EE 569 3D Vision: Post-Lab Report 4

Furkan Gül, Student ID: 28795

December 4, 2020

1 Experiments in Lab

This first part of the report is an introductory step to understand scripts, and functions implemented in Lab deeply.

Before diving into the first filtering method discussed in Lab, it will be better to mention the main script that calls all the required functions benefited in Lab. It is named "lab4main", which can be found in the Appendix. Because there exist fundamentally three parts that will be addressed in Lab, the main script is divided into three sections by starting the code line with two comment characters (%%). Each section refers correspondingly to three distinct functions made for every three problems in the Lab. At the beginning of all those three functions, there are a couple of lines of code in common to make them generic. They all start with checking the input images if it is RGB or grayscale. If they are RGB, they will convert it into a grayscale format. To avoid arithmetic errors due to using image matrices of uint8 data type, which is the default option for image matrices in Matlab, those three functions will also convert data types of image matrices to double format at the beginning. Finally, right before those three functions show returned images in the figure, the data type of the returned image's matrix form must be converted back into uint8. So that the returned image can be displayed without any issues.

In the first section of the main script, Tomasi Kanade corner detection algorithm will be addressed. "lab4ktcorners" named function will be called with an input image and a threshold value for choosing corner points. It will return one image with corners signed with red-colored identifiers. This function can be found in the Appendix. Fistly, to eliminate noise in the input image, "lab2gaussfilt" named function is called. It was generated in the Lab 2. But, this function can also be seen in the Appendix. Then, x and y derivatives of the input image are computed using "imgradient" built-in function on Matlab. Then the element-wise products of derivatives are calculated at every pixel by using a 3 by 3 window. After that, the sums of the products of derivatives are computed at each pixel so that these sums can form the covariance matrix, H. The Tomasi Kanade corner response, R, is calculated by taking the minimum of eigenvalues of the covariance matrix H. Finally, this cornerness measure R for each element is thresholded with a threshold value given as an input. If R is bigger than this threshold value, then the corresponding pixel point for that R is added to the corner point list. Lastly, the three input images together with all their

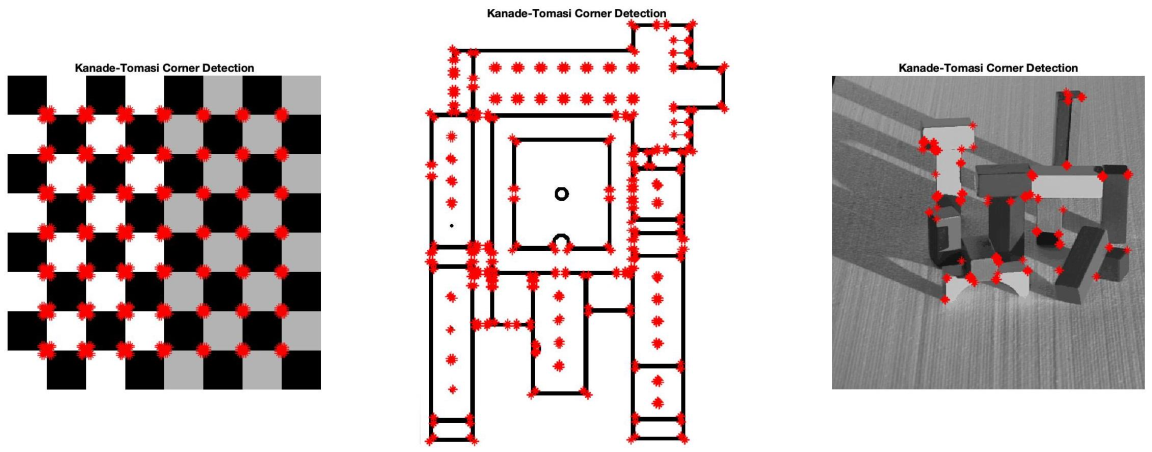


Figure 1: Corners detected with Tomasi Kanade Cornerness Measure

corner points in the lists will be displayed in the Matlab figure. This can be seen in Figure 1. To create a 20 by 20 checkerboard input image, checkerboard built-in function is used. However, it generates a binary checkerboard image. Since our algorithm expects images with pixel values of the range of [0, 255], "lab1linscale" function wrote in the Lab 1 is called to linearly scale the checkerboard image to [0, 255] pixel values. This function also can be found in the Appendix. Note that to get the same results as in the Lab 4 sheet, threshold values for each input image are optimized by hand. Those threshold values are 100000 for the checkerboard image, 200000 for the monastry image, 25000 for the blocks image.

