ASSIGNMENT A9

Shantnu Kakkar

CS 6320, Spring 2016

April 06, 2016

Section 1: Intro:

This assignment is based on the model fitting methods learnt in the lecture. The goal is to develop a set of functions that can retrieve lines from the Hall image given in the assignment. Hough transform would play a crucial role to achieve this goal. Hough transform is a way to fit a structure to a set of tokens (for example, a line to a set of points). In Hough, we cluster tokens that could lie in same structure by recording all structures on which each token lies and then look for structures that get many votes. Hough uses the parametric representation of a line as follows:

$$x\ cos\Theta + y\ sin\Theta + r = 0$$

where,

r is perpendicular distance from origin to line.

Θ is the angle made by x axis and the perpendicular line from origin to the line in consideration.

From this representation, any pair of (Θ, r) can be used to represent a unique line. Also, we can find a family of a lines passing through a point (a, b) as follows:

$$r = -a\ cos\Theta - \ bsin\Theta$$

This equation is used in making an accumulator array, which maintains the count of the r values corresponding to the Θ. For example, the following figure shows a Hough accumulator corresponding to a set of random points.

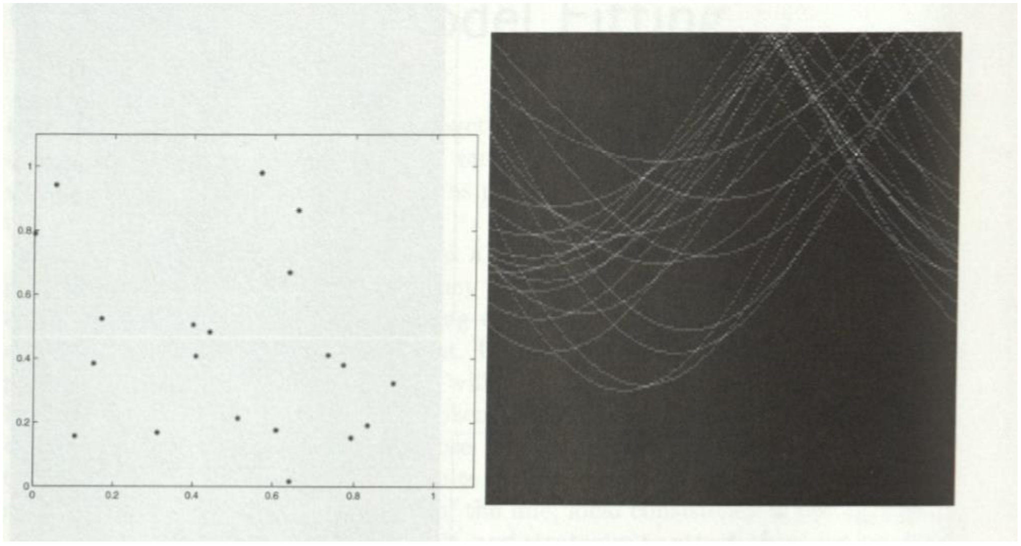


Figure 1 This figure shows a Hough transform for a set of random points. Left side is the set of random points and right is the corresponding Accumulator

For this problem, we are given a Hall image and we need to find lines in it. We are required to implement a total least squares function for line parameter estimation function, a Hough transform function, as well as some helper functions, and a CS5320 line segs, a function to return line segments based on the line finding results of other functions. After the implementation, I will be answering the following question:

- How robust is the Hough transform to noise in image?

Section 2: Method:

- Matlab is used to carry out the experiments.
- I will be using the combo, CS5320_line_between and CS5320_Hough_lines function, provided by professor to see some of my results,
- **Please note that to get better results for my CS5320_Hough with CS5320_Hough_draw_pts and CS5320_Hough_lines functions, it is important to do the local_max of the returned accumulator array.**
- **To check robustness of Hough, I am going to superimpose Gaussian noise with different standard deviations, on my image and see if the Hough is able to find lines. I will be monitoring the value of my variance in noise at which the Hough starts showing very bad results. The following is my method:**
    - *Read image and convert to gray*

      hall4 = imread('hall4.jpg');

      hall4gOriginal = rgb2gray(hall4);
    - *Set variance vs = [1:100:1000];*
    - *For v_index = 1:num_vs*

      v = vs(v_index);

      *hall4gModified(:,:) = double(hall4gOriginal) + sqrt(v)*randn(size(hall4gOriginal));*

      [H4,H4pts] = CS5320_Hough(double(hall4gModified));

      H4 = CS5320_local_max(H4,2);

      linesp = CS5320_Hough_draw_pts(hall4gModified,H4,H4pts,10,0);

      figure;combo(mat2gray(hall4gModified),linesp);
- I will be using the following formulas to find recall and precision for success optimization :

  precision = |{lines in image} intersect {lines retrieved}|/|{lines retrieved} -
  recall = |{lines in image} intersect {lines retrieved}|/{lines in image}|

Following functions are implemented

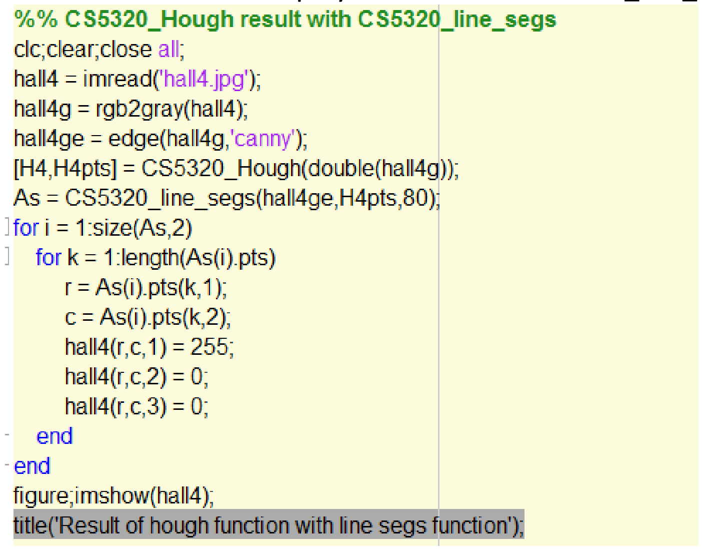
- CV_total_LS – This function is the total least squares method to fit best line to points. It takes input as the x coordinates of points and y coordinates of point coefficients of best fit line ax + by + c = 0, and error measure (sum of squares of distance of points to line). I have ensured that Eigen vector corresponding to lowest Eigen value is returned.
- CS5320_Hough – This function produces the Hough transform, that is the Hough accumulator array (rho values; theta values) and a struct which contains points which contributed to line.

The rho index is from [-length of image diagonal, + length of diagonal] and theta is from (0,180). The input to this function is a gray image.

- CS5320_Hough_draw_pts – This function takes in input as the original image, the Hough accumulator array and points, and a flag to draw the image or not (1 means draw, 0 don't). It returns a mask of n lines.
- CS5320_plot_line – This function plots line given parameters, and required x and y range values
- CS5320_line_segs – This function extract line segments from Hough info. Its input are edge image, Hough points array and minimum segment length. It's output is a vector of segments. Each element in the segment contains points that contributed to the segment, rho parameter of line, theta parameter of line and end points of line segment.

In addition to the above functions there is a script named **Verification** which contains how I call every function. Please note that I have divided this script into various cells (sections). Please run individual section for every function.

Please note that to display the results of CS5320_line_segs I have done following:



```
%% CS5320_Hough result with CS5320_line_segs
clc;clear;close all;
hall4 = imread('hall4.jpg');
hall4g = rgb2gray(hall4);
hall4ge = edge(hall4g,'canny');
[H4,H4pts] = CS5320_Hough(double(hall4g));
As = CS5320_line_segs(hall4ge,H4pts,80);
for i = 1:size(As,2)
   for k = 1:length(As(i).pts)
       r = As(i).pts(k,1);
       c = As(i).pts(k,2);
       hall4(r,c,1) = 255;
       hall4(r,c,2) = 0;
       hall4(r,c,3) = 0;
   end
end
figure;imshow(hall4);
title('Result of hough function with line segs function');
```

The following algorithm has been used for total least squares:

- *Initilize points and sum of errors*

  p = [];

  s = 0;
- *Make matrix as follows:*

  xMean = mean(x);

```
xSquareMean = mean(x.^2);
yMean = mean(y);
ySquareMean = mean(y.^2);
xyMean = mean(x.*y);

A = [xSquareMean - (xMean)^2, xyMean - xMean*yMean;
 xyMean - xMean*yMean, ySquareMean - (yMean)^2];
```

- *Find eigen vector corresponding to smallest Eigen vector as follows:*

```
 [V,D] = eigs(A);
[Vsorted,indexes] = sort(diag(D));
index = indexes(1);
 sol = V(:,index);
```

- *Save line parameters*

```
p(1) = sol(1);
p(2) = sol(2);
p(3) = -sol(1)*xMean - sol(2)*yMean;
```

- *Compute sum of errors*

```
for i = 1:length(x)
        dist = (abs( p(1)*x(i) + p(2)*y(i) + p(3)))/(sqrt( p(1)^2 + p(2)^2));
        s = s + dist;
end
```

The following algorithm has been used for CS5320_Hough

- Make edge imageim = edge(imo,'canny');
- Find size of image
    - [nr,nc] = size(im);
- Find diagonal
    - rSize = ceil(sqrt(nr^2 + nc^2));
- MAKE INDEXES ARRAY rArray = -rSize:rSize;
- Set rSize = 2*diagonal + 1
    - rSize = length(rArray);
- Find mid point mid = ceil((rSize-1)/2);
- Initialize outputs
    - pts(rSize,180).pts = [];
    - H = zeros(rSize,180);
- Loop for r = 1:nr
    - Loop for c = 1:nc

```
    if im(r,c)==1

        x = c;

        y = nr-r+1;

        loop for t = 1:180

                rho = -x*cosd(t) - y*sind(t);
```

```
rhoIndex = mid + ceil(rho) + 1;
H(rhoIndex,t) = H(rhoIndex,t)+1;
pts(rhoIndex,t).pts = [pts(rhoIndex,t).pts ; [r,c]];
```

The following method is used for draw_points function:

- *Sort the accumulator array*
  - *[H4Temp,I] = sort(H(:), 'descend');*
  - *ptsTemp = pts(I);*
- *Find image size[nr,nc] = size(im);*
- *Initialize lines = zeros(nr,nc);*
- *Loop for i = 1:n*

```
        A = ptsTemp(i).pts;
        [nrA,ncA] = size(A);
        Loop  for j = 1:nrA
              r = A(j,1);
              c = A(j,2);
              lines(r,c) = 1;
```

- *if dr == 1*

```
        combo(mat2gray(im),lines);
```

The following method is used for plot_lines

if x1~=x2

- *Make xVector = x1:0.1:x2;*
- *Make yVector = (-p(1)*xVector - p(3))/p(2);*
- *plot(xVector,yVector);*

else

- *Make yVector = y1:0.1:y2;*
- *Make xVector = (-p(2)*yVector - p(3))/p(1);*
- *plot(xVector,yVector);*

