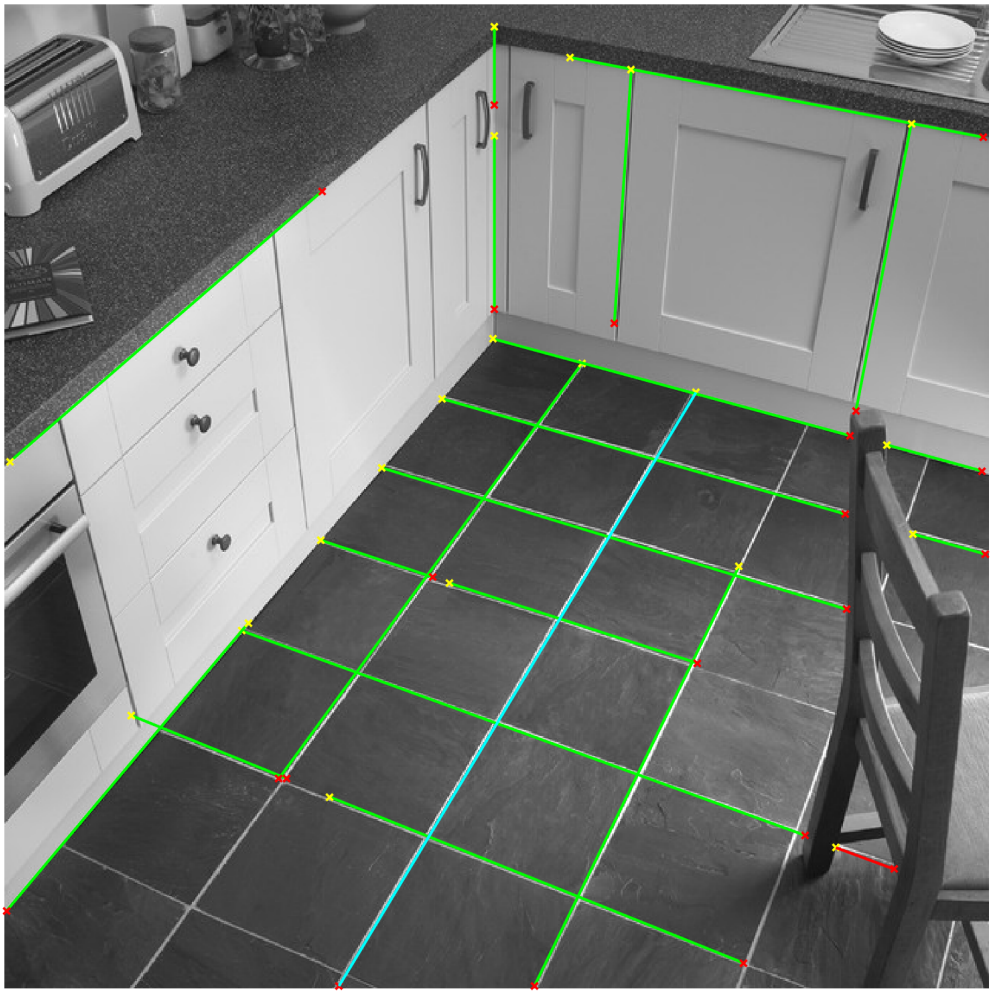
- In line 32, we use ***houghlines()*** built-in function in order to get the Hough Lines and assign it to ***lines*** variable. ***houghlines(I, theta, rho, peaks)*** extracts line segments in the image *I* associated with particular bins in a Hough Transform. *Theta* and *rho* are vectors returned by function hough. *Peaks* is a matrix returned by the houghpeaks() function that contains the row and column coordinates of Hough transform bins to use in searching for line segments. The return value *lines* contains information about the extracted line segments. By "'FillGap',10" input; when the distance between the line segments is less than the value specified, the outlines function merges the line segment into a single line segment. In our case, when the distance between 2 line segments are at most 10 pixels away, we merge into 1 whole line segment. Also. Thanks to "'MinLength',40" input; houglines function discards lines that are shorter than the value specified. In our example, houghlines function will discard the lines less than 40 pixels. (Shown in 6).