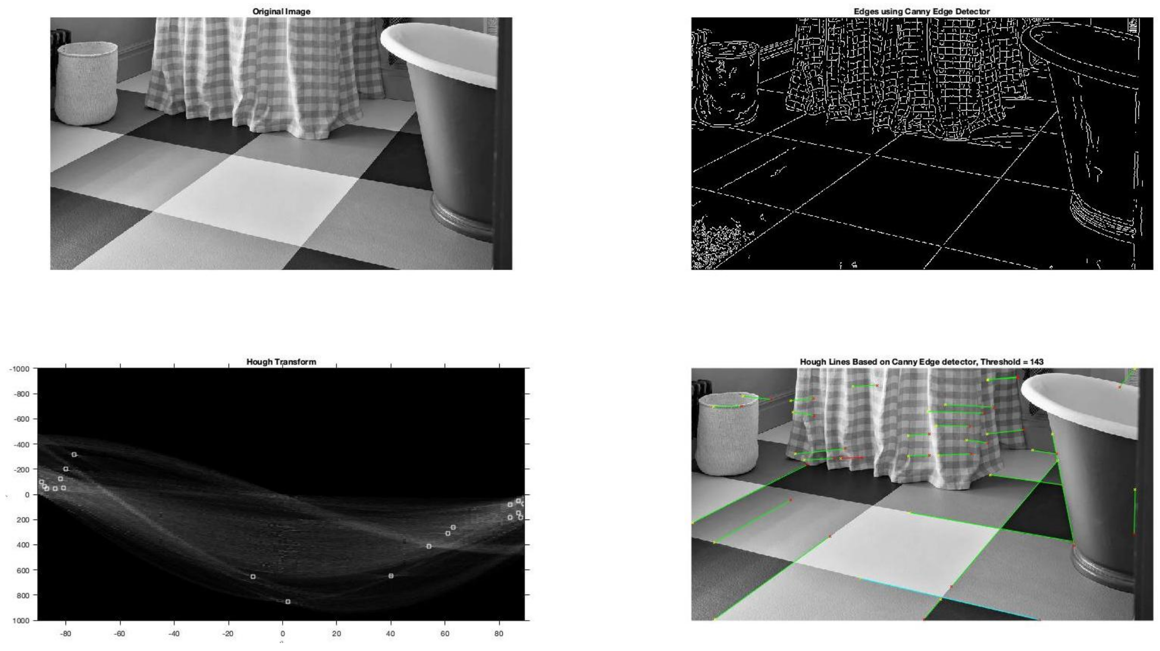


Figure 2: Steps of Detecting Lines using Hough Transformation Method

In the second section of the main script, there is a hough transformation method to detect lines in an input image. "lab4houghlines" named function will be called with an input image. It will return four images in one Matlab figure. This function can be found in the Appendix. Firstly, a canny edge detector is used to find edgels in the input image using a built-in edge function on Matlab. Then hough transform is applied to that edgels image by using another built-in function, hough. The threshold value for detecting hough lines is obtained by taking the half of the maximum number of votes in the accumulator array, H. By using houghpeaks built-in function, a maximum of 20 peaks are found using the previously calculated threshold value. Houghlines built-in function is called with those 20 peaks in Hough space, their line orientations (T), their distances between the origin and themselves (R), and also setting "FillGap" and "MinLength" parameter to 10, and 40 respectively. Houghlines function outputs hough lines detected in the input image with the parameters specified earlier. Lastly, the original input image, canny edge image, hough (parameter) space, and hough lines on the original image will be displayed in the Matlab figure. This can be seen in Figure 2.

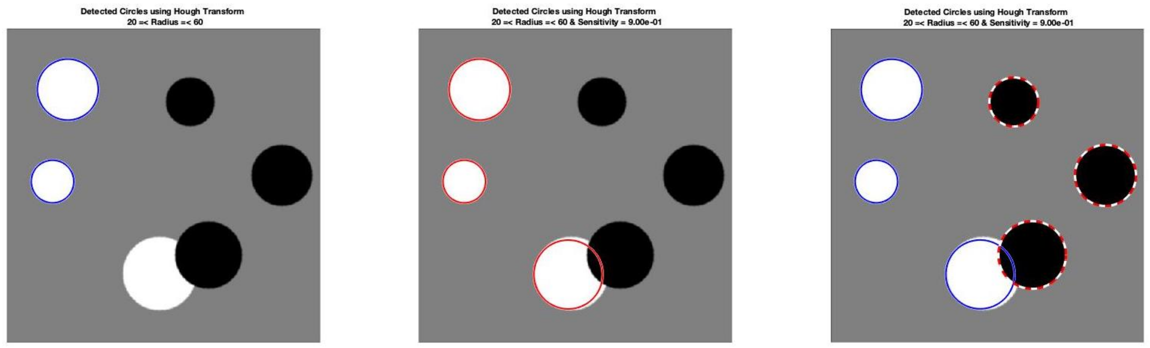


Figure 3: Bright and Dark Circles in the Image using Hough Circle Detection

In the third part of the main script, hough circle detection will be addressed to find out hough circles in a given input image. "lab4houghcircles" named function will be called with an input image and output the centers and the radius of all circles (bright and dark) in the given image. This function can be found in the Appendix. Firstly, sensitivity, minimum and maximum radius for target circles are set as 0.9, 20, and 60 respectively. Locations of bright circles are found by using the built-in imfindcircles function within the specified radius range and object polarity of "bright" option. Secondly, by keeping parameters of imfindcircles function and also adding specified sensitivity, the bright circles are detected with the corresponding sensitivity value. Thirdly; by keeping all parameters (radius range, sensitivity value) the same and just change the object polarity setting to "dark" option, the dark circles are detected with the given sensitivity value. Lastly, the bright hough circles without sensitivity in the input image, the bright hough circles with sensitivity in the input image, and both the bright and dark hough circles in the original input image will be displayed in the Matlab figure. This can be seen in Figure 3.

