EE 569 3D Vision: Post-Lab Report 5

Furkan Gül, Student ID: 28795

December 28, 2020

1 Experiments in Lab

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

This lab is intended to find correspondence points in an input image before calibrating the camera by using the projection matrix. Since the easiest feature for correspondence point is a corner, it would make sense to find corners in a given input image. There are mainly two methods for detecting corners in an image. These two methods are literally Harris corner detection and the intersection point of two lines. In Lab, these two methods will be implemented to extract corners in an image. There is only one script for exploiting corners in an image by means of two different methods. The main script that calls the required corner detection codes benefited in Lab is named "lab5calibprep", which can be found in the Appendix. In this main script, the given input calibration image for Lab is loaded into the workspace of Matlab. After transforming the given RGB image to the monochrome version, the edge detector type is specified as Canny and implemented on the gray-scale input image. Then, the lines in the edge image will be extracted by utilizing 'hough', 'houghpeaks', and 'houghlines' builtin Matlab functions with suitable parameters. Parameters: threshold for hough peak detection, "FillGap", and "MinLength" are the half of the maximum value of Hough accumulator array, 10, and 35 respectively. The overall lines, the starting point of the lines, and the end point of the lines will be drawn as green color, yellow color cross, and red color cross on the gray-scale version of the input image respectively. When the breakpoint is set to the line of 63, the resulting calibration object image with hough lines can be seen in Figure 1.

Then, two intersecting lines are selected manually from the plot in Figure 1. The selected first line has a [39, 130] coordinate of the beginning point and a [170, 180] coordinate of the endpoint. The second selected line has a [80, 112] coordinate of the beginning point and a [106, 246] coordinate of the endpoint. Then, the corresponding ρ (rho) and θ (theta) values from the output of 'houghlines' function will be extracted automatically using the built-in ismember function over the struct formatted "line" parameter. The equations of these two lines will be determined by using the line equation and then the lines will be plotted with magenta color on the same figure. Then, these two line equations will be solved to determine the intersection point with sub-pixel accuracy and then plot that point on the same figure

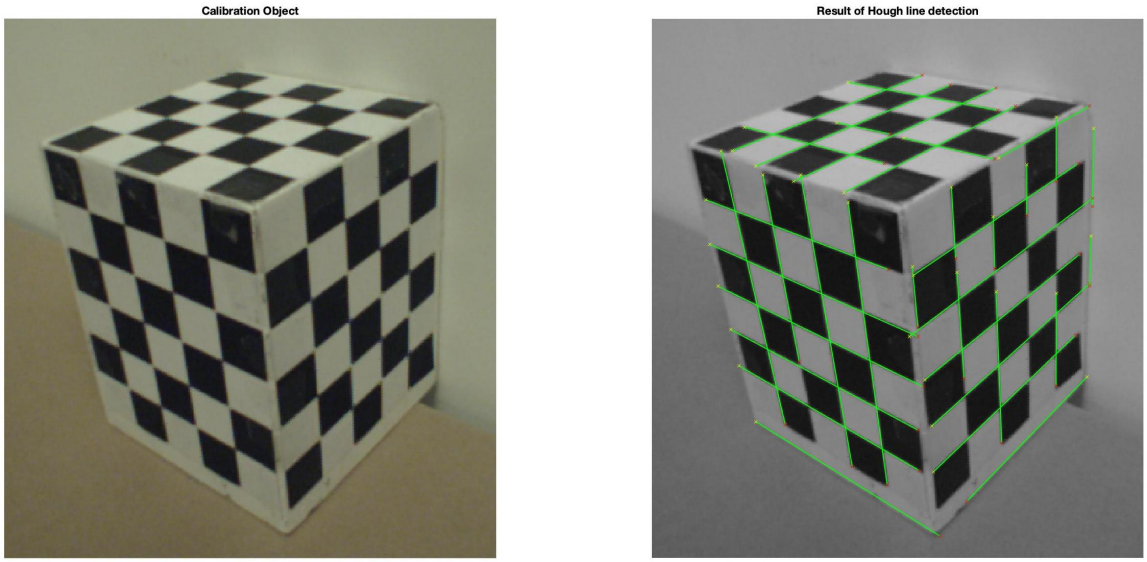


Figure 1: Hough line detection with "FillGap" parameter of 10 and "MinLength" parameter of 35

too. Harris corners will be detected by using the gray-scale version of the input calibration object image and plot them on the same figure with blue circles. Finally, by using the built-in line function, the distance between two corner points (closest two Harris corner points will be considered not one corner point) obtained with two different methods will be calculated and showed in the same figure. The resulting calibration object image with selected two lines and Harris corners together with the zoomed area of the intersection of lines can be seen in Figure 2. Note that, due to the resolution issue, euclidean distances may not be much visible. They are actually 1.68 and 3.37.

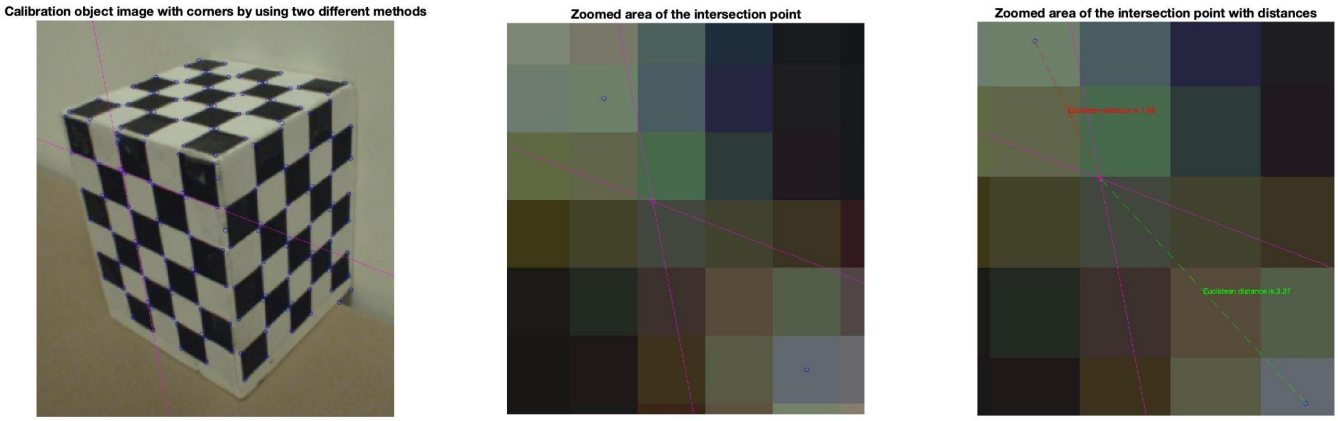


Figure 2: Corners detected by the intersection point of two lines and Harris corner detection algorithm

2 Results & Discussions

There are a few parameters to optimize the corner extraction process. These are the selection of the calibration object, type of edge detection algorithm, and parameters for houghline function: "FillGap" and "MinLength". The calibration object is created by taking a picture of a wire board over a carpet since this method will create a higher contrast image to be able to extract more corner points in the image. In the experiments, there will be only one calibration object to examine since it is really hard to find such an object with atleast 8 corner points and define them. For edge detection, Canny is selected as default since it will be very time consuming to detect all 8 couple of lines coordinates and process from the scratch. When houghline function detects two line segments associated with the same Hough transform bin that are separated by less than 'FillGap' distance, houghline algorithm merges them into a single line segment. Merged line segments that are shorter than 'MinLength' value are discarded.