lines

1x21 struct with 4 fields

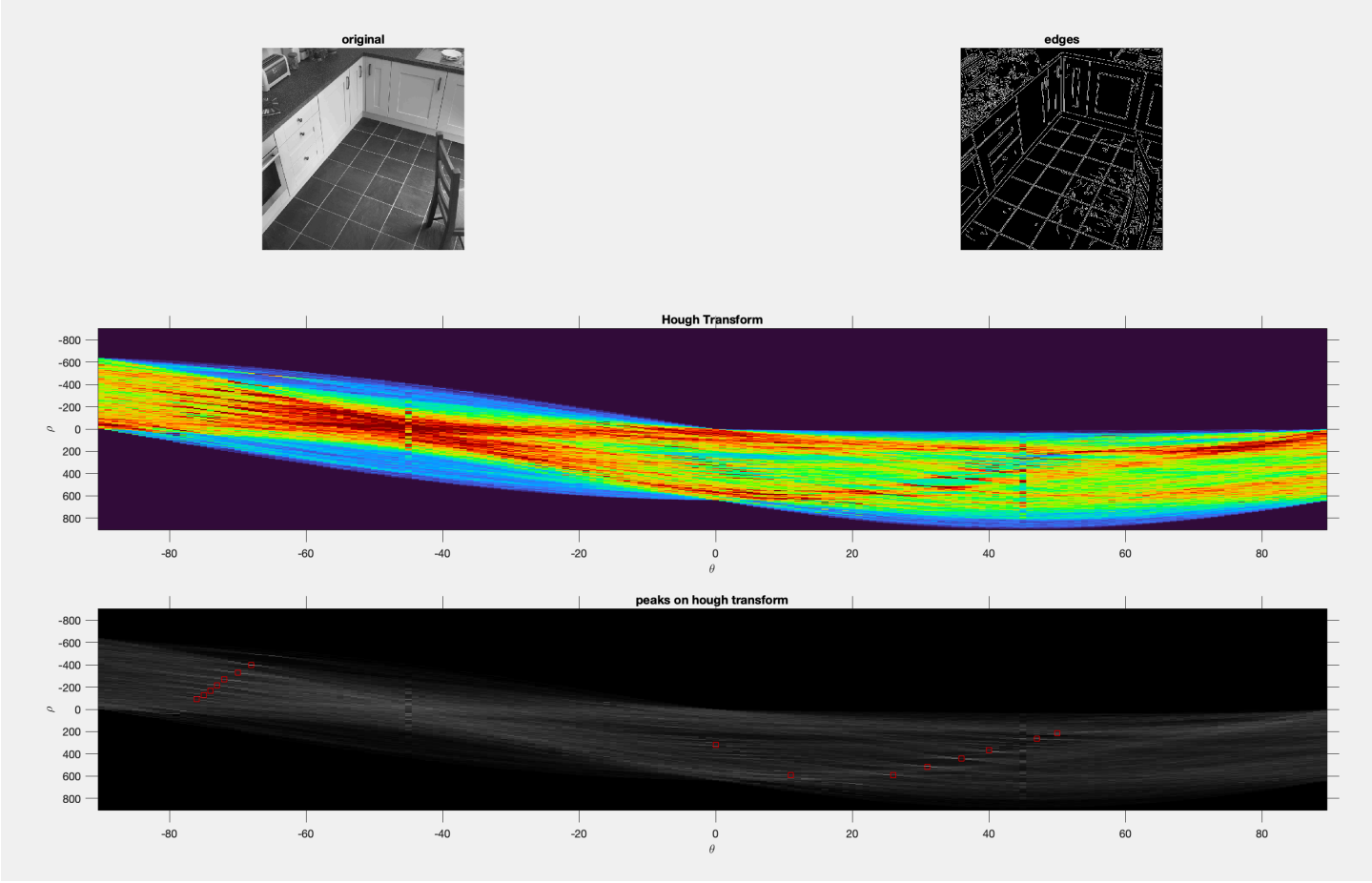
Fields	point1	point2	theta	rho
1	[155,408]	[521,541]	-70	-330
2	[541,549]	[579,563]	-70	-330
3	[450,253]	[218,639]	31	515
4	[478,366]	[345,639]	26	589
5	[83,463]	[184,504]	-68	-398
6	[212,516]	[481,624]	-68	-398
7	[376,234]	[179,504]	36	440
8	[368,35]	[637,87]	-79	37
9	[246,302]	[548,394]	-73	-216
10	[285,257]	[547,332]	-74	-168
11	[591,345]	[638,358]	-74	-168
12	[206,349]	[279,373]	-72	-268
13	[290,377]	[451,429]	-72	-268
14	[319,15]	[319,66]	0	318
15	[319,86]	[319,199]	0	318
16	[408,43]	[397,208]	4	409
17	[590,78]	[554,265]	11	593
18	[159,403]	[2,590]	40	379
19	[4,298]	[207,122]	49	226
20	[318,218]	[550,281]	-75	-128
21	[574,287]	[636,304]	-75	-128
22				

- In part 8, we are going to plot the lines one by one to the original input image that was plotted on part 7. For loop will iterate line length times so that ever line will be plotted one by one. In line 43, we assign x and y coordinates of each line's beginning and ending points into 'xy' variable. After that, we plot that line as it is. In lines 47 and 48, we plot the beginning of the line as yellow color and the ending of the line as red color.
- In part 9, we calculate the minimum and the maximum line length within the *lines* variable by comparing the length values of each lines with each other.
- At the end, since we had the shortest and the longest line segments already, we plot longest one with cyan color and shortest one with red color. Also we label the longest and shortest line segment in the plotted image. (Shown in 10).