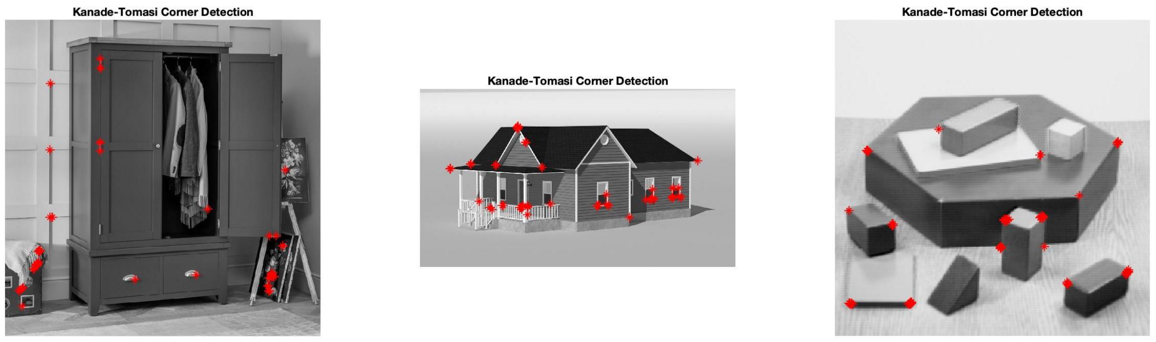

Kanade-Tomasi Corner Detection
Kanade-Tomasi Corner Detection
Kanade-Tomasi Corner Detection

Figure 4: Detecting Corners in Three Input Images using Thomasi Kanade Algorithm

2 Results & Discussions

To get a better understanding of how accurately Harris corner detector, hough line transformation, and hough circle detector will extract corners, lines, and circles features in a given input image, four different input images found online will be processed with those three methods with different hyperparameters like thresholds, sensitivity value. These input images are a wardrobe, a house, different shapes, a road, a soccer field, and circles. Wardrobe,

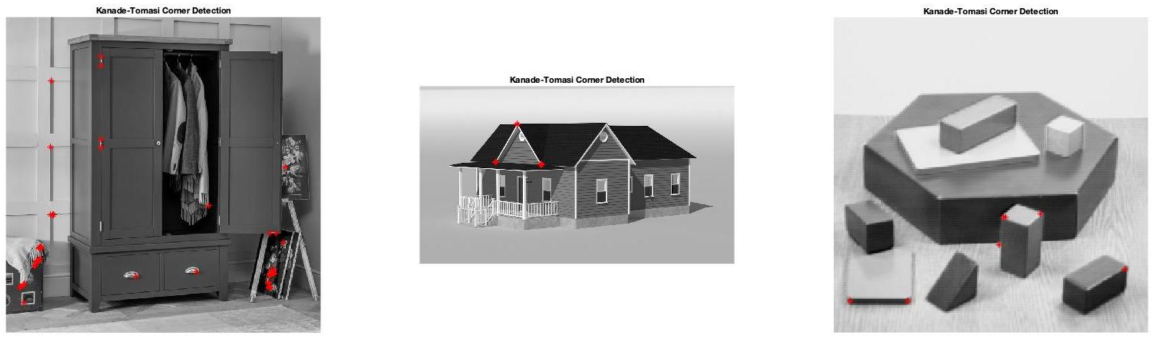

Kanade-Tomasi Corner Detection
Kanade-Tomasi Corner Detection
Kanade-Tomasi Corner Detection

Figure 5: Detecting Corners in Three Input Images using Thomasi Kanade Algorithm with High Threshold Values

house, shapes images are processed through Tomasi Kanade corner detection algorithm by calling ”lab4ktcorners” function. To detect corners in a better way, threshold values for those three images are adjusted manually to 100000 for wardrobe image, 150000 for house image, and 50000 for shapes image. The resulting corners can be seen in Figure 4. It is also noticed that when the threshold value is increased, detected corners will be less in each image and. To show this deduction, all three thresholds are multiplied by 2, the result can be seen in Figure 5. When the thresholds are reduced by half, there will be more corners detected in all three input images. It can be seen in Figure 6. The first two input images are rotated about 60 degrees in a counter clockwise by using imrotate built-in function and the third input image is scaled by 2 by using rescale built-in function. It can be seen in Figure 7 that Tomasi Kanade corner response is invariant to image rotation and intensity changes. However, it is not invariant to scale. In the third rescaled shapes image, the algorithm even does not find any corner due to scale change.

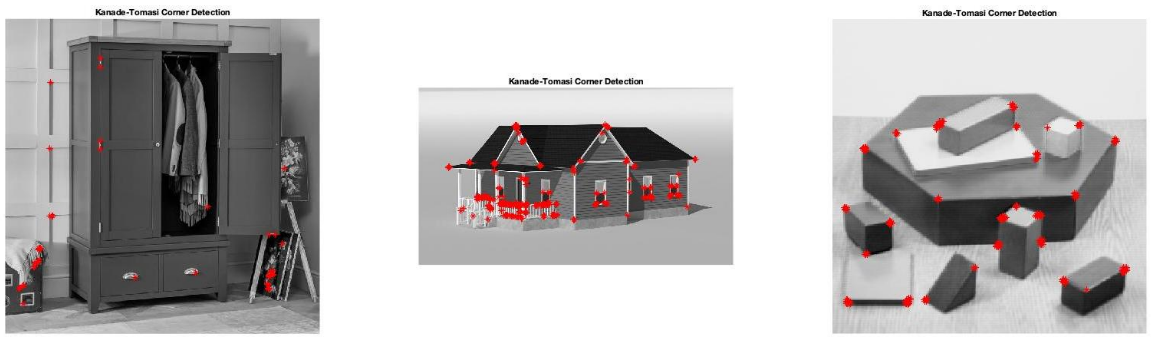

Kanade-Tomasi Corner Detection
Kanade-Tomasi Corner Detection
Kanade-Tomasi Corner Detection

Figure 6: Detecting Corners in Three Input Images using Thomasi Kanade Algorithm with Low Threshold Values