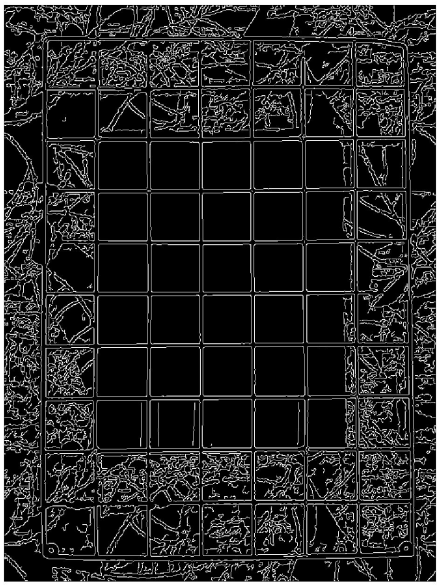


Figure 3: Canny Edgels in the Calibration Object

For the experiments in this section, there is a different Matlab script called "lab5calibprep2" which can be found in the Appendix. It will take the calibration object image and output 8 correspondence corner points for camera calibration. Since the overall structure of this script is very similar to "lab5calibprep" mentioned in the previous section, the differences will be emphasized here. The resolution of the calibration image is very high dimensional due to the high-quality camera on a mobile phone and its rotation is not correctly fit the Matlab figure. There is a preprocessing step for the calibration object which uses imresize and imrotate built-in functions. For the first experiment, Canny edge detector is selected. Canny edge image for the calibration object can be seen in Figure 3. "FillGap" and "MinLength" parameters are optimized manually to 80 and 30 respectively which takes a lot of

time. The resulting houghlines with previously mentioned parameters can be observed in Figure 4. Eight intersecting line couples are selected manually by specifying their beginning and endpoint coordinates. To solve two line equations 8 times and plot the houghline intersection points 8 times, there is a slight modification in the corresponding part of the script which uses for loop. Finally, there is a section in the script for calculating and displaying the distance between each two corner points obtained with two different methods. In this section, since a built-in line function will be used, a variable named line is removed (by using clear command) before calling line function to avoid any error. Then, the closest corner point by Harris method to the intersection of two lines together with this intersection will be manually localized for every 8 correspondences. Then, the euclidean distance between these two points will be calculated by using norm function. Line function with appropriate text on each line will be plotted on the same figure to show all distance values on the same figure. Each intersection point with the closest Harris corner point is cropped from the main figure. Since there occurs some resolution dropping when subplotting is used to combine all eight cropped images, all these eight cases can be seen in Figures 6, 7, 8, 9, 10, 11, 12, and 13 in detail.

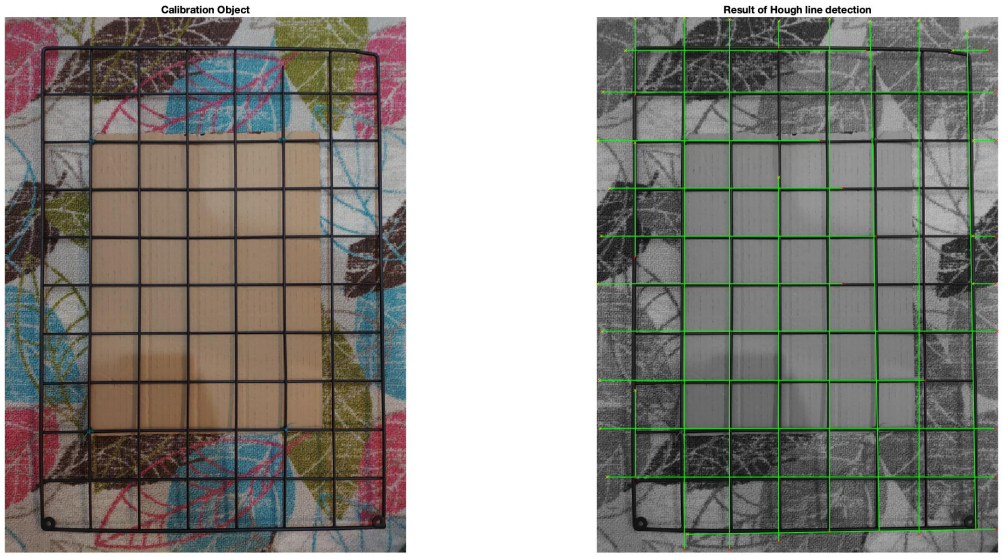


Figure 4: Calibration Object Image & Hough line detection with "FillGap" parameter of 80 and "MinLength" parameter of 30

In Figure 7, the second intersection point obtained by two-line intersection method is at the same coordinate as the corner point detected by the Harris method. As it can be observed in Figures 6, 9, 10, 12, and 13; respectively, the first, the fourth, the fifth, the seventh, and the eight intersection points are not better than corresponding Harris corners in terms of localizing the edges. In Figure 8, both Harris and the intersection line method almost perform the same in localization of the edge. In Figure 11, the intersecting line method gives a better edge location. Thus, it can be said that for this calibration object,

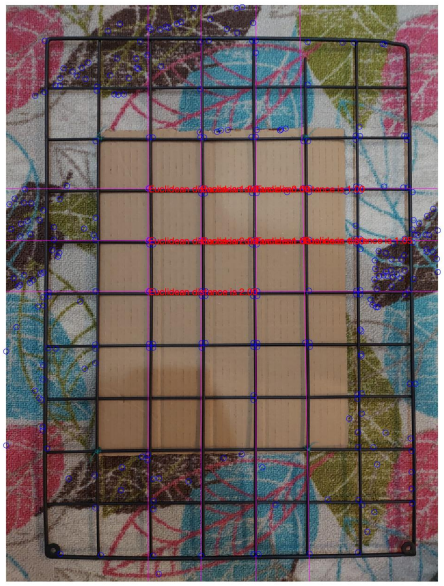


Figure 5: Corners detected by the intersection point of two lines and Harris corner detection algorithm in Calibration Object