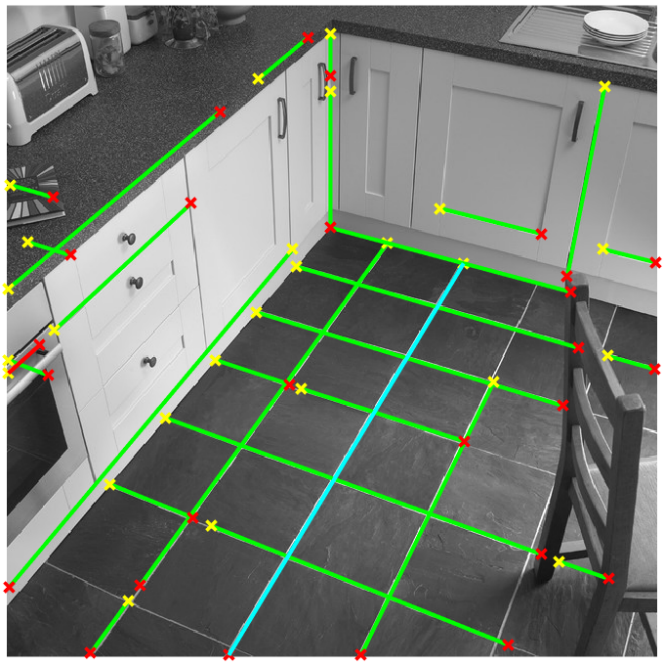
minimum length: 40.4969
maximum length: 450.3554

- The resulting image is shown above, notice that cyan colored line is the longest line that having 450.3554 length and red colored line is the shortest line having 40.4969 length.
- Now, before proceeding to Hough circles, let's use this algorithm on with another edge detection method and alter threshold value and see the variations.

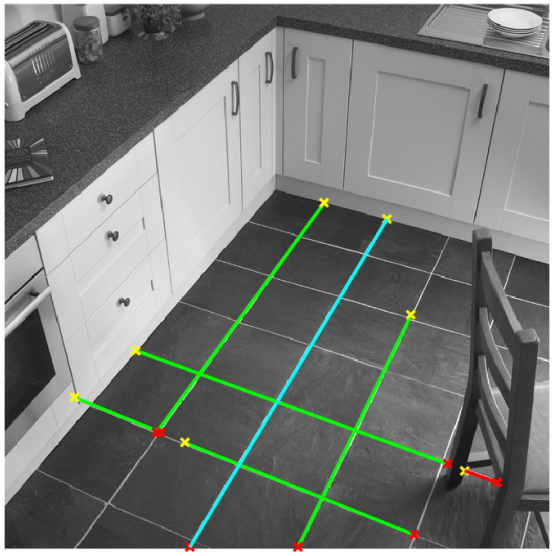


- Same image is used but this time with canny edge detector. Notice that edge image is more scrambled this time than the LoG method. So, the resulting line plotting will differ accordingly.

- Here, we have new line segments plotted with canny edge detector used and it is obvious that this time there are different lines detected than the other one, causing the shortest line segment change while the longest remains the same. As a conclusion, for this image specifically, I'd use LoG edge detection method since it gave more precise and proper result.



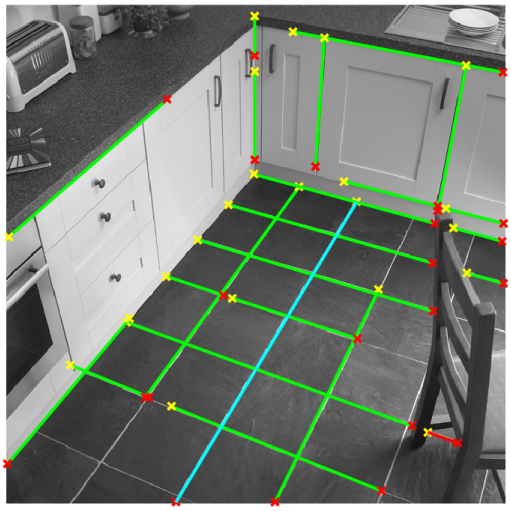
minimum length: 41.7732
maximum length: 448.9833



minimum length: 40.4969
maximum length: 450.3554

•Here, I have changed the threshold value from 0.5*max(H(:)) to 0.8*max(H(:)) so that most of the lines that are in the interval (0.5*max(H(:)), 0.8*max(H(:))) will be removed from the detected line segments.