The following method is used for line_segs function

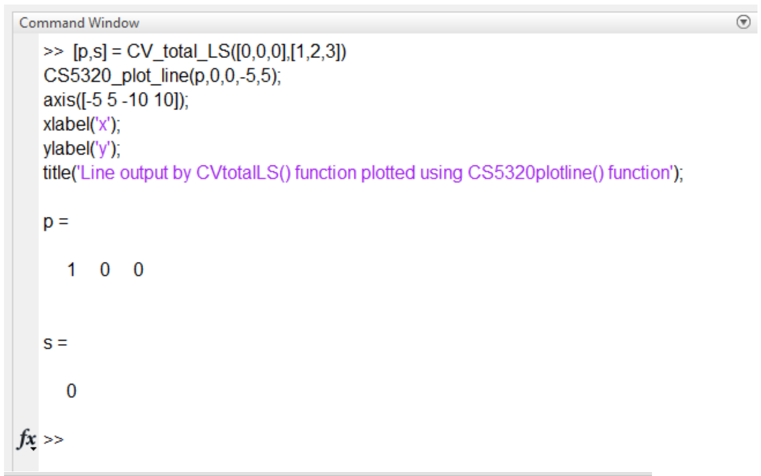
- *Find [rSize,t] = size(Hpts);*
- *Find middle index: mid = ceil((rSize-1)/2);*
- *Initialize*
  - *segments = [];*
  - *index = 0;*
- *Loop for rhoIndex = 1:rSize*
  - *Loop for thetaIndex = 1:t*
    - *Find: k = size(Hpts(rhoIndex,thetaIndex).pts,1);*

```
▪ if k>min_len
      ➢ increment: index = index + 1;
      ➢ set: segments(index).rho = rhoIndex-mid-1;
      ➢ set: segments(index).theta = thetaIndex;

      ➢ initialize: distArray = [];

      ➢ Make distance array as follows:
            for r = 1:k
                for c = 1:k
                    dist = norm(Hpts(rhoIndex,thetaIndex).pts(r,:) -...
                        Hpts(rhoIndex,thetaIndex).pts(c,:));
                    distArray(r,c) = dist;

      ➢ [M,I] = max(distArray(:));
      ➢ [I_row, I_col] = ind2sub(size(distArray),I);

      ➢ segments(index).endpt1 = Hpts(rhoIndex,thetaIndex).pts(I_row,:);

      ➢ segments(index).endpt2 = Hpts(rhoIndex,thetaIndex).pts(I_col,:);

      ➢ [rows,cols] = CS5320_line_between(segments(index).endpt1,...
        segments(index).endpt2);
      ➢ segments(index).pts = [rows',cols'];
```

Section 3: Verification:

- ### Testing CV_total_LS
  - ➢ To test this function, I find a vertical line, a horizontal line, and a diagonal line. I look whether my line parameters are coming correct or not. I also plot the line that I get using my CS5320_plot_line function

1) Testing vertical line: We take points (0,1), (0,2) and (0,3). From the figure, we can see that we are getting line parameters as 1,0,0. So, the equation is *x = 0,* which means a straight vertical line along y-axis. Thus, we have obtained the right line. Plotting further illustrates that my CS5320_plot_line() is correct. The SSE is 0, which is correct since line passes through these points.



```
Command Window
>> [p,s] = CV_total_LS([0,0,0],[1,2,3])
CS5320_plot_line(p,0,0,-5,5);
axis([-5 5 -10 10]);
xlabel('x');
ylabel('y');
title('Line output by CVtotalLS() function plotted using CS5320plotline() function');

p =

   1   0   0


s =

   0

fx >>
```

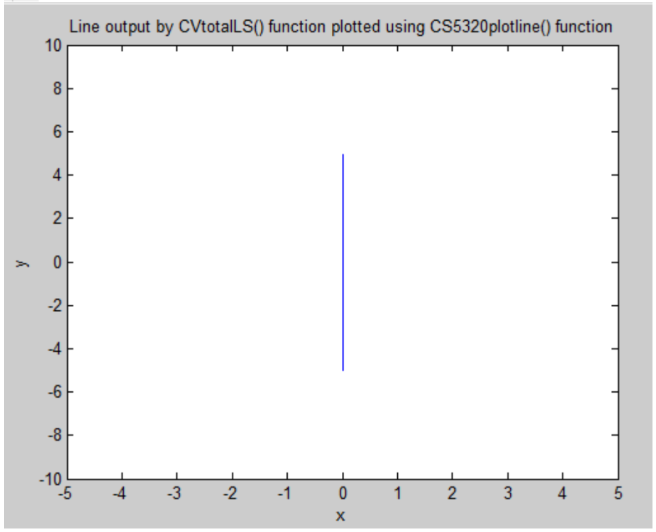
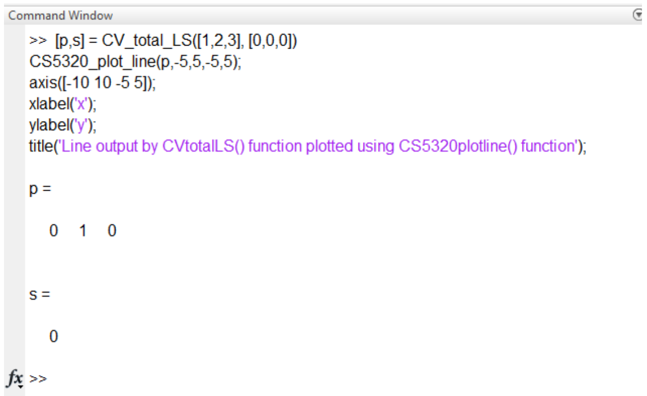


Figure 2 Testing vertical line Line output by CV_totalLS() function plotted using CS5320_plot_line() function

2) *Testing horizontal line:* We take points (1,0),(2,0) and (3,0). From the figure, we can see that we are getting line parameters as 0,1,0. So, the equation is *y = 0,* which means a straight horizontal line along x-axis. Thus, we have obtained the right line. Plotting further illustrates that my CS5320_plot_line() is correct. The SSE is 0, which is correct since line passes through these points.



```
Command Window
>> [p,s] = CV_total_LS([1,2,3], [0,0,0])
CS5320_plot_line(p,-5,5,-5,5);
axis([-10 10 -5 5]);
xlabel('x');
ylabel('y');
title('Line output by CVtotalLS() function plotted using CS5320plotline() function');

p =

     0     1     0


s =

     0

fx >>
```

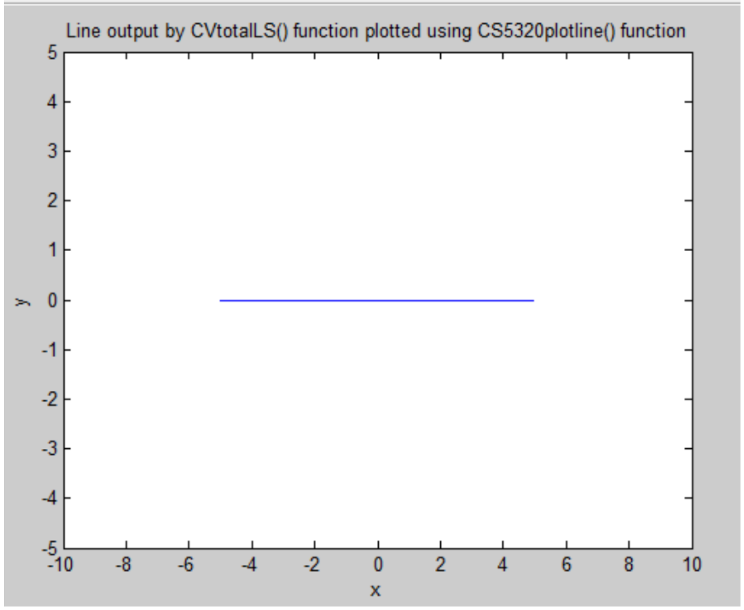
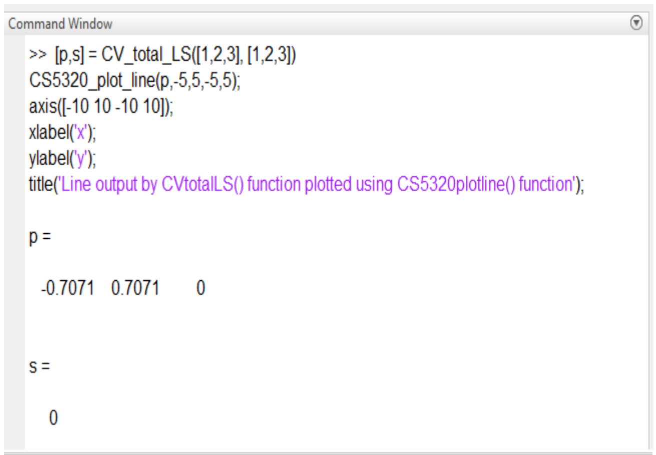


Figure 3 Testing horizontal Line output by CV_totalLS() function plotted using CS5320_plot_line() function

3) *Testing Diagonal line.* We take points (1,1),(2,2) and (3,3). From the figure, we can see that we are getting line parameters as -0.7071,0.7071,0. So, the equation is *x = y,* which means a diagonal line inclined equally to both the axis. Thus, we have obtained the right line. Plotting further illustrates that my CS5320_plot_line() is correct. The SSE is 0, which is correct since line passes through these points.



```
Command Window
>> [p,s] = CV_total_LS([1,2,3], [1,2,3])
CS5320_plot_line(p,-5,5,-5,5);
axis([-10 10 -10 10]);
xlabel('x');
ylabel('y');
title('Line output by CVtotalLS() function plotted using CS5320plotline() function');

p =

  -0.7071    0.7071         0


s =

     0
```

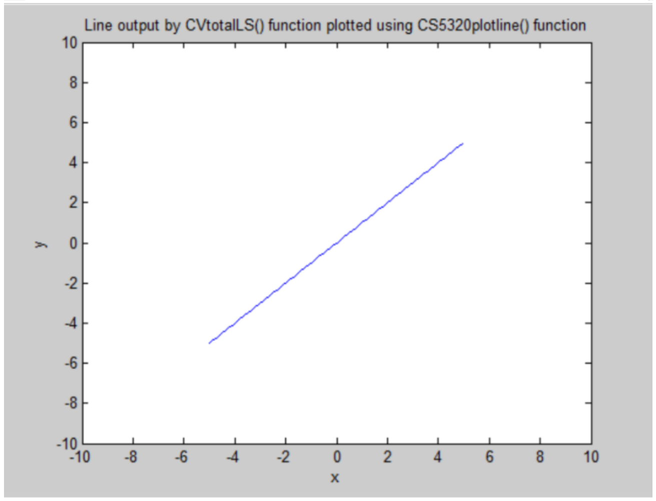
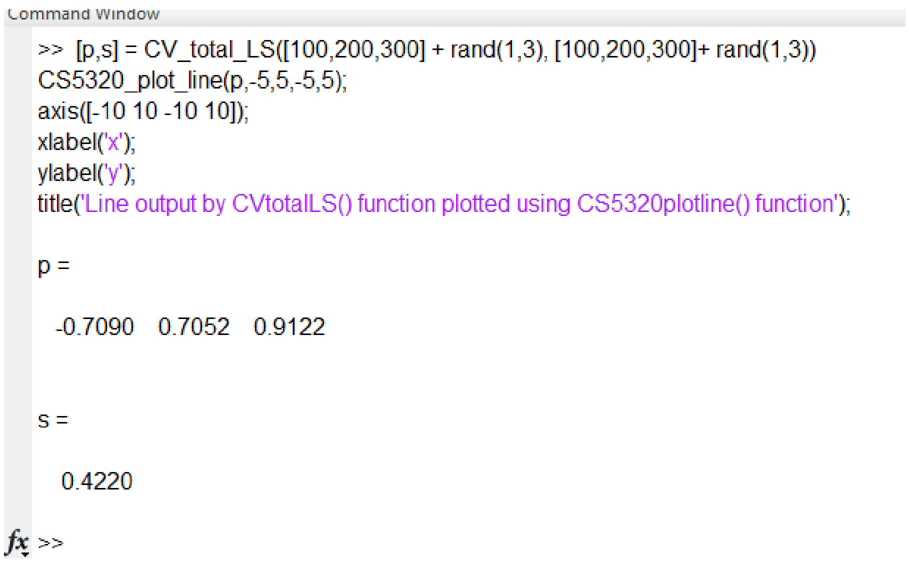