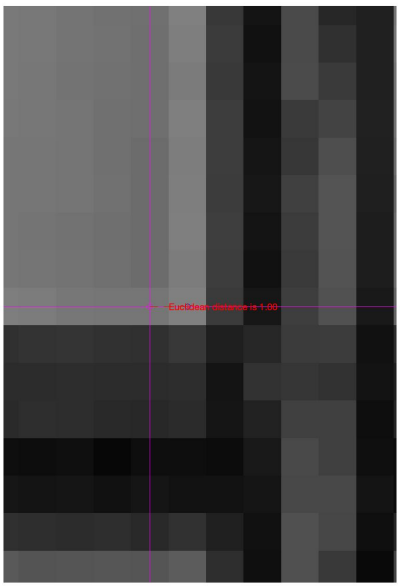


Figure 6: Zoomed area of the first intersection point and the closest Harris corner in Calibration Object

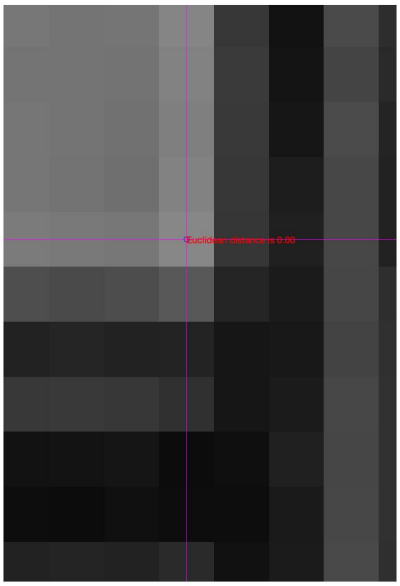


Figure 7: Zoomed area of the second intersection point and the closest Harris corner in Calibration Object

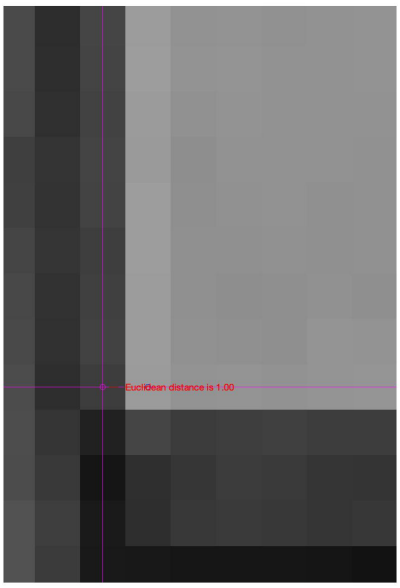


Figure 8: Zoomed area of the third intersection point and the closest Harris corner in Calibration Object

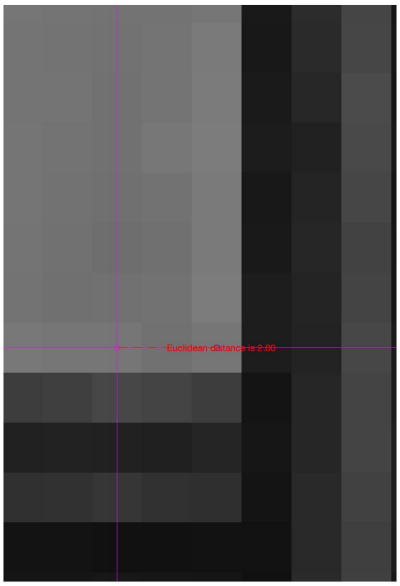


Figure 9: Zoomed area of the fourth intersection point and the closest Harris corner in Calibration Object

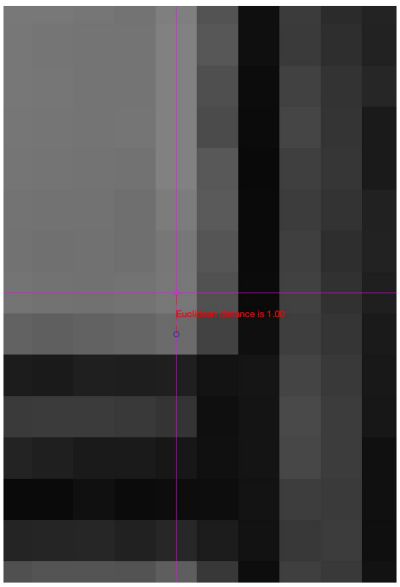


Figure 10: Zoomed area of the fifth intersection point and the closest Harris corner in Calibration Object

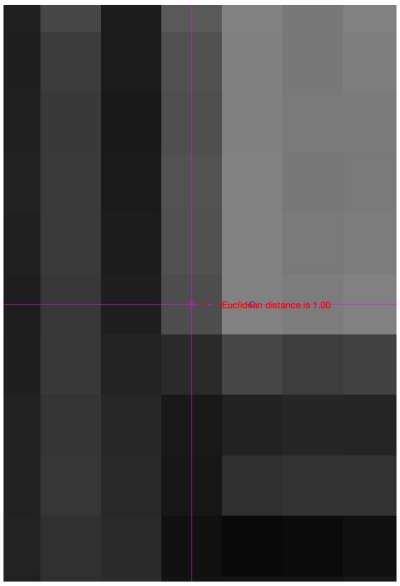


Figure 11: Zoomed area of the sixth intersection point and the closest Harris corner in Calibration Object

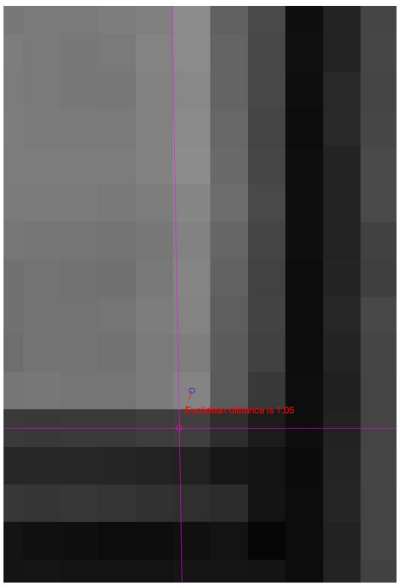


Figure 12: Zoomed area of the seventh intersection point and the closest Harris corner in Calibration Object