- Also, when I change the number of edges that will be detected above threshold from 15 to 50, there is no change on line segments since there are at most 15 lines that are above given threshold already. So, whatever value I would increase the number of line segments that will be taken into account, there will be no difference after value 15.



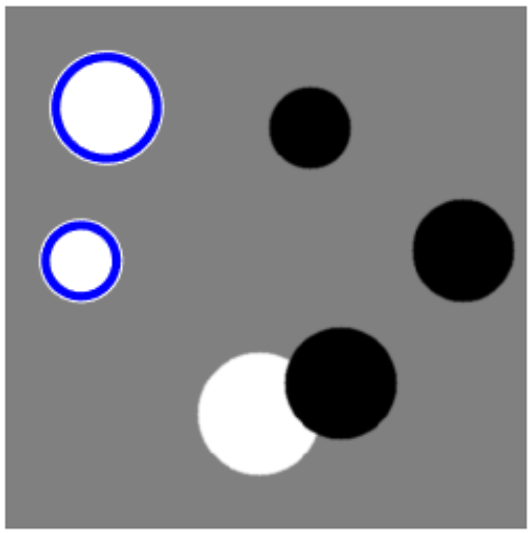
minimum length: 40.4969
maximum length: 450.3554

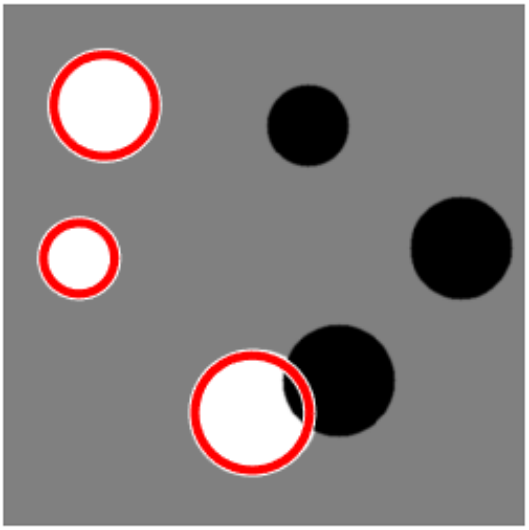
Also, elapsed time is calculated as 1.892110 seconds for this Hough lines algorithm to apply.

```
1  function [centersBright, radiiBright, centersDark, radiiDark] = lab4houghcircles(img)
2  [~,~,ch] = size(img);
3  if (ch==3)
4      img = rgb2gray(img);
5  end
6
7  Rmin = 20;
8  Rmax = 60;
9
10 [centers1, radii1] = imfindcircles(img,[Rmin Rmax]);
11 [centers2, radii2] = imfindcircles(img, [Rmin Rmax], 'Sensitivity', 0.9);
12
13 %for bright circles
14 [centersBright, radiiBright] = imfindcircles(img,[Rmin Rmax], 'Sensitivity', 0.9, 'ObjectPolarity', 'bright');
15 %for dark circles
16 [centersDark, radiiDark] = imfindcircles(img,[Rmin Rmax], 'Sensitivity', 0.9, 'ObjectPolarity', 'dark');
17
18 %plottings
19 figure
20 subplot(1,3,1);
21 imshow(img);
22 viscircles(centers1, radii1,'EdgeColor','b');
23
24 subplot(1,3,2);
25 imshow(img);
26 viscircles(centers2, radii2,'EdgeColor','r');
27
28 subplot(1,3,3);
29 imshow(img);
30 viscircles(centersBright, radiiBright,'EdgeColor','b');
31 viscircles(centersDark, radiiDark,'LineStyle','--' , 'EdgeColor','r');
32 end
33
```

- This function takes an arbitrary image ***img*** and find the circles inside it.
- As always, we start with extracting the channel size of the input image and if it is 3, which implies image being rgb scaled, we convert it to gray-scaled image.
- Since the radius values interval that will be detected is already given as between [20,60], I set Rmin:20 and RMax: 60.
- In line 10, I detected the circles by using ***imfindcircles()*** built-in function. ***[centers,radii] = imfindcircles(A,radiusRange)*** finds circles which are in the range specified by radiusRange. The additional output argument, *radii*, contains the estimated radius corresponding to each circle center in *centers*.
- In line 11, again I detected the circles by using ***imfindcircles()*** built-in function. However, this time I set the sensitivity value to 0.9 and obtained more circles as it will be shown in outputs below.