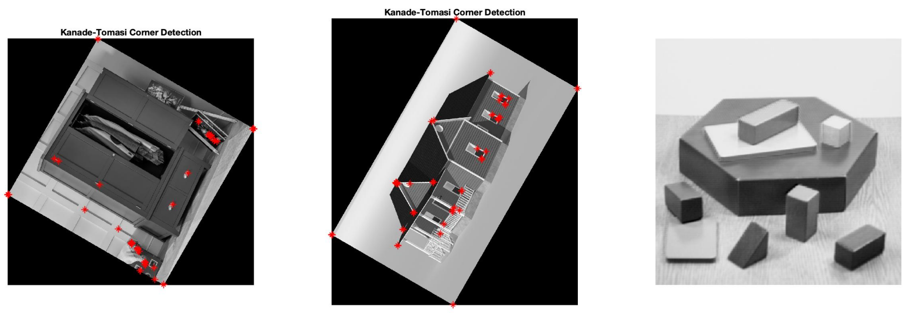

Kanade-Tomasi Corner Detection
Kanade-Tomasi Corner Detection

Figure 7: Detecting Corners in Three Input Images using Thomasi Kanade Algorithm with Rotation and Scale in Images

Hough transform is a voting scheme. Points vote for a set of parameters describing a line/circle or curve (parametric models). The more votes for a particular set, the more evidence that the corresponding curve is present in the image. Advantages of Hough transform are the following: 1- it deals with occlusion well, 2- it detects multiple instances in one shot, 3- it is robust to noise, 4- it is robust to scale and clutter. Disadvantages of Hough transform are the following: 1- computational complexity is high, 2- it is hard to set parameters of models (like curves). As the noise level in an image increases, the maximum number of votes for a line, circle (shapes) becomes fewer in the right bin of the accumulator array. As the number of noise points becomes more, the maximum number of votes for a line, circle (shapes) becomes more in the wrong bin of accumulator array. If the Radius of the circle is not known, then we would have 3D Hough Space. Accumulator array becomes A(a, b, r). Surface shape in Hough Space becomes complicated. Almost all of those advantages can be seen in the example explained below. To observe how lines are detected by using

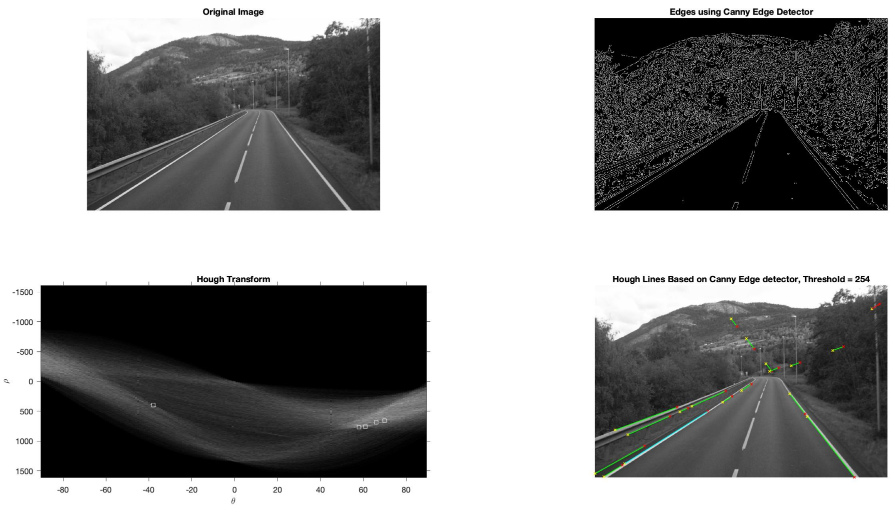


Figure 8: Hough Space and Hough Lines in Road Image

hough transformation method, two input images are used. These are a road, and a soccer field images. The result of hough spaces and hough lines for those two images can be seen in Figures 8 and 9. To get rid of outlier lines, noise must be removed from input images. That's why before starting hough transformation, the noise removal step is applied using the built-in imgaussfilt function. Since there are not many lines in those two input images, the parameter for hough peaks number is chosen as 5, not 20 as it is in the Lab. Occlusion in edge points is not affecting the performance of hough transformation algorithm as it is in the case for canny edge detector.

To see the effect of hough transform method on detecting circles, circles image is used with different radius intervals and sensitivity values. In Figure 10, the circles of radius between 20 and 60 are tried to identify with 0.9 sensitivity. Since there are fewer circles observed, it is decided to increase the maximum radius value to 160 keeping the minimum radius and sensitivity the same. As can be seen in Figure 11, it gives the same result as in Figure 10. So, it is time to just reduce the minimum radius to 10 and keep the maximum radius as 60 and sensitivity as 0.9. The result can be seen in Figure 12. There are now many more hough circles detected in the input image. So, the conclusion is that circles in the input image have a smaller radius. To see how sensitivity affects detecting circles, by keeping all parameters the same and just change the sensitivity from 0.9 to 0.7. The resulting hough circles can be seen in Figure 13. As it is seen in Figure 13, when sensitivity decreases, the number of hough circles becomes less and less. So, the bottom-up is that choose your radius interval according to the circles in your input image and do not drop the sensitivity value too much.