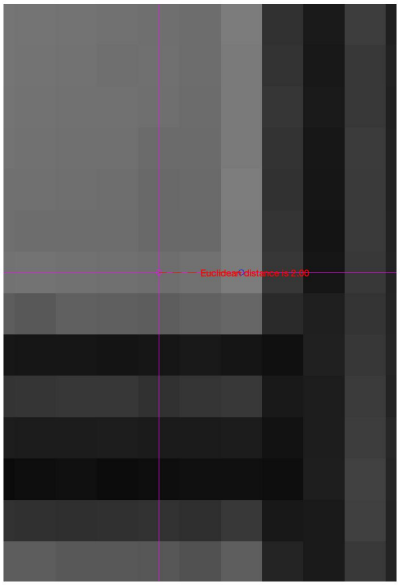


Figure 13: Zoomed area of the eighth intersection point and the closest Harris corner in Calibration Object

Harris corner method performs better than the intersection two lines method. However, there are cases where the opposite is true depending on the selection of the calibration object. To utilize both methods, I will personally combine both methods like an ensemble learning method in Machine Learning area. I will give weights between 0 and 1 to each method depending on the selection of the calibration object and engineering insight about which method would be relatively better. Then, I will linearly combine the corner locations of these two methods with the weights to obtain a more powerful corner extraction method which benefits from both methods.

If the edge detector is changed to Sobel, there will be more or less houghlines with different lengths compared to Canny edge selected result. However, if the same houghline segments can be detected as it is in the case for Canny (by tuning "MinLength" and "FillGap" parameter), then the result will not be changed if the line coordinates will be correctly chosen by hand. Generally, Sobel is giving fewer houghlines compared to Canny edge detector. That's why it would be wise to select edge detector as Canny.

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.

Contents of lab5calibprep

- Start
- HOUGH TRANSFORM - Extract Lines
- PLOT HOUGHLINES
- SELECT TWO INTERSECTING LINES MANUALLY
- PLOT INTERSECTING LINES
- Solving the 2 line equations to find intersection point (corner)
- HARRIS CORNERS
- PLOTTING CORNERS FOR COMPARISON
- Calculate and display the distance between two corner points that
- For better visualization

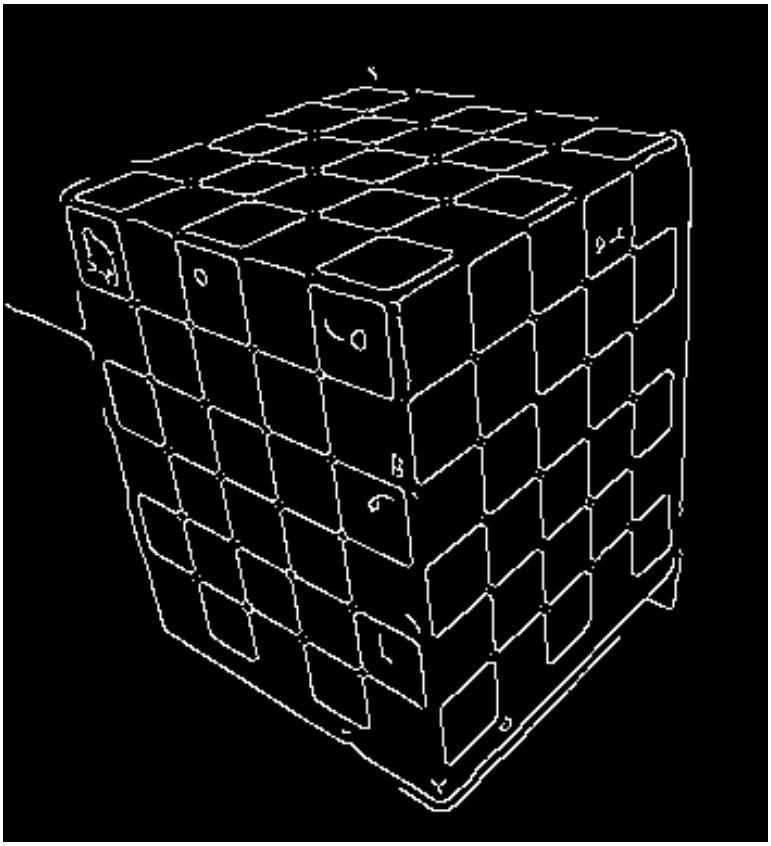
Start

```
clear; close all; clc;

% Read input calibration image
img = imread('calibrationObject.png');

% Check if it is RGB or not. Then, if it is RGB, transform it to
% monochrome image.
[row, col, ch]=size(img);
if ch==3
    img_gray = rgb2gray(img);
end

% Apply canny edge detector
img_edge = edge(img_gray,'canny');
imshow(img_edge);
```



HOUGH TRANSFORM - Extract Lines

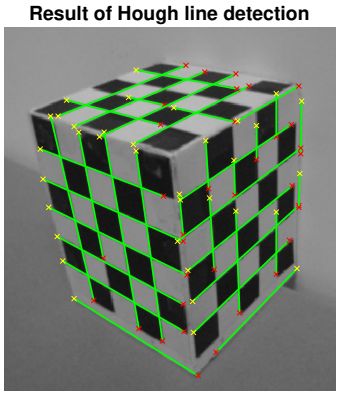
Hough Transform

```
[H, theta, rho]=hough(img_edge);
% Hough Peaks
Thresh = ceil(0.5*max(H(:)));
P = houghpeaks(H, 60, "threshold", Thresh);
% Hough Lines with given parameters in lab sheet
line = houghlines(img_edge, theta,rho,P,"FillGap", 10, "MinLength", 35);
```

PLOT HOUGHLINES

```
figure
subplot(1,2,1), imshow(img), title ("Calibration Object")
subplot(1,2,2), imshow(img_gray), title ("Result of Hough line detection")
hold on
for k = 1:length(line)
    xy = [line(k).point1; line(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','green');
```

```
    plot(xy(1,1),xy(1,2),'x','MarkerSize',4,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','MarkerSize',4,'Color','red');
    len = norm(line(k).point1 - line(k).point2);
end
hold off
```



SELECT TWO INTERSECTING LINES MANUALLY

```
Lines_B = [39 130;80 112]; % Beginning points
Lines_E = [170 180;106 246]; % End points

% Extract corresponding theta (T) and rho (R) values from the output of 'houghlines' fun

Thetas =[];
Rhos =[];
for p = 1:length(Lines_B)
    for k=1:length(line)
        if ismember(line(k).point1, Lines_B(p,:), "rows") && ismember(line(k).point2, Li
            Thetas = [Thetas; line(k).theta];
            Rhos = [Rhos; line(k).rho];
        end
    end
end
```