Figure 4 Testing diagonal line Line output by CV_totalLS() function plotted using CS5320_plot_line() function

4) Testing how correct is my error. Super imposing noise on my points, which are (100,100), (200,200) and (300,300). I should get a line with slope nearly same as line x = y, and some error value. The following figure shows that I am getting the desired results. See the line doesnot exactly pass through the origin now. It is little deviated.



```
Command Window
>> [p,s] = CV_total_LS([100,200,300] + rand(1,3), [100,200,300]+ rand(1,3))
CS5320_plot_line(p,-5,5,-5,5);
axis([-10 10 -10 10]);
xlabel('x');
ylabel('y');
title('Line output by CVtotalLS() function plotted using CS5320plotline() function');

p =

   -0.7090    0.7052    0.9122


s =

    0.4220

fx >>
```

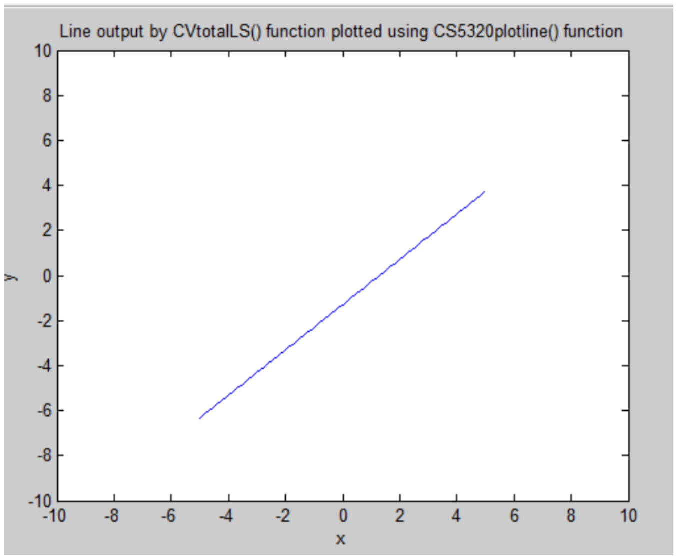
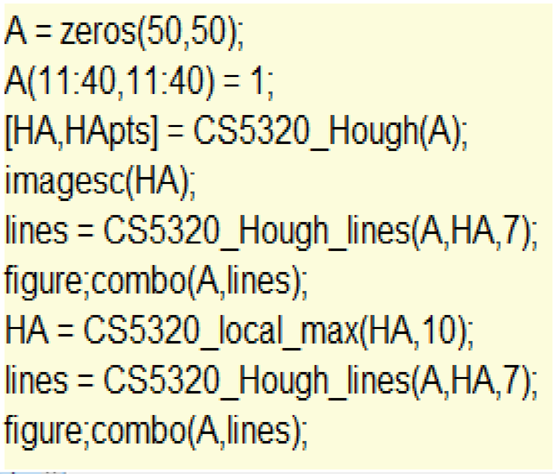
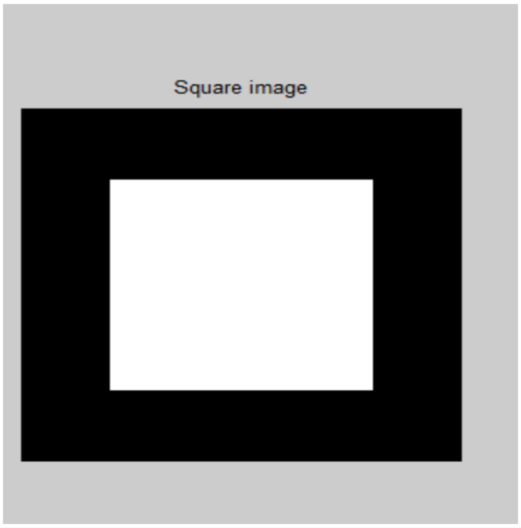


Figure 5 Testing error in line parameters by adding gaussian noise to points.

- **Testing CS5320_Hough**
  - I will be testing my Hough on a simpler image. The following figures shows the script executed and the corresponding results. The results are really good as can be seen below:



```
A = zeros(50,50);
A(11:40,11:40) = 1;
[HA,HApts] = CS5320_Hough(A);
imagesc(HA);
lines = CS5320_Hough_lines(A,HA,7);
figure;combo(A,lines);
HA = CS5320_local_max(HA,10);
lines = CS5320_Hough_lines(A,HA,7);
figure;combo(A,lines);
```



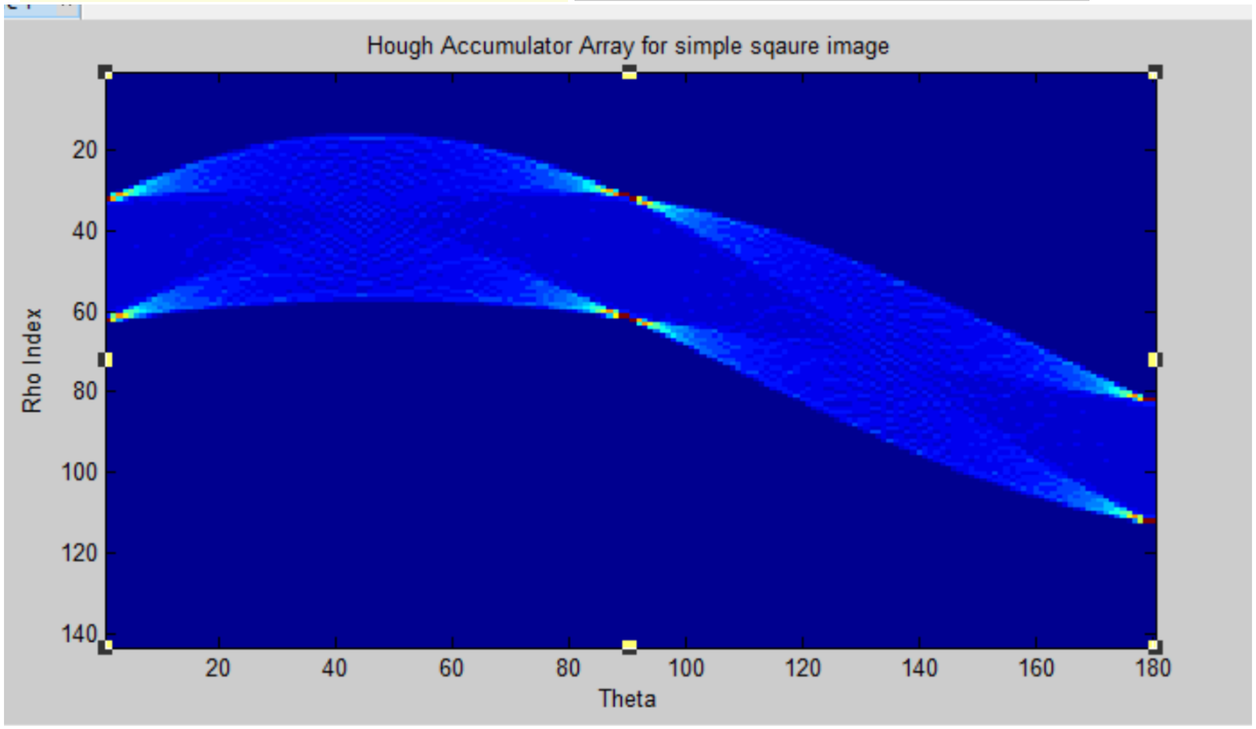


Figure 6 Hough Accumulator Array for simple square image

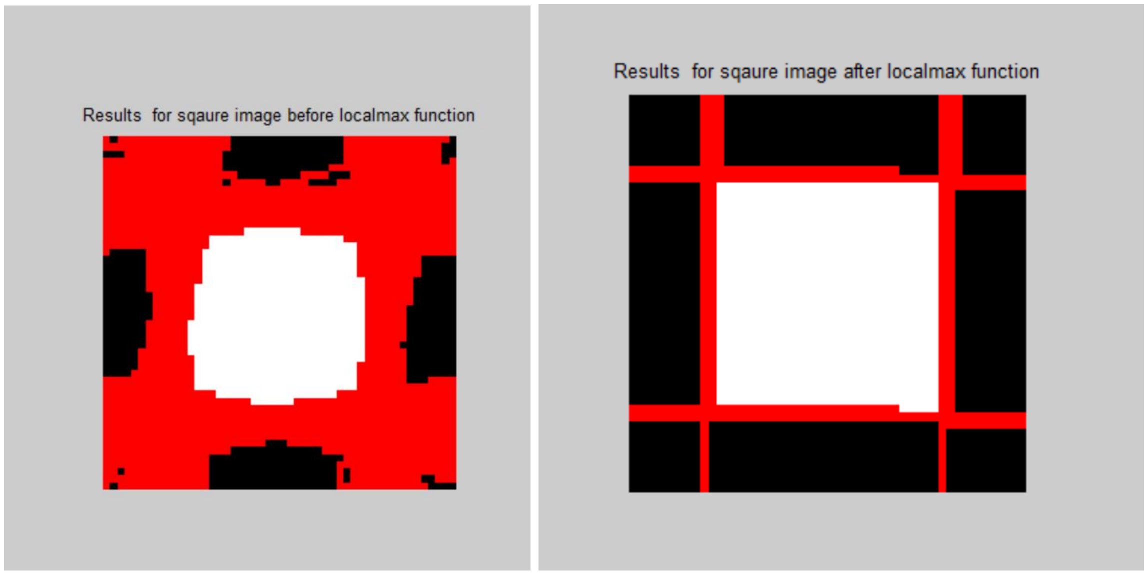
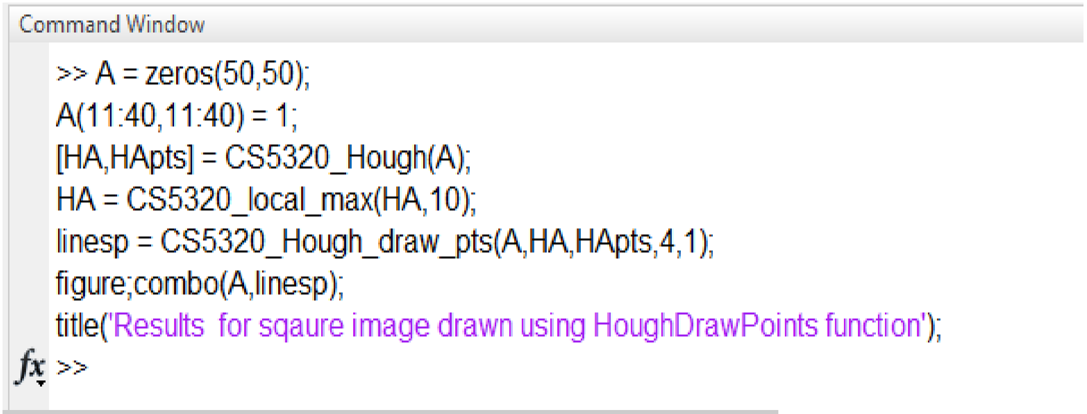