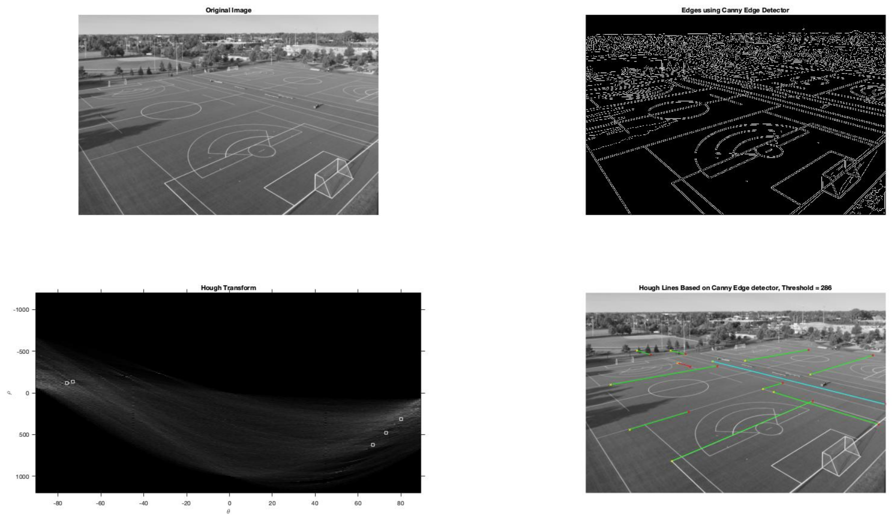

Original Image
Edges using Canny Edge Detector
Hough Transform
Hough Lines Based on Canny Edge detector, Threshold = 286
-1000
-500
0
500
1000
ρ
-80
-60
-40
-20
0
20
40
60
80
θ

Figure 9: Hough Space and Hough Lines in Road Image

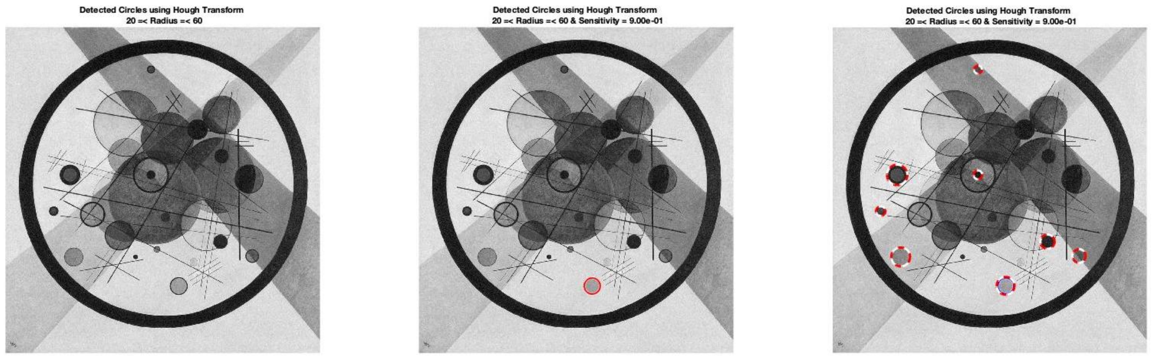

Detected Circles using Hough Transform
20 =< Radius =< 60
Detected Circles using Hough Transform
20 =< Radius =< 60 & Sensitivity = 9.00e-01
Detected Circles using Hough Transform
20 =< Radius =< 60 & Sensitivity = 9.00e-01

Figure 10: Hough Circles in Circles Image

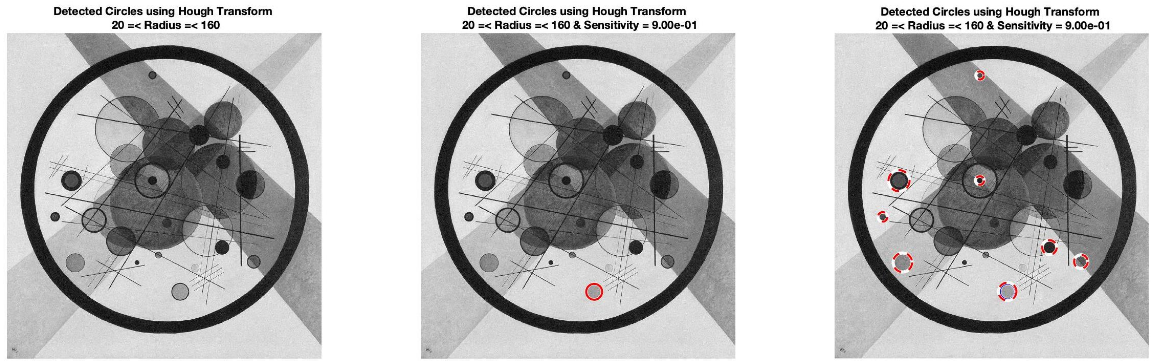

Detected Circles using Hough Transform
20 =< Radius =< 160
Detected Circles using Hough Transform
20 =< Radius =< 160 & Sensitivity = 9.00e-01
Detected Circles using Hough Transform
20 =< Radius =< 160 & Sensitivity = 9.00e-01

Figure 11: Hough Circles in Circles Image with Increasing Maximum Radius

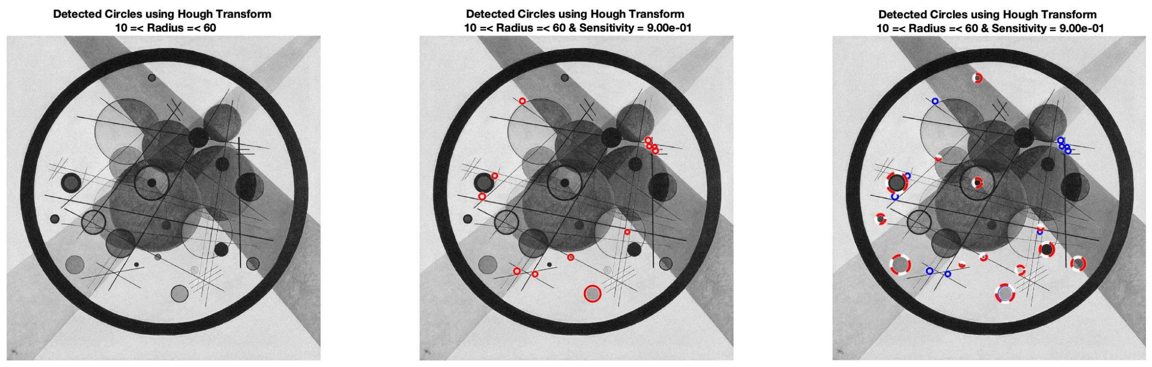

Detected Circles using Hough Transform
10 =< Radius =< 60
Detected Circles using Hough Transform
10 =< Radius =< 60 & Sensitivity = 9.00e-01
Detected Circles using Hough Transform
10 =< Radius =< 60 & Sensitivity = 9.00e-01