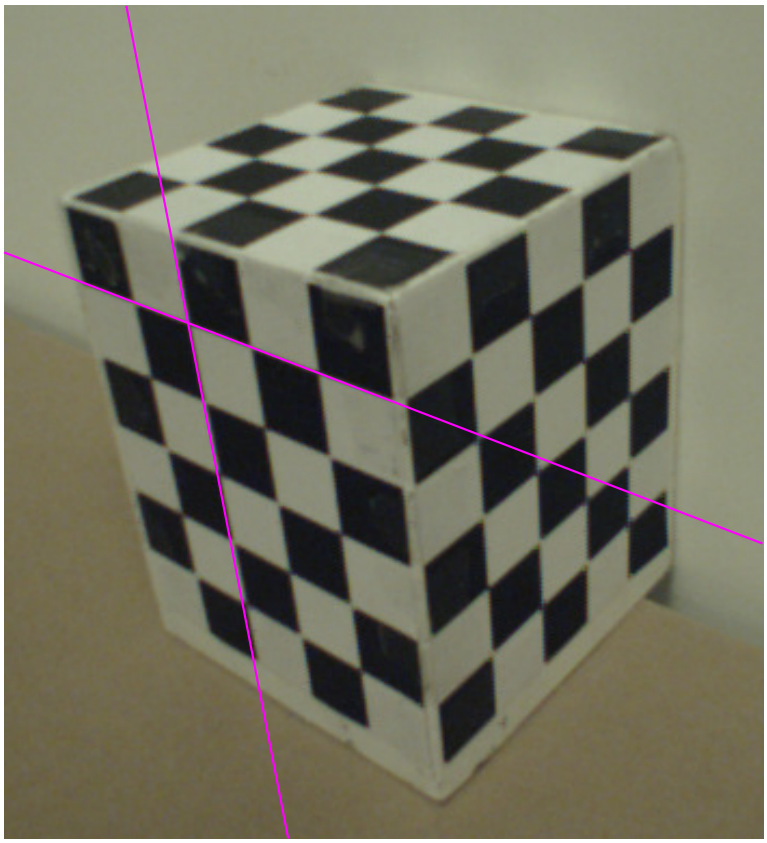
PLOT INTERSECTING LINES

```
x_v = 0:size(img,1);
x_h = 0:size(img,2);

figure
imshow(img)
hold on
```

```
for i=1:length(Thetas)
    xPoint = 0:col;
    yPoint=(Rhos(i) -xPoint*cosd(Thetas(i)))/sind(Thetas(i));
    plot(xPoint,yPoint, "color", "magenta")

end
```



Solving the 2 line equations to find intersection point (corner)

```
b = Rhos;
A = [cosd(Thetas(1)), sind(Thetas(1));cosd(Thetas(2)), sind(Thetas(2))];
C = A\b;
```

HARRIS CORNERS

```
corners = corner(img_gray);
```

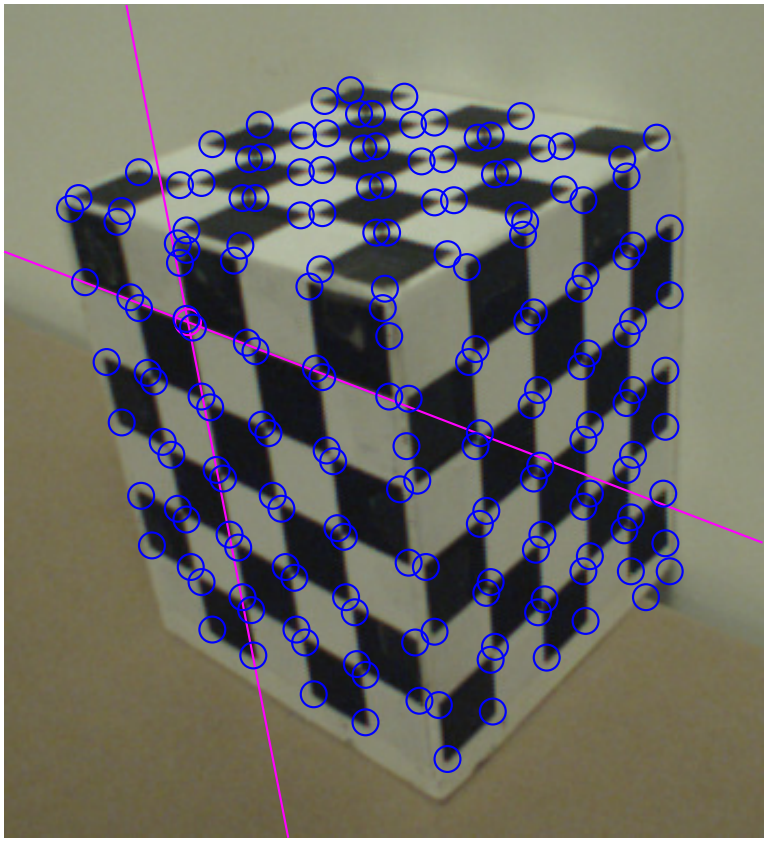
PLOTTING CORNERS FOR COMPARISON

```
figure
imshow(img)
hold on

% plot houghlines intersection point as magenta circle
plot(C(1), C(2),"mo" )
hold on

for i=1:length(Thetas)
    xPoint = 0:col;
    yPoint=(Rhos(i) -xPoint*cosd(Thetas(i)))/sind(Thetas(i));
    plot(xPoint,yPoint, "color", "magenta")

end
% plot harris corners as red circles
plot(corners(:,1),corners(:,2),'bo');
```