Figure 7 Results for square image with and without local_max function

- **Testing CS5320_Hough_draw_pts**
  - Let's test this function on the hough accumulator that I obtained for square image above and see whether this function finds the masks correctly or not. The following script is executed and the results are as follows. These results show that Hough_draw_points correctly finds the mask.



```
>> A = zeros(50,50);
A(11:40,11:40) = 1;
[HA,HApts] = CS5320_Hough(A);
HA = CS5320_local_max(HA,10);
linesp = CS5320_Hough_draw_pts(A,HA,HApts,4,1);
figure;combo(A,linesp);
title('Results  for sqaure image drawn using HoughDrawPoints function');
>>
```

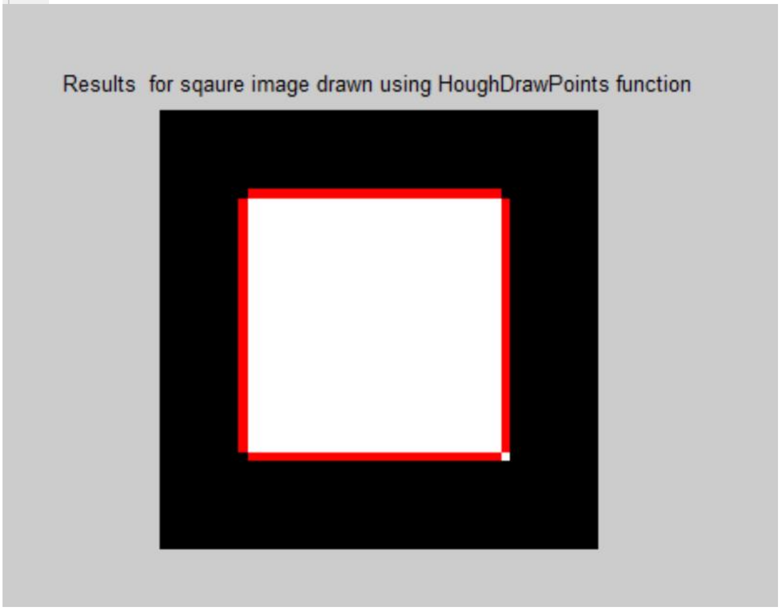


Figure 8 Results for square image drawn using Hough_draw_points function

- Verification for three lines: linesp = CS5320_Hough_draw_pts(A,HA,HApts,3,1);

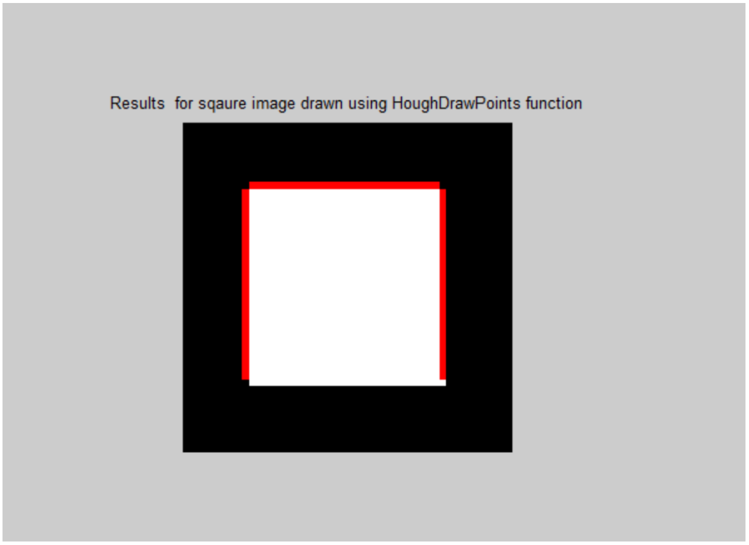
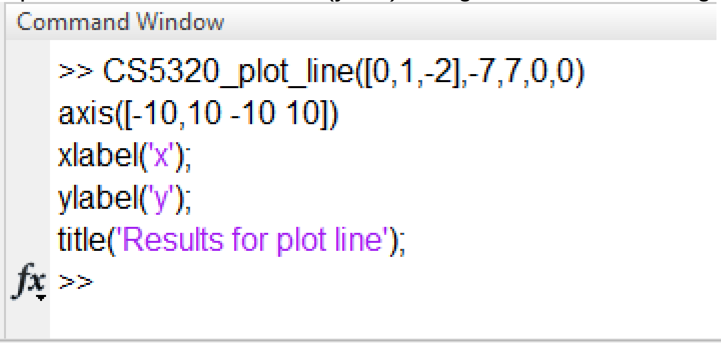


Figure 9 Verifcation of draw_points for 3 lines.

- **Testing CS5320_plot_line**
    - ➢ Let's test this function on simple cases as follows. First, test for the case when x1 is not equal to x2. We draw a line (y = *2)* using x values in the range -7 to 7 as follows:



```
>> CS5320_plot_line([0,1,-2],-7,7,0,0)
axis([-10,10 -10 10])
xlabel('x');
ylabel('y');
title('Results for plot line');
>>
```

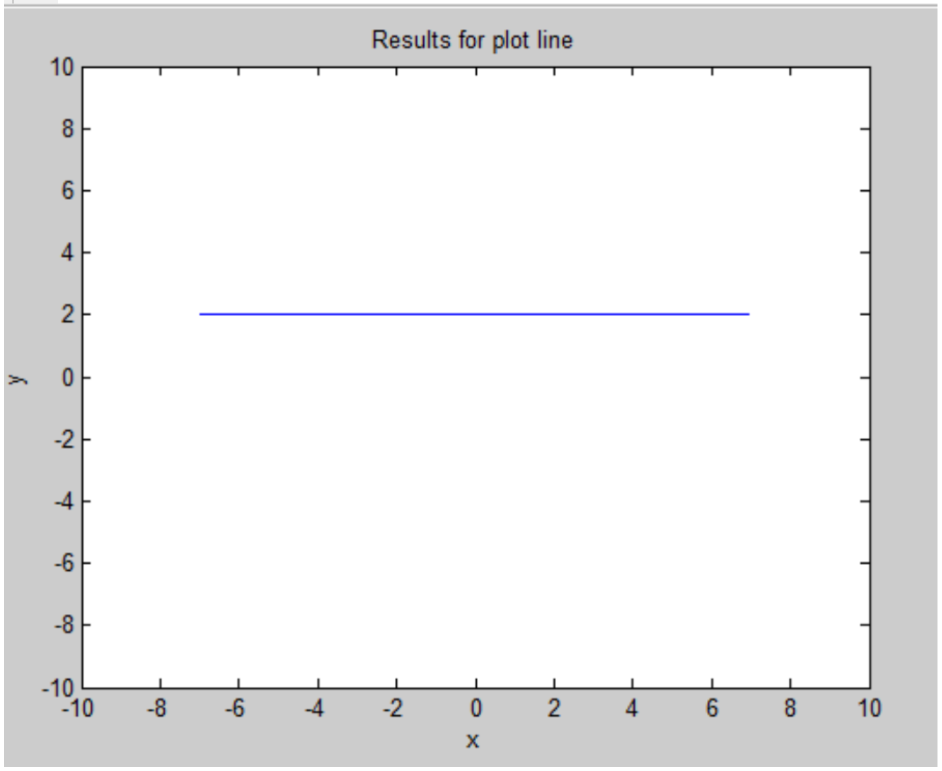
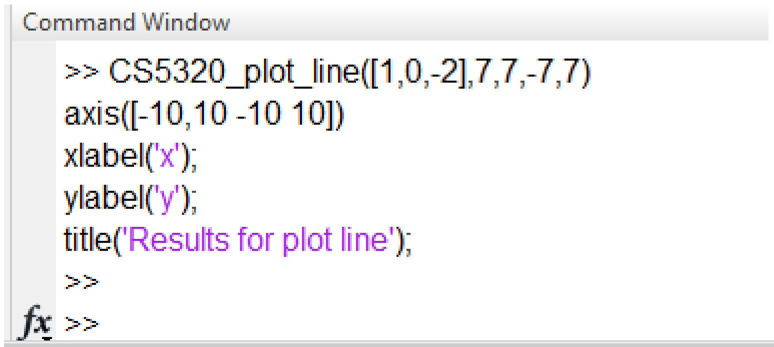


Figure 10 Testing plot_line function for line y=2 in the range of x values from -7 to 7

- ➢ Now let's test for the case when x1 is equal to x2. We draw a line (*x = 2)* using yvalues in the range -7 to 7 as follows:

- 

```
Command Window
>> CS5320_plot_line([1,0,-2],7,7,-7,7)
axis([-10,10 -10 10])
xlabel('x');
ylabel('y');
title('Results for plot line');
>>
fx >>
```

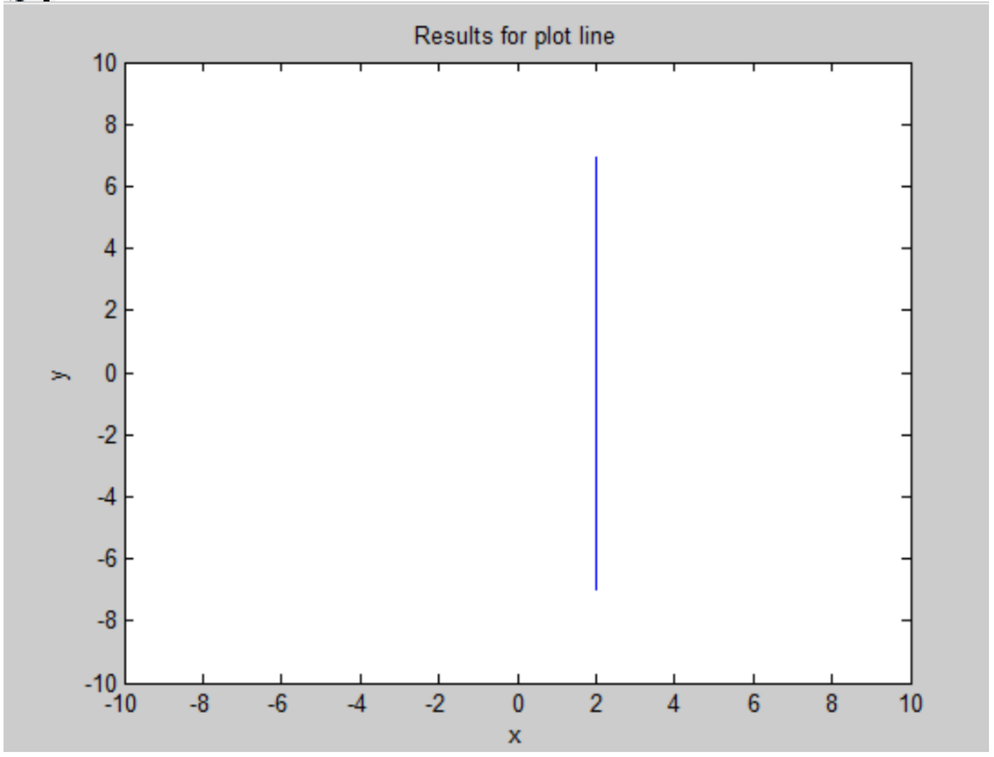
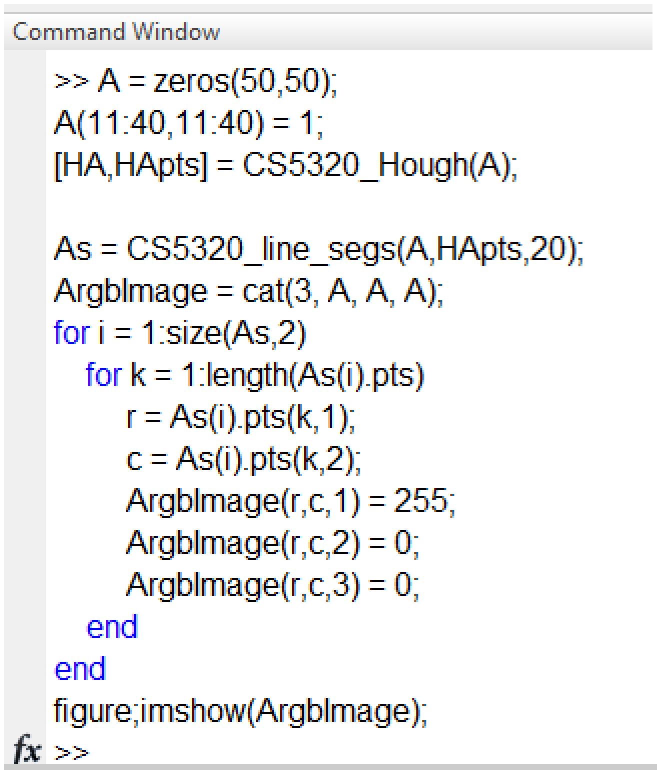