Figure 12: Hough Circles in Circles Image

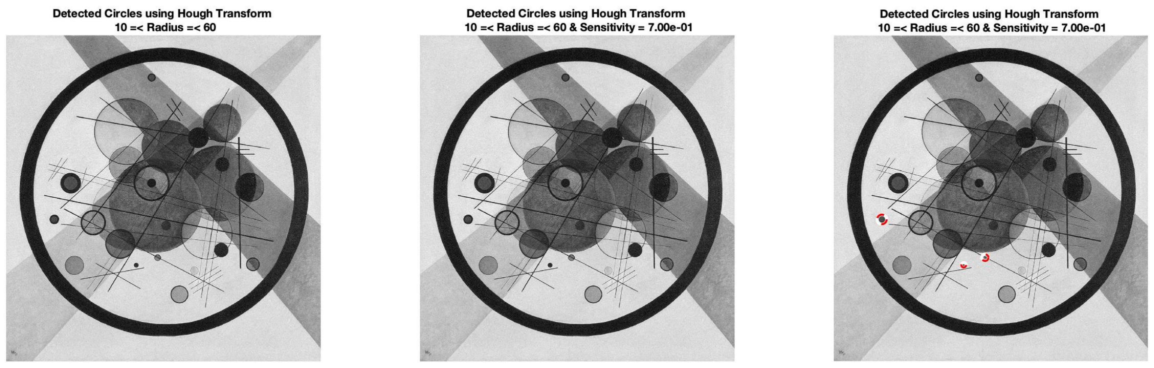

Detected Circles using Hough Transform
10 =< Radius =< 60
Detected Circles using Hough Transform
10 =< Radius =< 60 & Sensitivity = 7.00e-01
Detected Circles using Hough Transform
10 =< Radius =< 60 & Sensitivity = 7.00e-01

Figure 13: Hough Circles in Circles Image with Increasing Maximum Radius

3 Appendix

All the below codes and their corresponding outputs were obtained as latex output format without any manual intervention by using the "Publish" Method in Matlab.

lab4main Script

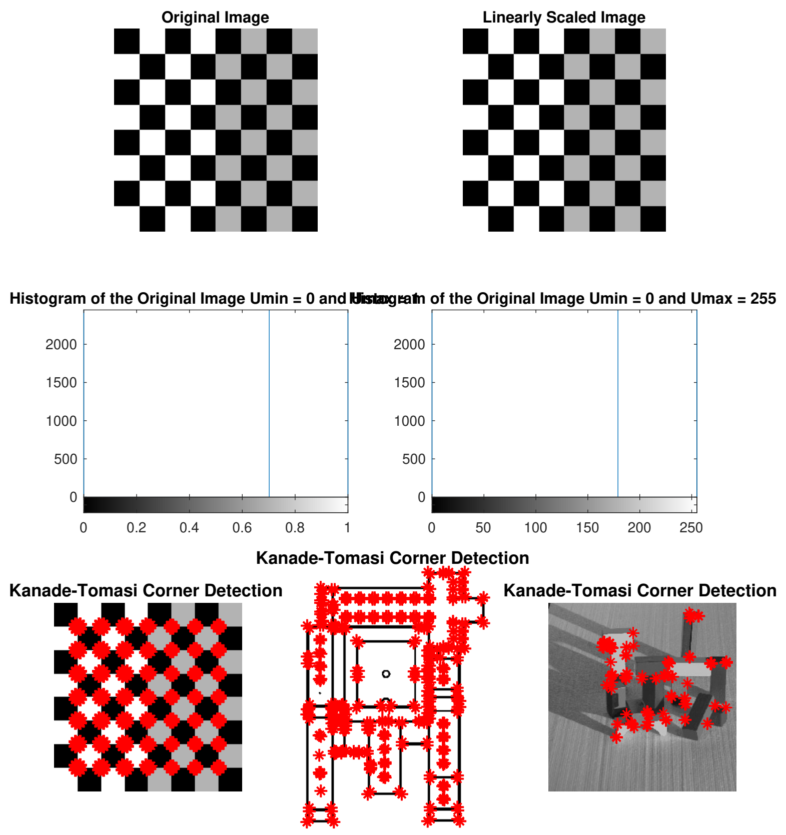
Contents of lab4main Script

Part 1: Corner Detection

```
clc;clear;close;
tic
% Let's create checker board image
img0 = checkerboard(20);
% checkerboard image is binary, it must be linearly scaled to [0, 255]
[img0] = lab1linscale(img0);
% Let's get Monastry and blocks image
img1 = imread("Monastry.bmp");
img2 = imread("blocks.png");
T0 = 100000;
[corners_filtered0] = lab4ktcorners(img0, T0);
T1 = 200000;
[corners_filtered1] = lab4ktcorners(img1, T1);
T2 = 25000;
[corners_filtered2] = lab4ktcorners(img2, T2);
toc

% Plot corners with the original input images
figure()
subplot(1,3,1)
imshow(img0);
hold on;
plot(corners_filtered0(:,2), corners_filtered0(:,1),"r*", "MarkerSize", 7, "Linewidth",
title("Kanade-Tomasi Corner Detection")

subplot(1,3,2)
imshow(img1);
hold on;
```

```
plot(corners_filtered1(:,2), corners_filtered1(:,1),"r*", "MarkerSize", 7, "Linewidth",
title("Kanade-Tomasi Corner Detection")

subplot(1,3,3)
imshow(rgb2gray(img2));
hold on;
plot(corners_filtered2(:,2), corners_filtered2(:,1),"r*", "MarkerSize", 7, "Linewidth",
title("Kanade-Tomasi Corner Detection")
```

```
Elapsed time is 5.288082 seconds.
```