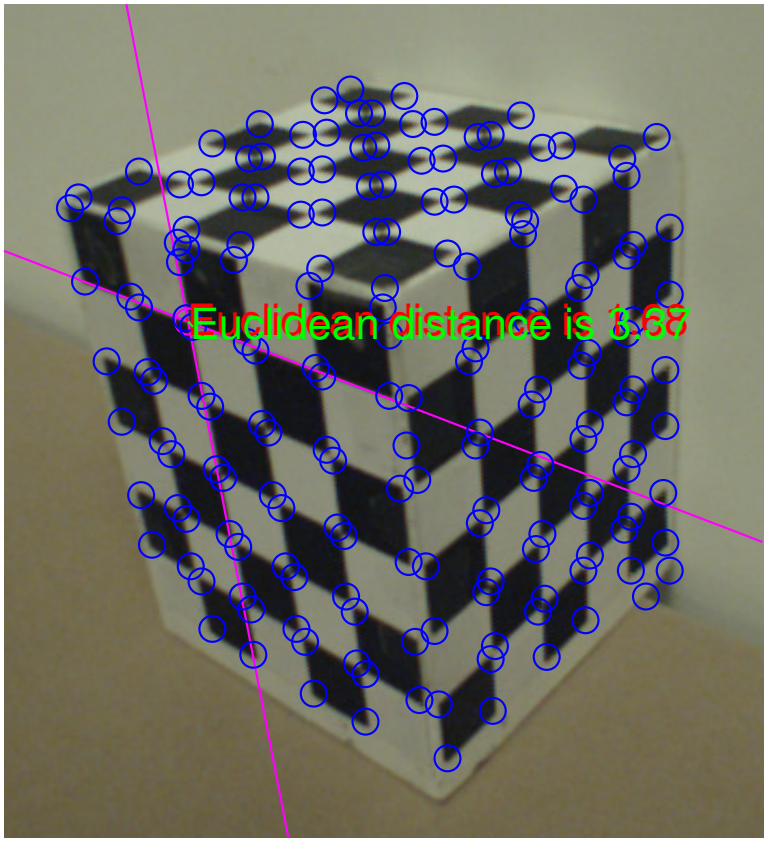
Calculate and display the distance between two corner points that

you obtained with two different methods.

```
% Select closest two corner points to the intersection of two lines:
% Note that they are already from corners struct variable
firstCorn = [85, 146];
secondCorn = [88, 150];
interPoint = [C(1), C(2)];

d1 = norm(firstCorn - interPoint);
d2 = norm(secondCorn - interPoint);
hold on

% Since there is a variable called line, we need to remove it before
% calling line function.
clear line
line([firstCorn(1) interPoint(1)],[firstCorn(2) interPoint(2)],'Color','red','LineStyle'
str1 = sprintf('Euclidean distance is %.2f',d1);
text((firstCorn(1)+interPoint(1))/2, (firstCorn(2)+interPoint(2))/2, str1, 'Color','red'
line([secondCorn(1) interPoint(1)],[secondCorn(2) interPoint(2)],'Color','green','LineSt
str2 = sprintf('Euclidean distance is %.2f',d2);
text((secondCorn(1)+interPoint(1))/2, (secondCorn(2)+interPoint(2))/2, str2, 'Color','gr
```



For better visualization

```
figure
subplot(1,3,1)
imshow("2.jpg"), title("Calibration object image with corners by using two different met
subplot(1,3,2)
imshow("3.jpg"), title("Zoomed area of the intersection point")
subplot(1,3,3)
imshow("5.jpg"), title("Zoomed area of the intersection point with distances")
```

Contents of lab5calibprep2

- Solving the 2 line equations 8 times to find intersection points (corners)
- HARRIS CORNERS
- PLOTTING CORNERS FOR COMPARISON
- Calculate and display the distance between two corner points that

Start

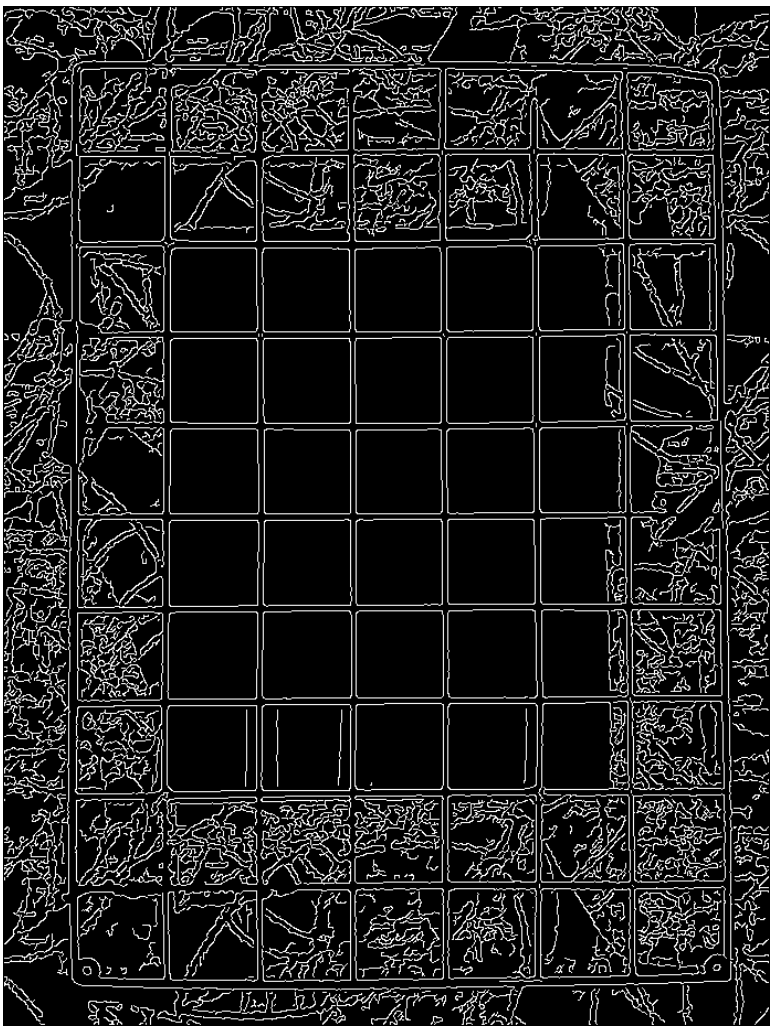
```
clear; close all; clc;

% Read input calibration image
im = imread('myCalObj7.jpg');

% Preprocess the input calibration image
img_resized = imresize(im, 0.1);
img = imrotate(img_resized, 270);

% Check if it is RGB or not. Then, if it is RGB, transform it to
% monochrome image.
[row, col, ch]=size(img);
if ch==3
    img_gray = rgb2gray(img);
end

% Apply canny edge detector
img_edge = edge(img_gray,'canny');
imshow(img_edge);
```