Figure 119 Testing plot_line function for line x=2 in the range of y values from -7 to 7, and same x values

- **Testing CS5320_line_segs**

  - Let’s test this function on simple square image above. The following script is executed, and we can see the results as below:



Command Window

```
>> A = zeros(50,50);
A(11:40,11:40) = 1;
[HA,HApts] = CS5320_Hough(A);

As = CS5320_line_segs(A,HApts,20);
ArgbImage = cat(3, A, A, A);
for i = 1:size(As,2)
   for k = 1:length(As(i).pts)
       r = As(i).pts(k,1);
       c = As(i).pts(k,2);
       ArgbImage(r,c,1) = 255;
       ArgbImage(r,c,2) = 0;
       ArgbImage(r,c,3) = 0;
   end
end
figure;imshow(ArgbImage);
fx >>
```

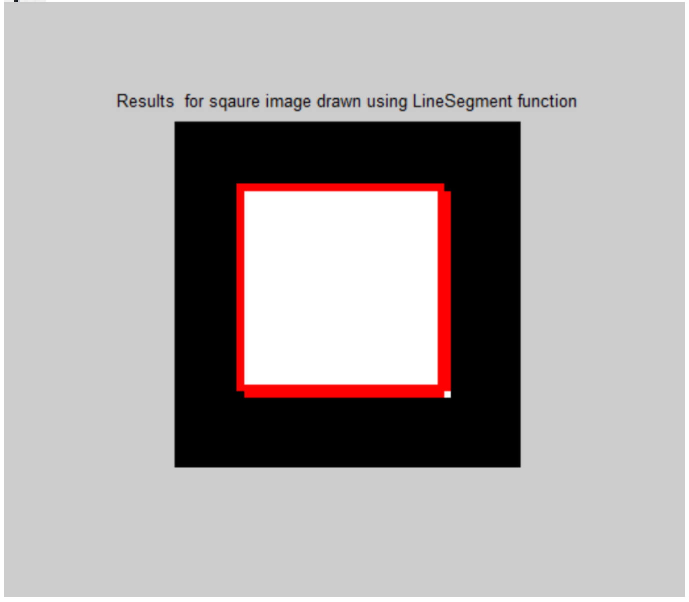
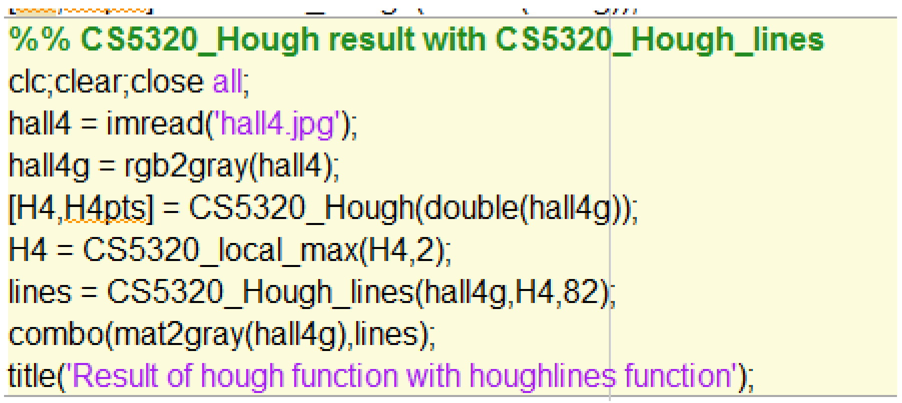


Figure 12 'Results for square image drawn using CS5320_line_segs function

Section 4: Data:

The following figures show my results for the hall image for various functions:



```
%% CS5320_Hough result with CS5320_Hough_lines
clc;clear;close all;
hall4 = imread('hall4.jpg');
hall4g = rgb2gray(hall4);
[H4,H4pts] = CS5320_Hough(double(hall4g));
H4 = CS5320_local_max(H4,2);
lines = CS5320_Hough_lines(hall4g,H4,82);
combo(mat2gray(hall4g),lines);
title('Result of hough function with houghlines function');
```

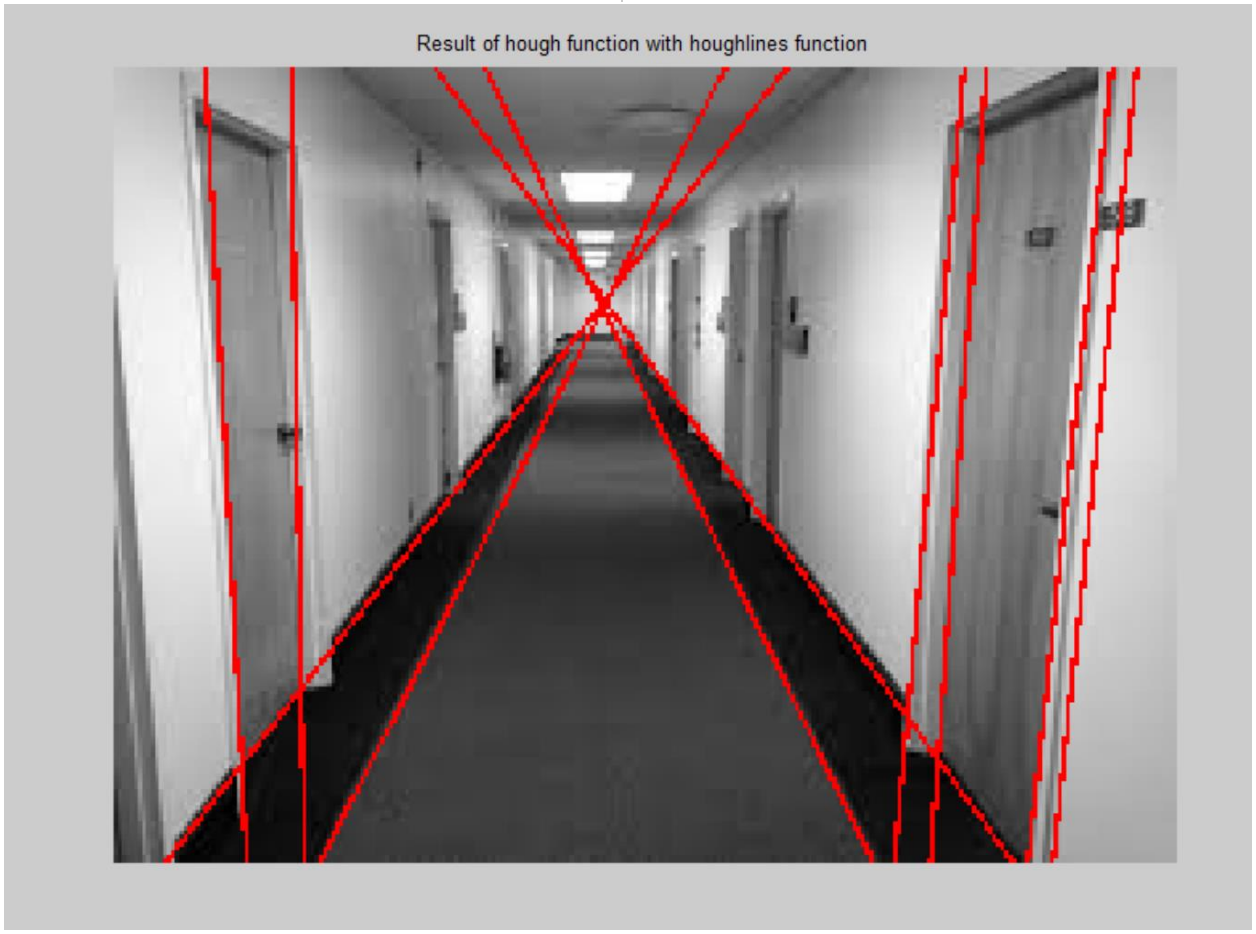


Figure 13 Result of CS5320_Hough function with CS5320_Hough_liness function

```
%% CS5320_Hough result with CS5320_Hough_draw_pts
hall4 = imread('hall4.jpg');
hall4g = rgb2gray(hall4);
[H4,H4pts] = CS5320_Hough(double(hall4g));
 H4 = CS5320_local_max(H4,2);
% load('ResultHall4.mat');
linesp = CS5320_Hough_draw_pts(hall4,H4,H4pts,10,1);
%  combo(mat2gray(hall4g),linesp);
title('Result of hough function with drawpoints function');
```