lab4ktcorners Function

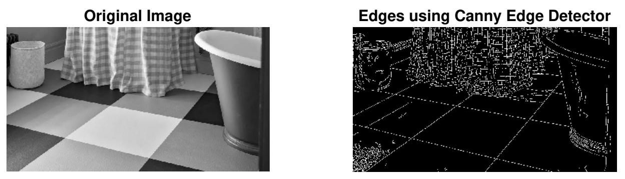
```
function [corners] = lab4ktcorners(img, T)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end
```

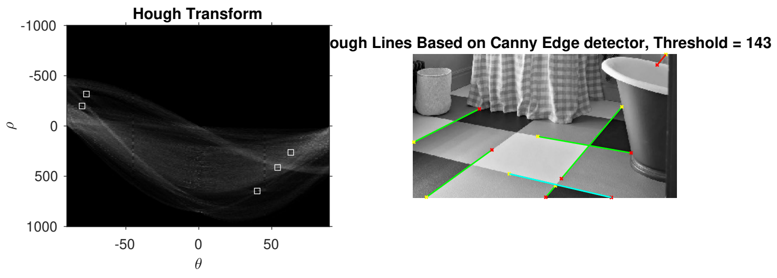
```
% Transforming to double
img = double(img); % In order to have proper matrix operations, convert
% datatype from uint8 to double

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

% Smooting
%Simg =  lab2gaussfilt(img);
Simg = imgaussfilt(img, 1.1);

% Compute gradient of Image
[Gx, Gy] = imgradientxy(Simg);

% Go over center pixels. We have total 2k+1 parameters in every row & column of
% filter. Since we have a valid boundary, at the start, the center of the image
% coincides the filter at (k+1, k+1) position. The filter (window) will slide
% one by one until (h-k,w-k) position in the image.

% Finding H for each Pixel using windows
for i = k+1:h-k
    for j = k+1:w-k
        % Find window sized gradient in x and y directions
        Ix = Gx(i-k:i+k, j-k:j+k);
        Iy = Gy(i-k:i+k, j-k:j+k);

        % Find elements of H matrix
        h11 = sum(sum(Ix.^2));
        h12 = sum(sum(Ix.*Iy));
        h21 = h12;
        h22 = sum(sum(Iy.^2));

        % Form H matrix
        H(1,1) = h11;
        H(1,2) = h12;
        H(2,1) = h21;
        H(2,2) = h22;

        % Finding eigenvalues and thresholding to get corner points
        [~, eigval] =eig(H);

        if min(eigval(1,1), eigval(2,2))>T
            corners = [corners;i j];
        end
    end
end

% Change the datatype of the images from double to uint8
% so that it can be displayed without any problem.
img = uint8(img);

% Plot corners with the original input images
% figure()
%
% imshow(img);
% hold on;
% plot(corners(:,2), corners(:,1),"r*", "MarkerSize", 7, "Linewidth", 1);
% title("Kanade-Tomasi Corner Detection")
end
```

Part 2: Lines Detection

```
clc;clear;close;
tic
% Let's get checker image
img = imread("checker.jpg");
[H, theta, rho] = lab4houghlines(img);
toc
```

```
Elapsed time is 0.344350 seconds.
```





lab4houghlines Function

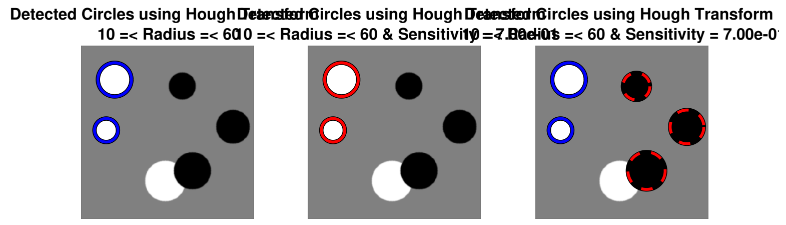
```
function [H, T, R] = lab4houghlines(img)
% Chech the original image if it is RGB or monocolor
[~, ~, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end
% If the input image has some noise
%img = imgaussfilt(img, 1.1);

% Edge Detection by Canny
edge_detector = "Canny";
img_edges = edge(img, edge_detector);

% Hough Transform
[H, T, R]=hough(img_edges);

% Display Part for Original Image, Canny Edge, and Hough Transform
figure("Name", "Hough Transform", "NumberTitle", "off");
subplot(2,2,1)
imshow(img)
title("Original Image")

subplot(2,2,2)
imshow(img_edges)
title(sprintf("Edges using %s Edge Detector", edge_detector))

subplot(2,2,3)
imshow(H, [], "XData", T, "YData", R, "InitialMagnification", "fit");

title("Hough Transform")
xlabel('\theta');
ylabel("\rho");
axis on, axis normal, hold on;


% Hough Peaks
Thresh = ceil(0.5*max(H(:)));

P = houghpeaks(H, 5, "threshold", Thresh);
x = T(P(:,2));
y = R(P(:,1));
```

```
plot(x,y,"s","color","white");

% Hough Lines with given parameters in lab sheet
lines = houghlines(img_edges, T,R,P,"FillGap", 10, "MinLength", 40);

% Plotting Hough Lines
subplot(2,2,4);
imshow(img), title(sprintf('Hough Lines Based on %s Edge detector, Threshold = %d', edge

max_len = 40;
min_len = 2000;

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

       % Plot beginnings and ends of lines
       plot(xy(1,1),xy(1,2),'x','LineWidth',1,'Color','yellow','Markersize',3);
       plot(xy(2,1),xy(2,2),'x','LineWidth',1,'Color','red','Markersize',3);

       % Determine the endpoints of the longest line segment
       len = norm(lines(k).point1 - lines(k).point2);
       if ( len > max_len)
          max_len = len;
          xy_long = xy;
       end

       if (len < min_len)
           min_len = 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');
end
```