HOUGH TRANSFORM - Extract Lines

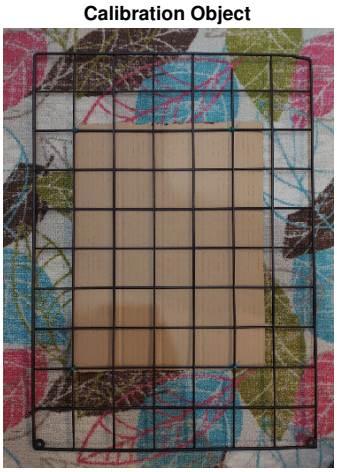
Hough Transform

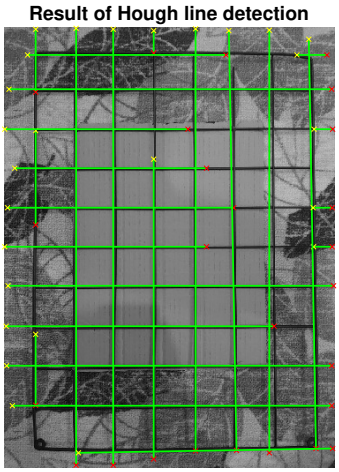
```
[H, theta, rho]=hough(img_edge);
% Hough Peaks
Thresh = ceil(0.5*max(H(:)));
P = houghpeaks(H, 60, "threshold", Thresh);
% Hough Lines with given parameters in lab sheet
line = houghlines(img_edge, theta,rho,P,"FillGap", 80, "MinLength", 30);
```

PLOT HOUGHLINES

```
figure
subplot(1,2,1), imshow(img), title ("Calibration Object")
```

```
subplot(1,2,2), imshow(img_gray), title ("Result of Hough line detection")
hold on
for k = 1:length(line)
    xy = [line(k).point1; line(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','green');
    plot(xy(1,1),xy(1,2),'x','MarkerSize',4,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','MarkerSize',4,'Color','red');
    len = norm(line(k).point1 - line(k).point2);
end
hold off
```





SELECT TWO INTERSECTING LINES MANUALLY

```
Lines_B = [
    2 461;
    230 4;
    23 296;
    403 3;
    23 296;
    230 4;
    23 296;
    315 277;
    8 379;
    315 277;
    8 379;
    403 3;
    8 379;
    230 4;
    8 379;
    473 7;
    ]; % Beginning points
```

```
Lines_E = [
    426 461;
    230 920;
    426 296;
    403 890;
    426 296;
    230 920;
    426 296;
    315 907;
    483 379;
    315 907;
    483 379;
    403 890;
    483 379;
    230 920;
    483 379;
    489 889;
    ]; % End points

% Extract corresponding theta (T) and rho (R) values from the output of 'houghlines' fun

Thetas =[];
Rhos =[];
for p = 1:length(Lines_B)
    for k=1:length(line)
        if ismember(line(k).point1, Lines_B(p,:), "rows") && ismember(line(k).point2, Li
            Thetas = [Thetas; line(k).theta];
            Rhos = [Rhos; line(k).rho];
        end
    end
end
```

PLOT INTERSECTING LINES

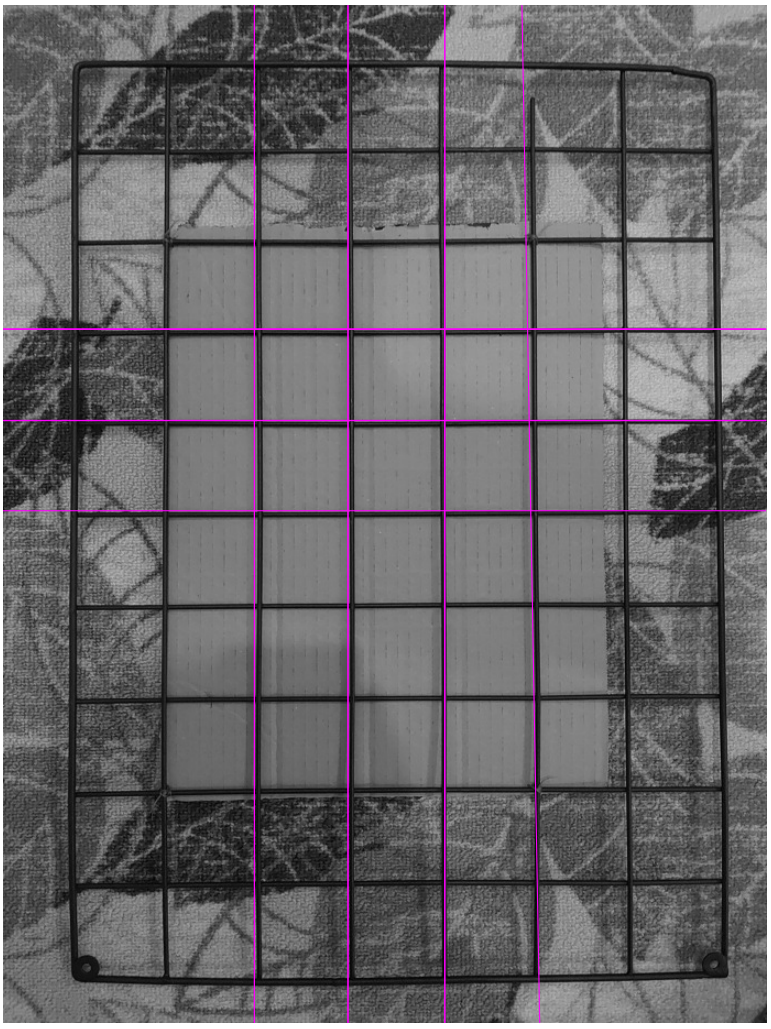
```
x_v = 0:size(img,1);
x_h = 0:size(img,2);

figure
imshow(img_gray)
hold on

for i=1:length(Thetas)
    xPoint = 0:col;
    yPoint=(Rhos(i) -xPoint*cosd(Thetas(i)))/(sind(Thetas(i)+0.00001));
```

```
    plot(xPoint,yPoint, "color", "magenta")

end
```



Solving the 2 line equations 8 times to find intersection points (corners)

```
for i = 1:length(Lines_B)/2
    b = Rhos(2*i-1:2*i);
    A = [cosd(Thetas(2*i-1)), sind(Thetas(2*i-1));cosd(Thetas(2*i)), sind(Thetas(2*i))];
    C(2*i-1:2*i, 1) = A\b;
end
```