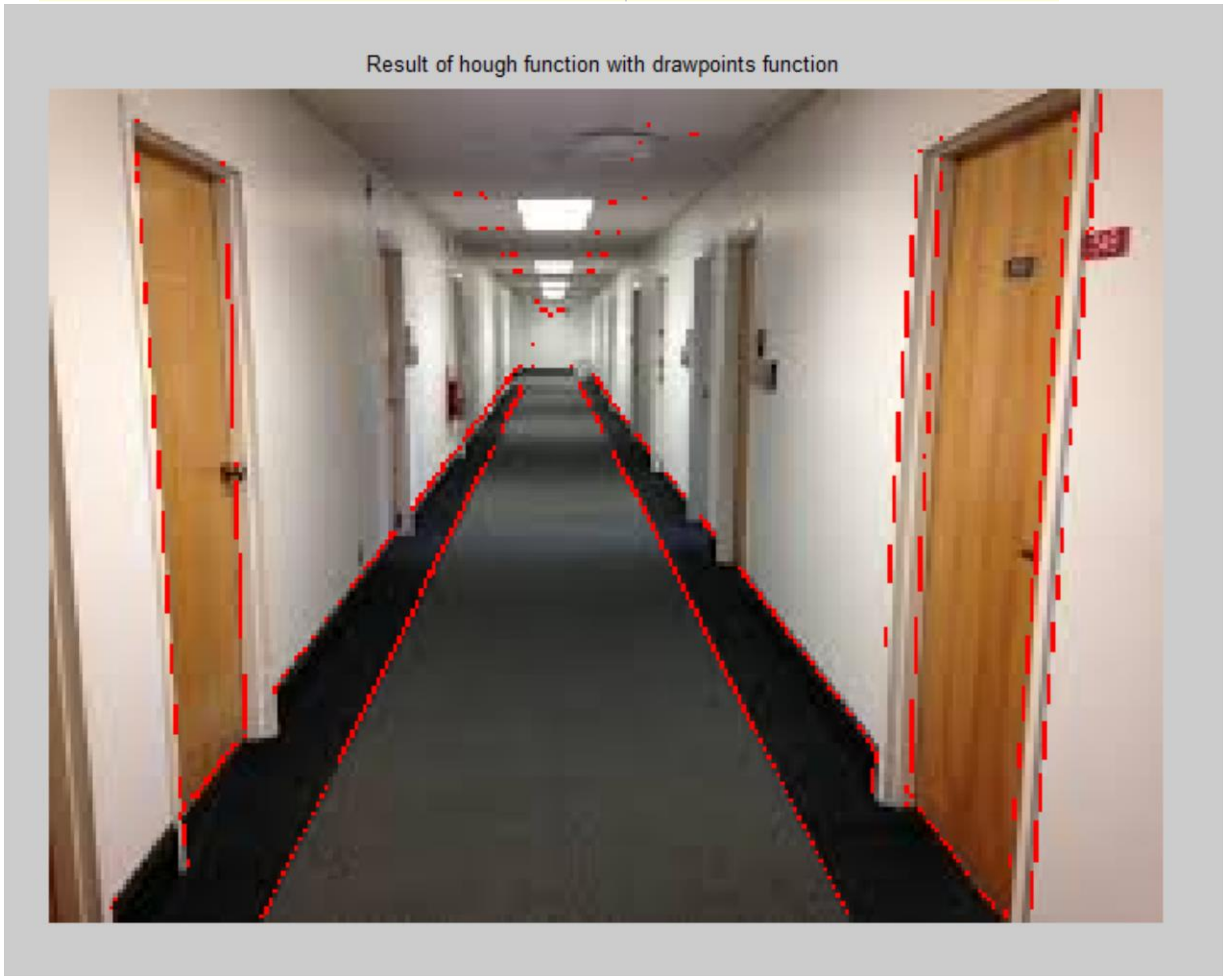
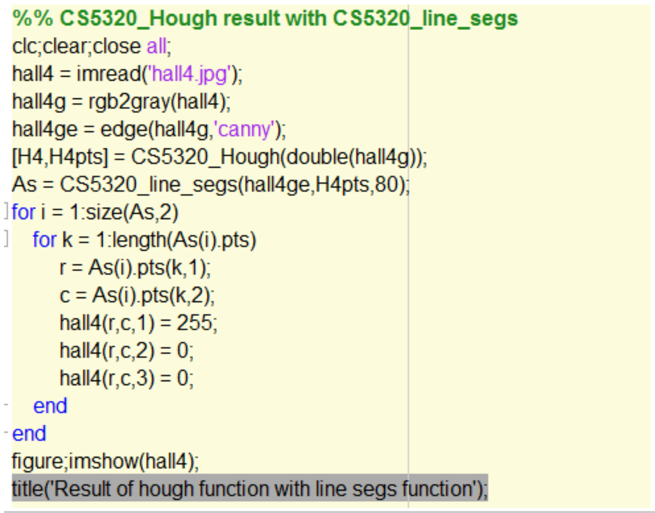


Figure 14 'Result of CS5320_Hough function with CS5320_Hough_draw_pts function



```
%% CS5320_Hough result with CS5320_line_segs
clc;clear;close all;
hall4 = imread('hall4.jpg');
hall4g = rgb2gray(hall4);
hall4ge = edge(hall4g,'canny');
[H4,H4pts] = CS5320_Hough(double(hall4g));
As = CS5320_line_segs(hall4ge,H4pts,80);
for i = 1:size(As,2)
  for k = 1:length(As(i).pts)
      r = As(i).pts(k,1);
      c = As(i).pts(k,2);
      hall4(r,c,1) = 255;
      hall4(r,c,2) = 0;
      hall4(r,c,3) = 0;
  end
end
figure;imshow(hall4);
title('Result of hough function with line segs function');
```

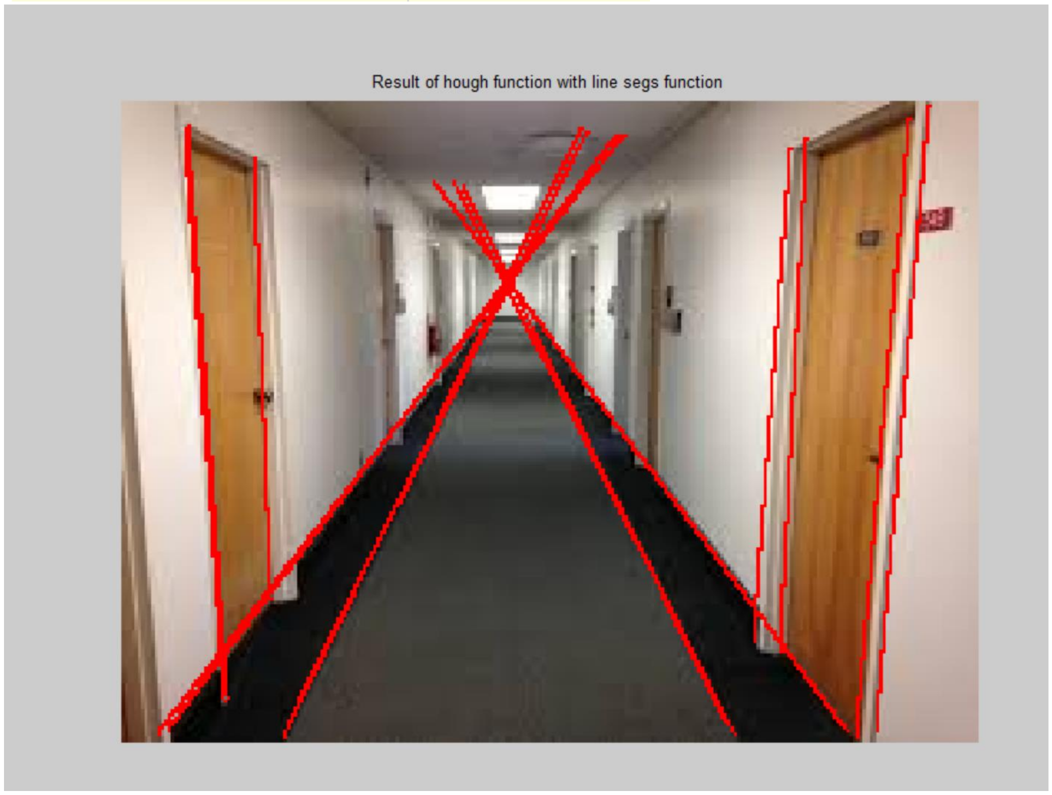
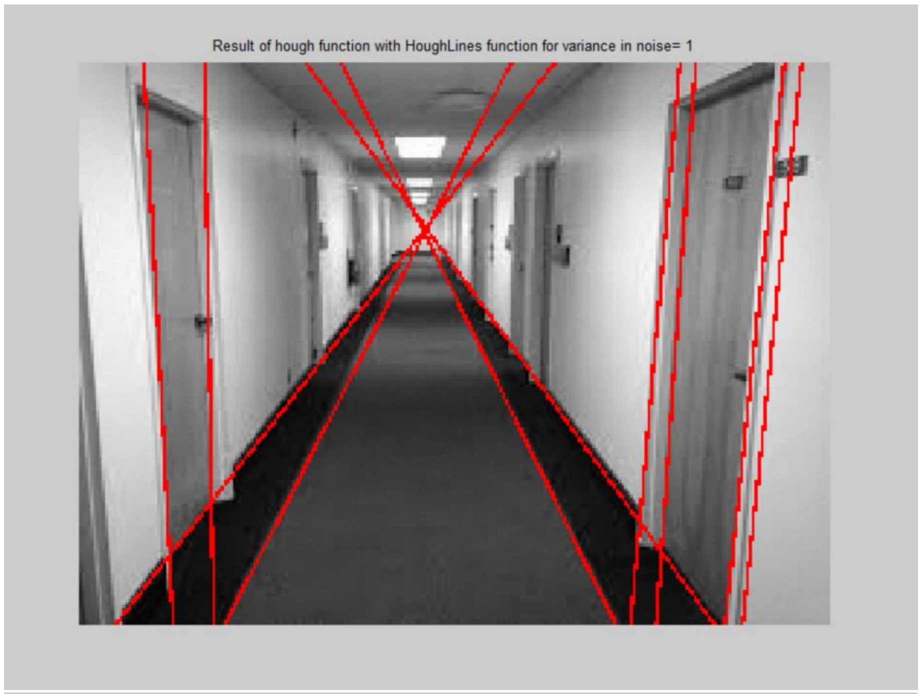
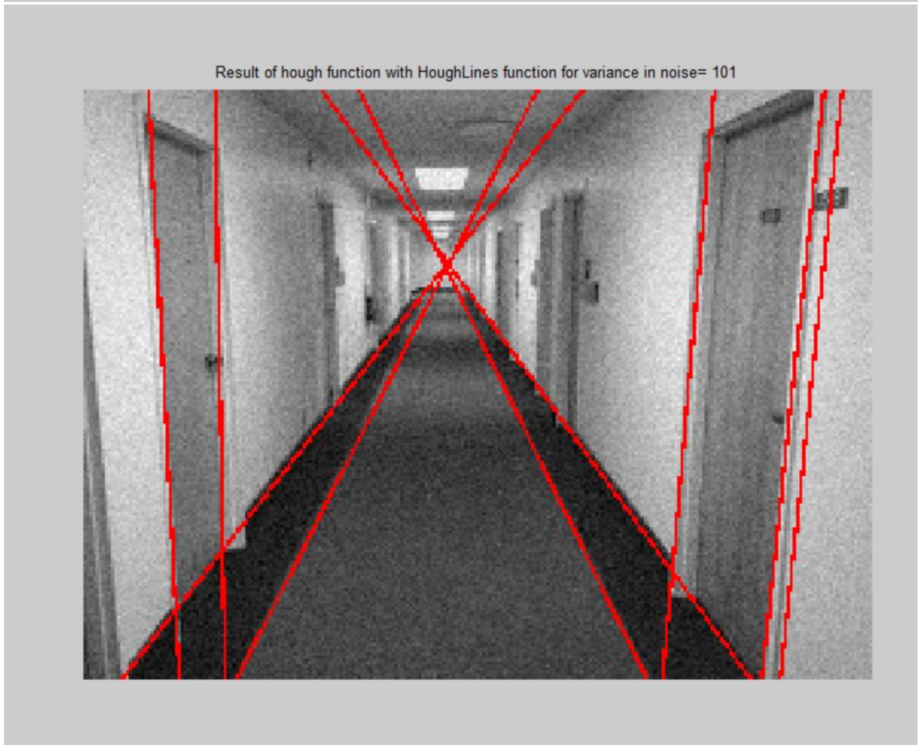
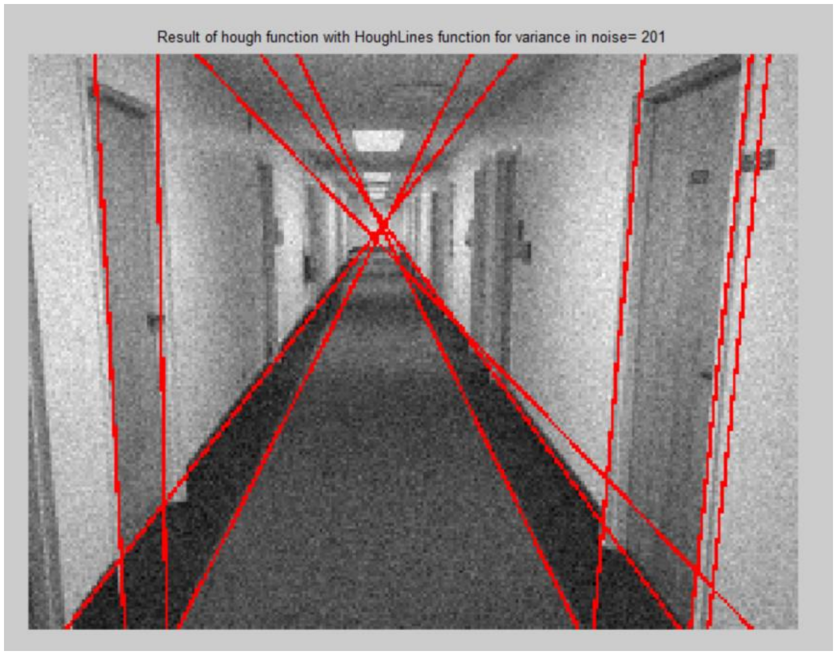