- Here what we have is the output that was obtained by using [centers1, radii1] = imfindcircles(img,[Rmin Rmax]); line of code which is the one without specifying the sensitivity. As it is shown, the bottom bright circle which is distorted by other black circle is not found as a circle. Note that these outputs are shown due to the lines between 20 and 22.

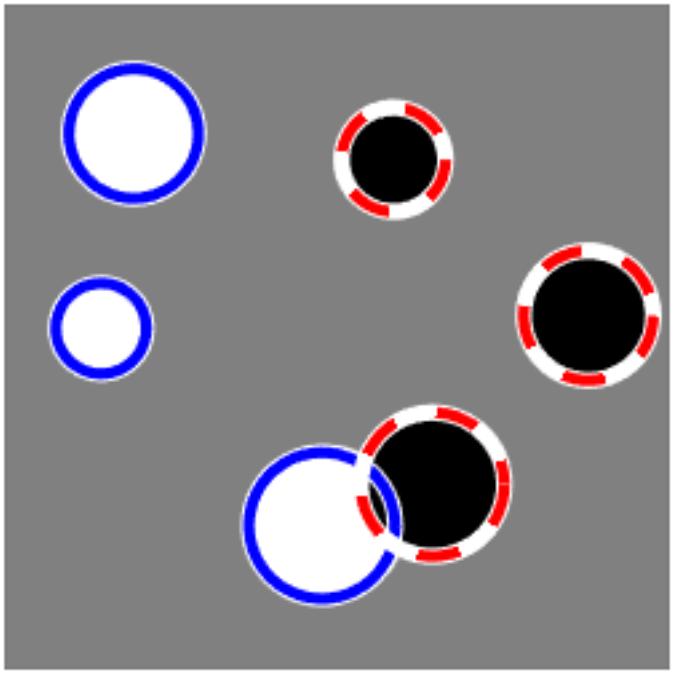




•However, when I utilized [centers2, radii2] = imfindcircles(img, [Rmin Rmax], 'Sensitivity', 0.9); line of code, I detected that bottom bright circle as circle although it was distorted by another black circle and is not seen as whole circle in the image. By altering the sensitivity value, I get different circles that was not detected before.

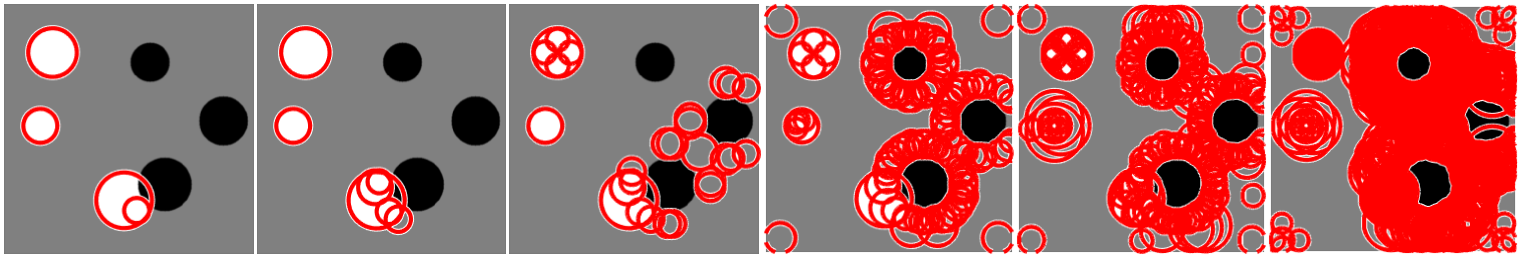
•Note that these outputs are shown due to the lines between 24 and 27.

- In the upcoming parts, in other words in lines 13 to 16; I detected bright and dark circles with also setting the sensitivity values to 0.9. I again used ***imfindcircles()*** built-in function in order to get those circles but this time beside setting sensitivity to 0.9 I also set the 'ObjectPolarity' attribute to 'bright' and 'dark' respectively. In the end, what I got are;



• Now, we see the bright and dark circles that are labelled as blue and red respectively. Note that since we set sensitivity value to 0.9, we detected bottom bright circle as well as the previous example. Note that these outputs are shown due to the lines between 28 and 32.

LETS ELABORATE MORE ON THE VALUES;



The sensitivity values are varying from 0.95 to 1 from left to right respectively.