HARRIS CORNERS

```
corners = corner(img_gray, 300);
```

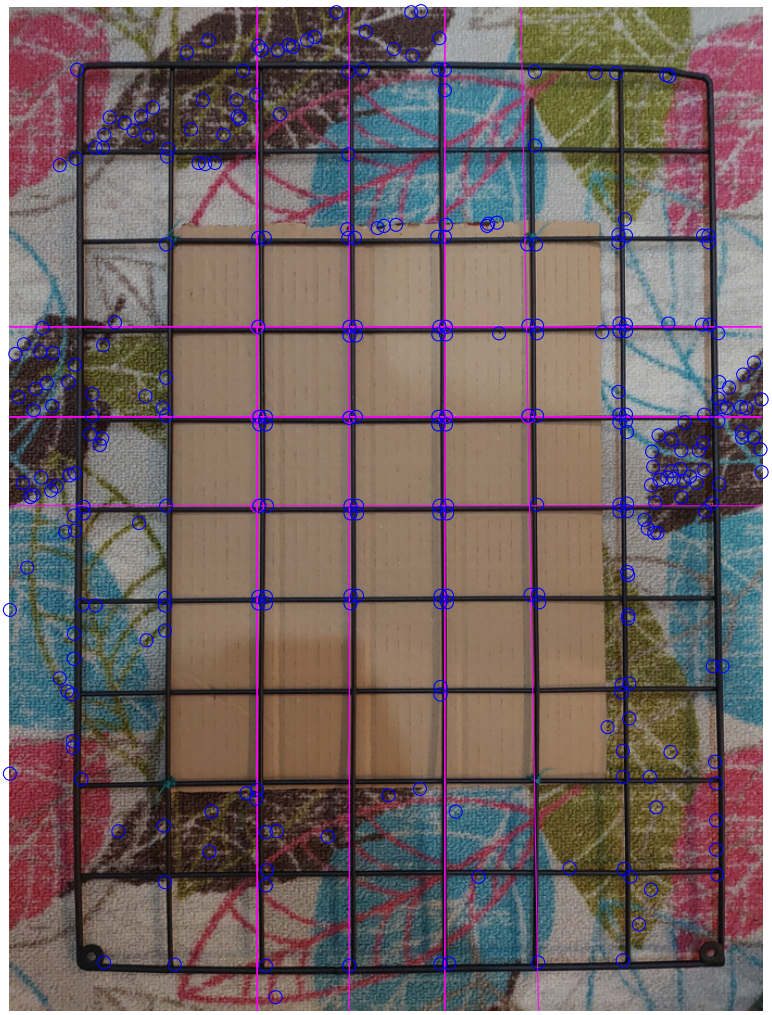
PLOTTING CORNERS FOR COMPARISON

```
figure
imshow(img)
hold on

% plot houghlines intersection point as magenta circle
for i = 1:length(Lines_B)/2
    plot(C(2*i-1), C(2*i),"mo" )
    hold on
end


for i=1:length(Thetas)
    xPoint = 0:col;
    yPoint=(Rhos(i) -xPoint*cosd(Thetas(i)))/sind(Thetas(i)+0.00001);
    plot(xPoint,yPoint, "color", "magenta")

end
% plot harris corners as red circles
plot(corners(:,1),corners(:,2),'bo');
```



Calculate and display the distance between two corner points that

you obtained with two different methods.

```
% Since there is a variable called line, we need to remove it before
% calling line function.
clear line

% Select closest two corner points to the intersection of two lines:
% Note that they are already from corners struct variable
firstCorn = [231, 460];
interPoint1 = [C(1), C(2)];
d1 = norm(firstCorn - interPoint1);
hold on
line([firstCorn(1) interPoint1(1)],[firstCorn(2) interPoint1(2)],'Color','red','LineStyl
str1 = sprintf('Euclidean distance is %.2f',d1);
```

```
text((firstCorn(1)+interPoint1(1))/2, (firstCorn(2)+interPoint1(2))/2, str1, 'Color','re

secCorn = [403, 295];
interPoint2 = [C(3), C(4)];
d2 = norm(secCorn - interPoint2);
hold on
line([secCorn(1) interPoint2(1)],[secCorn(2) interPoint2(2)],'Color','red','LineStyle','
str2 = sprintf('Euclidean distance is %.2f',d2);
text((secCorn(1)+interPoint2(1))/2, (secCorn(2)+interPoint2(2))/2, str2, 'Color','red')

thrCorn = [230, 295];
interPoint3 = [C(5), C(6)];
d3 = norm(thrCorn - interPoint3);
hold on
line([thrCorn(1) interPoint3(1)],[thrCorn(2) interPoint3(2)],'Color','red','LineStyle','
str3 = sprintf('Euclidean distance is %.2f',d3);
text((thrCorn(1)+interPoint3(1))/2, (thrCorn(2)+interPoint3(2))/2, str3, 'Color','red')

fortCorn = [314, 295];
interPoint4 = [C(7), C(8)];
d4 = norm(fortCorn - interPoint4);
hold on
line([fortCorn(1) interPoint4(1)],[fortCorn(2) interPoint4(2)],'Color','red','LineStyle'
str4 = sprintf('Euclidean distance is %.2f',d4);
text((fortCorn(1)+interPoint4(1))/2, (fortCorn(2)+interPoint4(2))/2, str4, 'Color','red'

fifCorn = [314, 379];
interPoint5 = [C(9), C(10)];
d5 = norm(fifCorn - interPoint5);
hold on
line([fifCorn(1) interPoint5(1)],[fifCorn(2) interPoint5(2)],'Color','red','LineStyle','
str5 = sprintf('Euclidean distance is %.2f',d5);
text((fifCorn(1)+interPoint5(1))/2, (fifCorn(2)+interPoint5(2))/2, str5, 'Color','red')

sixCorn = [403, 378];
interPoint6 = [C(11), C(12)];
d6 = norm(sixCorn - interPoint6);
hold on
line([sixCorn(1) interPoint6(1)],[sixCorn(2) interPoint6(2)],'Color','red','LineStyle','
str6 = sprintf('Euclidean distance is %.2f',d6);
text((sixCorn(1)+interPoint6(1))/2, (sixCorn(2)+interPoint6(2))/2, str6, 'Color','red')

sevCorn = [231, 378];
interPoint7 = [C(13), C(14)];
d7 = norm(sevCorn - interPoint7);
```

```
hold on
line([sevCorn(1) interPoint7(1)],[sevCorn(2) interPoint7(2)],'Color','red','LineStyle','
str7 = sprintf('Euclidean distance is %.2f',d7);
text((sevCorn(1)+interPoint7(1))/2, (sevCorn(2)+interPoint7(2))/2, str7, 'Color','red')

eigCorn = [479, 377];
interPoint8 = [C(15), C(16)];
d8 = norm(eigCorn - interPoint8);
hold on
line([eigCorn(1) interPoint8(1)],[eigCorn(2) interPoint8(2)],'Color','red','LineStyle','
str8 = sprintf('Euclidean distance is %.2f',d8);
text((eigCorn(1)+interPoint8(1))/2, (eigCorn(2)+interPoint8(2))/2, str8, 'Color','red')


% %% For better visualization
% figure
% subplot(1,2,1)
% imshow("1.jpg"), title("Zoomed area of the first intersection point")
% subplot(1,2,2)
% imshow("2.jpg"), title("Zoomed area of the second intersection point")
```