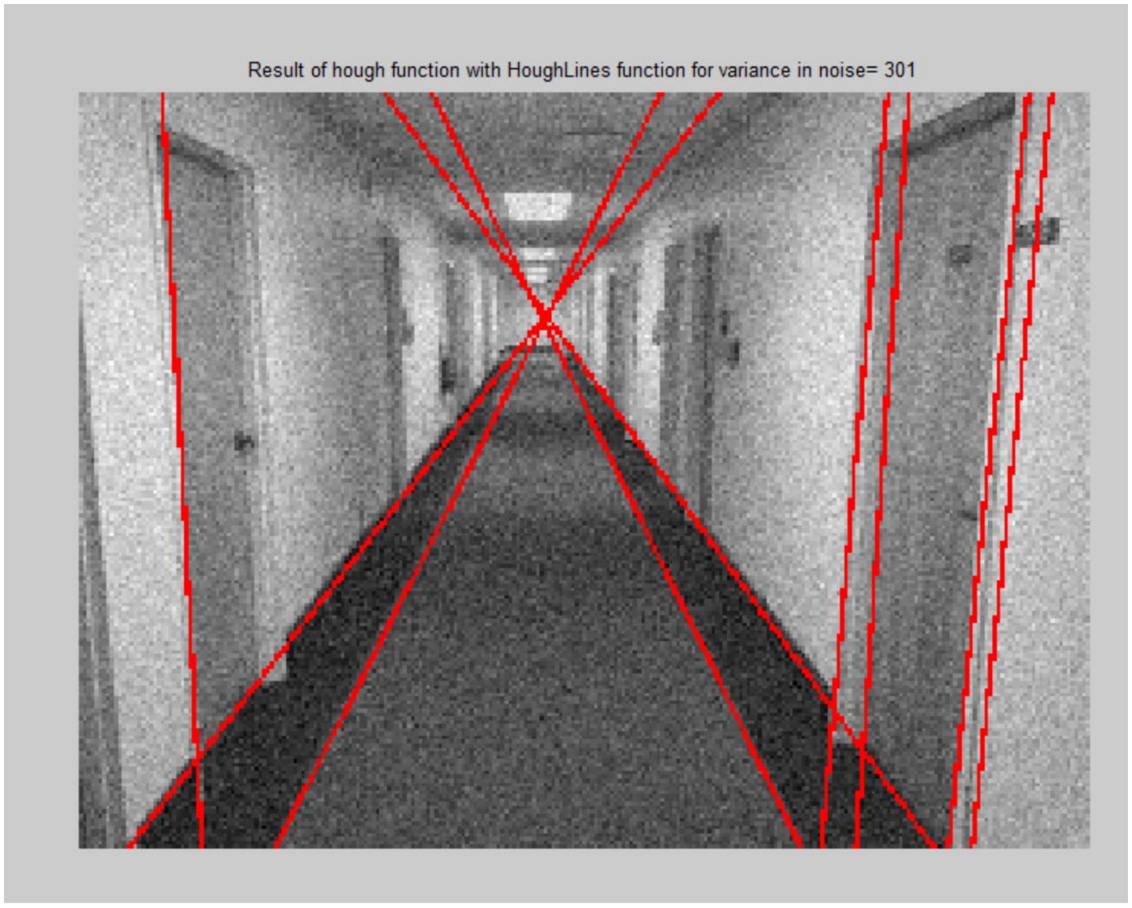
Figure 15 Result of CS5320_Houghfunction with CS5320_line_segs function

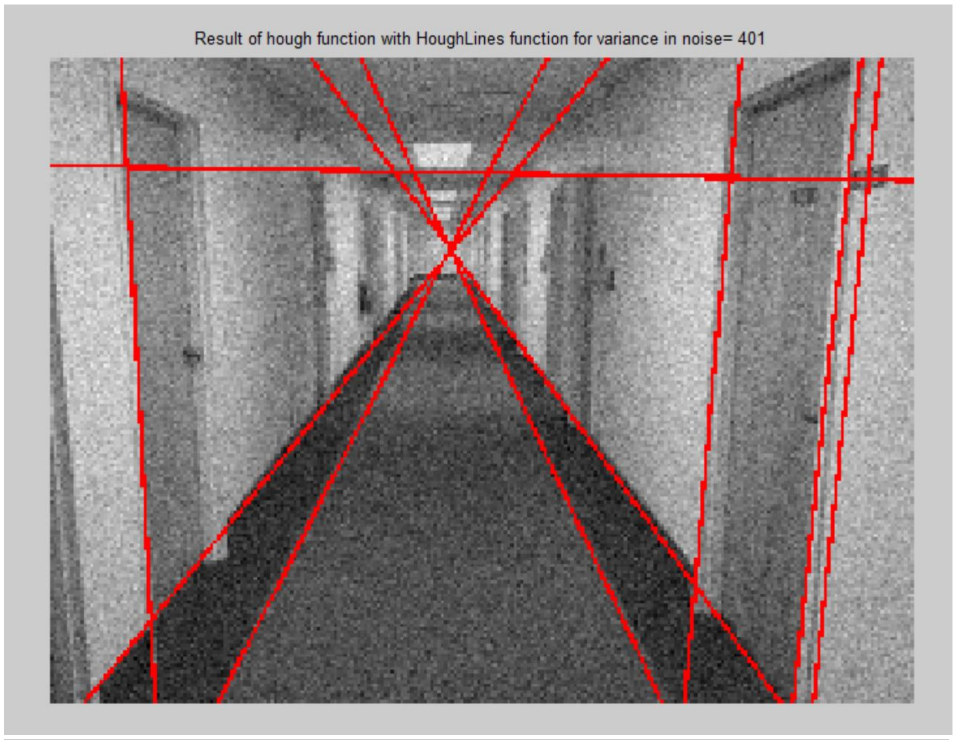
The following is the data for the noise that was imposed as described in methods section.