Part 3: Circle Detection

```
clc;clear;close;
tic
% Let's get circlesBrightDark image
img = imread("circlesBrightDark.png");
[centersBright, radiiBright,centersDark, radiiDark] = lab4houghcircles(img);
toc
```

```
Elapsed time is 0.495703 seconds.
```


Detected Circles using Hough Transform
10 =< Radius =< 60
Detected Circles using Hough Transform
10 =< Radius =< 60 & Sensitivity = 7.00e-01
Detected Circles using Hough Transform
10 =< Radius =< 60 & Sensitivity = 7.00e-01

lab4houghcircles Function

```
function [centersBright, radiiBright,centersDark, radiiDark] = lab4houghcircles(I)
% Chech the original image if it is RGB or monocolor
[~,~,ch] = size(I);
    if (ch==3)
         I = rgb2gray(I);
    end

% Identify min and max radius ans sensitivity parameters
Rmin = 10;
Rmax = 60;
Sens = 0.7;

% Find circles in a given image using parameters above

% Find all the bright circles in the image within the radius range
[centersBright1, radiiBright1] = imfindcircles(I,[Rmin Rmax],'ObjectPolarity','bright');

% Find all the bright circles in the image within the radius range and
% sensitivity
[centersBright, radiiBright] = imfindcircles(I,[Rmin Rmax],'ObjectPolarity','bright','Se
% Find all the dark circles in the image within the radius range and
% sensitivity
[centersDark, radiiDark] = imfindcircles(I,[Rmin Rmax],'ObjectPolarity','dark','Sensitiv

% Plot circles
figure()
subplot(1,3,1)
imshow(I),title(sprintf('Detected Circles using Hough Transform \n %d =< Radius =< %d',
viscircles(centersBright1, radiiBright1,'EdgeColor','b');

subplot(1,3,2)
imshow(I),title(sprintf('Detected Circles using Hough Transform \n %d =< Radius =< %d &
viscircles(centersBright, radiiBright,'EdgeColor','r');
```

```
subplot(1,3,3)
imshow(I),title(sprintf('Detected Circles using Hough Transform \n %d =< Radius =< %d &
viscircles(centersBright, radiiBright,'Color','b');
viscircles(centersDark, radiiDark,'LineStyle','--');
end
```

Examples4Lab4 Script

Contents of Examples4Lab4 Script

Part 1: Corner Detection

```
clc;clear;close;
tic
img0 = imread("wardrobe.jpg");
img0 = imrotate(img0,60);
img1 = imread("house.jpg");
img1 = imrotate(img1,60);
img2 = imread("shapes.gif");
img2 = imresize(img2,2);
T0 = 100000;
%T0 = 2*T0;
%T0 = T0/2;
[corners_filtered0] = lab4ktcorners(img0, T0);
T1 = 150000;
%T1=2*T1;
%T1=T1/2;
[corners_filtered1] = lab4ktcorners(img1, T1);
T2 = 50000;
%T2 = 2*T2;
%T2 = T2/2;
[corners_filtered2] = lab4ktcorners(img2, T2);
toc

% Plot corners with the original input images
figure()
subplot(1,3,1)
imshow(rgb2gray(img0));
```

```
hold on;
plot(corners_filtered0(:,2), corners_filtered0(:,1),"r*", "MarkerSize", 7, "Linewidth",
title("Kanade-Tomasi Corner Detection")

subplot(1,3,2)
imshow(rgb2gray(img1));
hold on;
plot(corners_filtered1(:,2), corners_filtered1(:,1),"r*", "MarkerSize", 7, "Linewidth",
title("Kanade-Tomasi Corner Detection")

subplot(1,3,3)
imshow(img2);
hold on;
plot(corners_filtered2(:,2), corners_filtered2(:,1),"r*", "MarkerSize", 7, "Linewidth",
title("Kanade-Tomasi Corner Detection")
```

```
Elapsed time is 18.542514 seconds.

Index in position 2 exceeds array bounds.

Error in Examples4Lab4 (line 42)
plot(corners_filtered2(:,2), corners_filtered2(:,1),"r*", "MarkerSize", 7, "Linewidth",
```

Part 2: Lines Detection

```
clc;clear;close;
tic
% Let's get checker image
img = imread("road.jpg");
[H, theta, rho] = lab4houghlines(img);
img1 = imread("soccerField.jpeg");
[H, theta, rho] = lab4houghlines(img1);
toc
```

Part 3: Circle Detection

```
clc;clear;close;
tic
% Let's get circlesBrightDark image
img = imread("circles.jpg");
[centersBright, radiiBright,centersDark, radiiDark] = lab4houghcircles(img);
toc
```