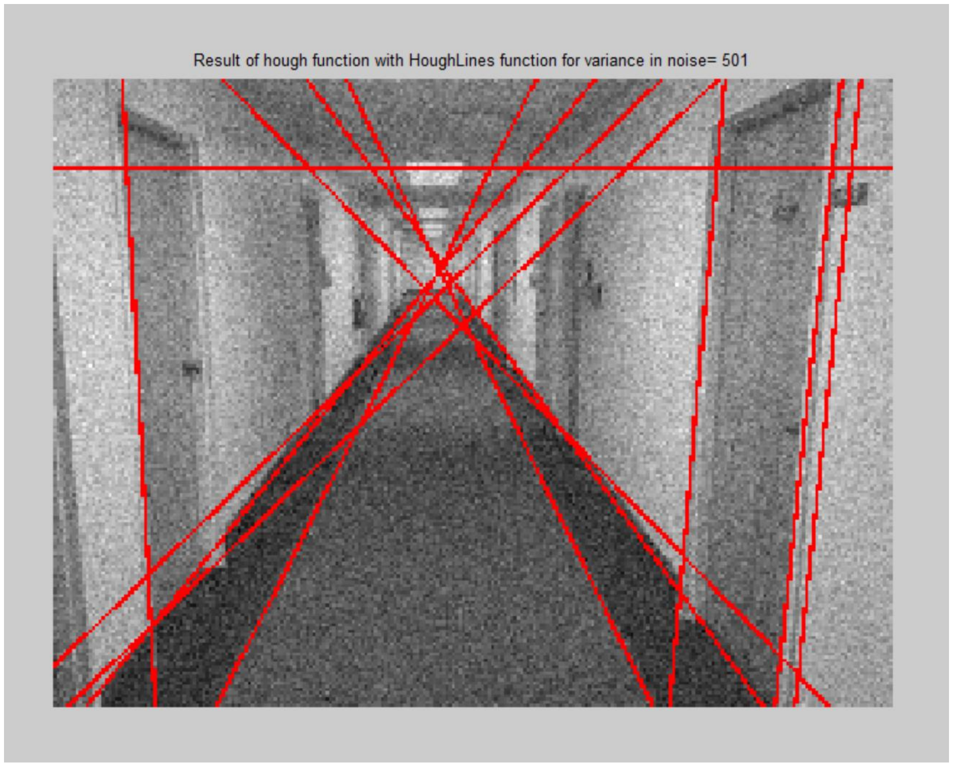

Result of hough function with HoughLines function for variance in noise= 1

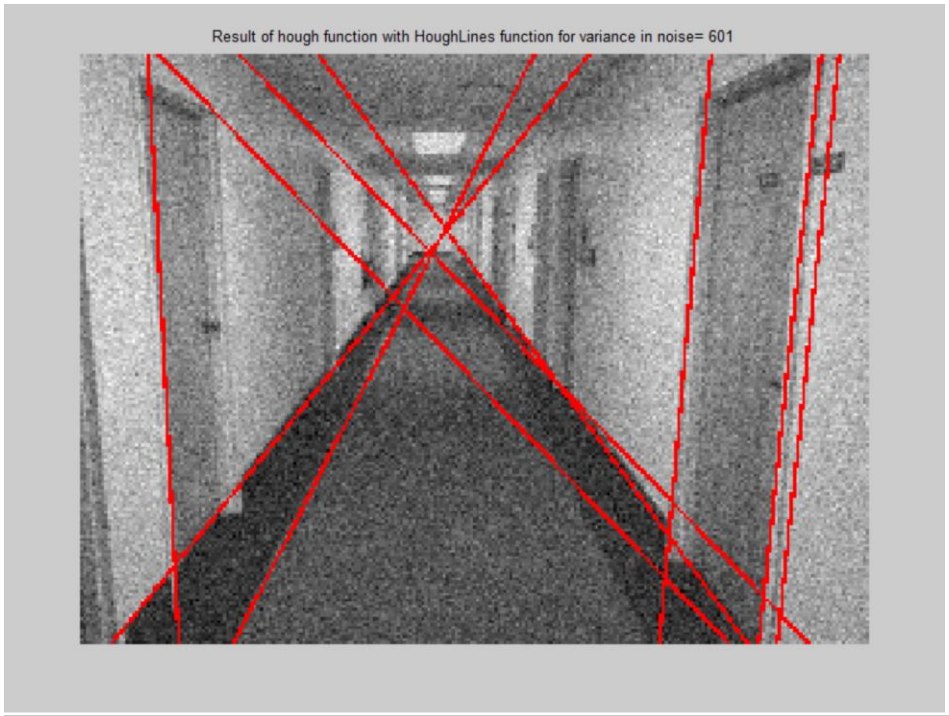

Result of hough function with HoughLines function for variance in noise= 101

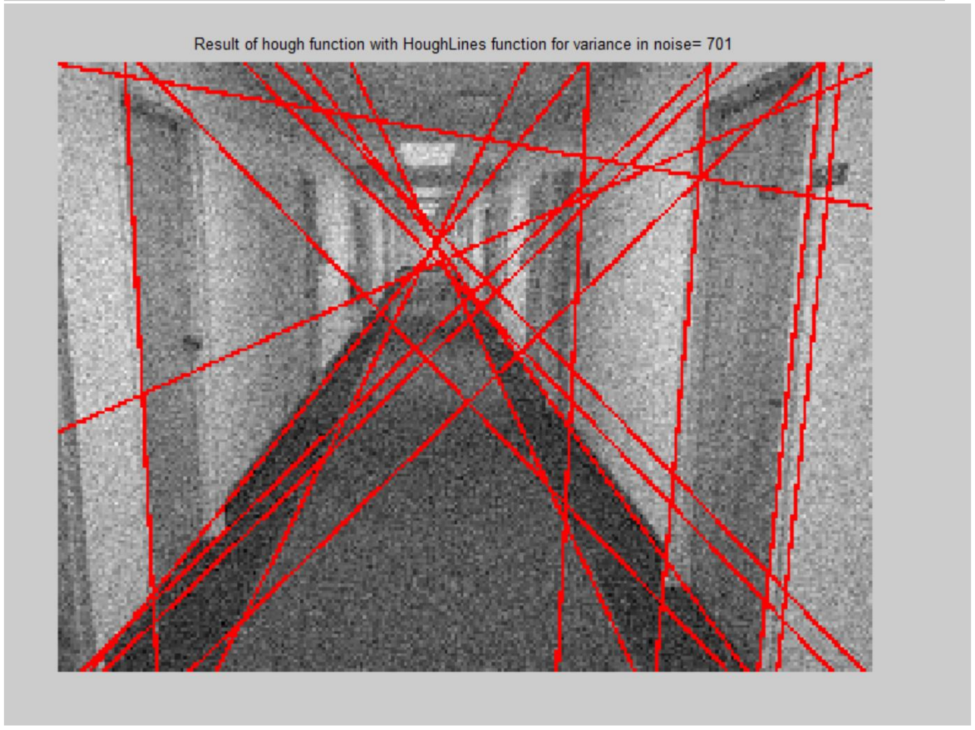

Result of hough function with HoughLines function for variance in noise= 201

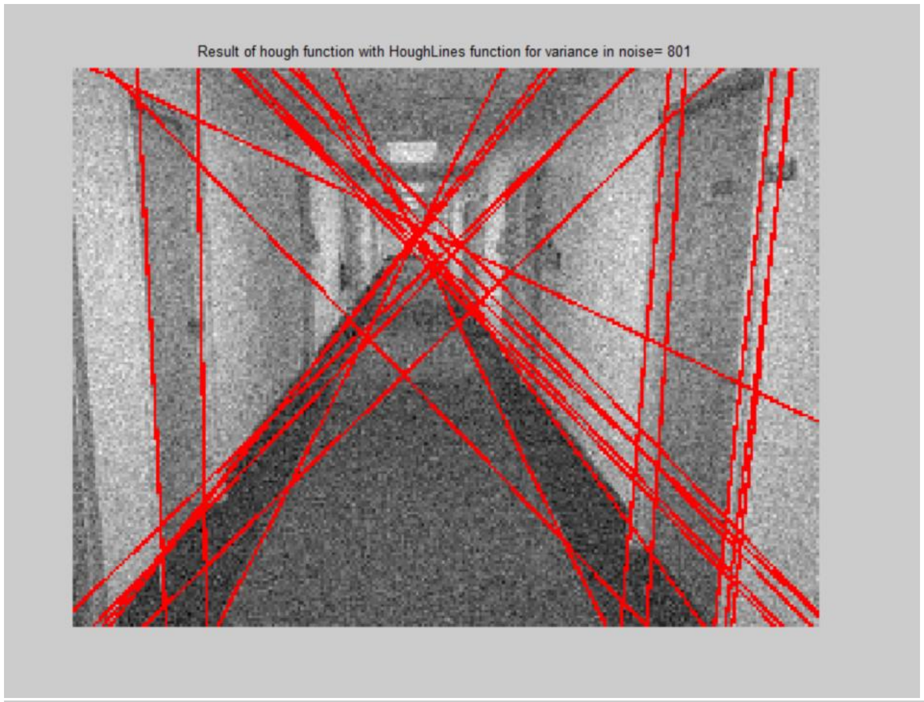

Result of hough function with HoughLines function for variance in noise= 301

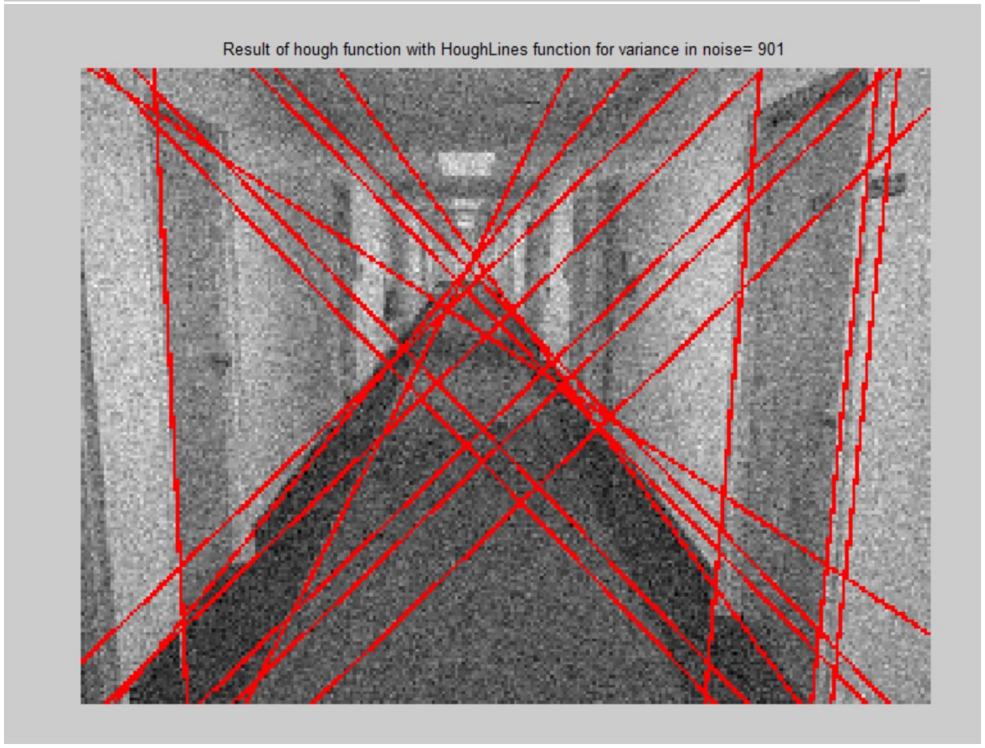

Result of hough function with HoughLines function for variance in noise= 401


Result of hough function with HoughLines function for variance in noise= 501


Result of hough function with HoughLines function for variance in noise= 601


Result of hough function with HoughLines function for variance in noise= 701


Result of hough function with HoughLines function for variance in noise= 801


Result of hough function with HoughLines function for variance in noise= 901

Section 5: Analysis:

The following is the summary of Precision and recall for various noise levels. Before that, we establish the ground truth that means the lines

that we consider to be actually a line in the image. Note, I have established ground truth this way but it could vary according to requirement. There are 16 total lines according to this ground truth.

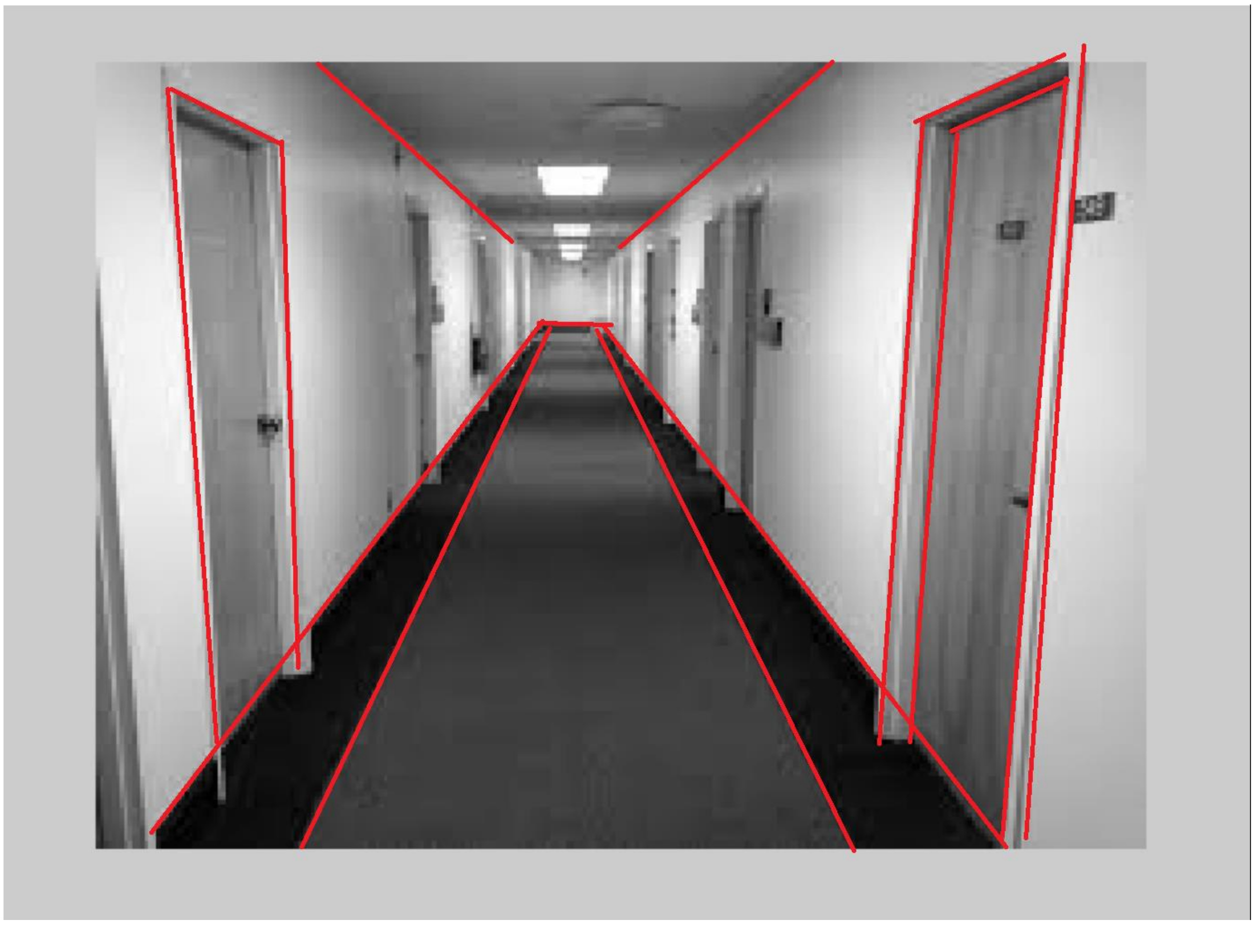


Figure 16 Established ground truth

Variance	**0**	**1**	**101**	**201**	**301**	**401**	**501**	**601**	**701**	**801**
Precision	10/14	10/14	9/13	10/15	9/13	8/13	10/18	8/13	12/25	10/30
Recall	10/16	10/16	9/16	10/16	9/16	8/16	10/16	8/16	12/16	10/16

<u>Section 6: Interpretation:</u>

The following are my observations:

- Thus we can see from the analysis that the Precision is adversely affected by noise. As the variance is increased, the Precison valu becomes lower. After variance above 500, the precision value is very low. Thus, for this image, Hough can tolerate variance in noise up to a value of approximately 500.

- The recall on the other hand is relatively stable. Its value remains between 0.5-0.7 for almost all values of variance in noise.

Section 7: Critique:

The experiment could be improved by following ways:

- Drawing plots of precision vs recall to know better how far am I from the ideal situation.
- Using imdilate and imerode for segments function.

Section 8: log

18 hours total