EE417 - Assignment 5 - Post-Lab Report

Baris Sevilmis
Lab Date: 13/11/2018
Due Date: 23/11/2018

Overview

Fifth lab assignment of EE417 focuses on Camera Calibration with help of previous methods that were used such as Hough Line Detection. Results of camera calibration in terms of corner detection will be compared, in which sub-pixel accuracy corner detection and Harris Corner Detection results will be discussed.

Part 1: Hough Line Detection

Hough line detection is an unique algorithm in order to detect straight lines and also curves in given image. Two important concepts of this algorithm are the image space and parameters space. Transforming between these spaces provides us features such as detection of collinear edges in an image. Given formulas below ensure the formulas used for transferring between these two spaces:

Image space line equation: $y = mx + n$
Parameter space line equation: $n = (-x)m + y$

These equations are actually same, however second equation provides us to depict lines within the parameter space. As known already, lines in image space are points in parameter space, points in image space are lines in parameter space. Therefore, collinear points in image space are intersecting lines in parameter space. Hough transform technique starts giving algorithm a set of collinear edge points from image space meaning that intersecting lines in parameter space. Given that these lines intersect at point (m,n) in parameter space, this point intersection point refers to a line in image space. Therefore, by giving some collinear points(certain edge points) in image space, and by intersection of corresponding lines in parameter space, at the end we get corresponding lines in image space. For each point in parameter space, array of counters are used in terms of counting lines that are passing through this point. As a result, more lines passing indicate more edges are collinear in image space. Finding a peak point through a threshold in these counter arrays ensures bright points in parameter space. In terms of solving infinite parameter space issue and representation of vertical lines, normal equation of line turns into following sinusoid form in which ρ is distance from origin and $\theta \in [0, \pi]$:

$$\rho = xcos(\theta) + ysin(\theta)$$

With help of these formula straight lines can be detected in image with no issues remaining. As mentioned, this algorithm will provide us the lines that are needed to detect corners in a relative detailed manner. Intersection points of these lines will be considered in order to detect sub-pixel accurate corners.

Part 1: Camera Calibration

For camera calibration, our algorithm begins with Hough Lines Detection as mentioned previously. As peak amount 500 is selected because of having high amount of intersecting lines in image space. Hough lines are detected and plotted on the greyscale version of image, in which intersecting lines are saved. These intersecting lines will be redrawn on another greyscale version of image in terms of focusing only on these specific lines. There will be 16 lines in total in order to extract 8 sub accurate pixel intersection points. Meanwhile, on this image Harris Corner Detector will detect corner points. In the end, greyscale image will contain both the subaccuracy corner detection results and Harris Corner Detection results. Comparing euclidean distance of nearest detected corner points of these two algorithms will demonstrate the success rate difference of both algorithms in terms of corner detection. Following table depicts start and end points of these intersecting lines in addition to their θ and ρ values:

Intersection	Start Point	End Point	θ	ρ	Harris Corner
1	[24, 130]	[308, 100]	84	131	[149, 117]
1	[149, 77]	[149, 161]	-0.5	147	[149, 117]
2	[27, 470]	[296, 574]	-69	-429	[62.2, 483.3]
2	[62, 400]	[62, 511]	-0.5	57	[62.2, 483.3]
3	[24, 130]	[308, 100]	84	131	[298, 101]
3	[298, 60]	[297, 208]	0.5	298	[298, 101]
4	[21, 200]	[231, 200]	-90	-199	[61, 200]
4	[59, 143]	[60, 239]	-0.5	57	[61, 200]
5	[21, 200]	[231, 200]	-90	-199	[150, 200]
5	[150, 173]	[150, 242]	-0.5	147	[150, 200]
6	[27, 437]	[297, 527]	-71.5	-405	[43, 442]
6	[43, 402]	[43, 480]	-0.5	38	[43, 442]
7	[23, 336]	[314, 390]	-79.5	-325	[151, 117]
7	[150, 279]	[151, 401]	-0.5	147	[151, 117]
8	[22, 235]	[190, 242]	-87.5	-233	[41, 236]
8	[41, 198]	[42, 341]	-0.5	38	[41, 236]

Listing 1, 2, 3, 4 and 5 are providing the required MatLab code for camera calibration task including all the necessary plot functions and distance calculation. To be more specific, Listing 1 contains the Harris Corner Detection and Hough Lines Detection. Listing 2, 3 and 4 provide calculation steps of 8 intersection points, therefore necessary sinusoid line functions and matrix multiplications to compute intersections. Finally, Listing 5 ensures all of the distance calculations between sub-accurate detected corners and Harris Detected Corners. For the sake of testing, calibration image was converted into 640 x 600 format in terms of pixels.

Listing 1: MatLab Code for Camera Calibration - Part 1

```matlab
   function [J, lines] = lab5calibprep(img)
    [row, col, ch] = size(img);
    if(ch == 3)
        rgbimg = rgb2gray(img);
    end
    %Canny edge detection
    J = edge(rgbimg,'Canny');
    %Hough Transform
    [H,theta,rho] = hough(J,'RhoResolution',1,'Theta'
       ,-90:0.5:89);
    %Hough Peaks
    threshold = 0.5*max(H(:));
    J_peaks = houghpeaks(H,500, 'Threshold', 100);
    figure(1);
    imshow(img);title('Calibration Object');

    figure(2);
    imshow(rgbimg);title('Results of Hough Line Detection');
    hold on;
    %Hough Lines
    lines = houghlines(J,theta,rho,J_peaks,'FillGap',10, '
       MinLength',25);

    %Hough Lines Plot
    max_len = 0;
    min_len = 200;
    for k = 1:length(lines)
        xy = [lines(k).point1; lines(k).point2];
        plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

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

        % Determine the endpoints of the longest line segment
        len = norm(lines(k).point1 - lines(k).point2);

    end
    %Harris
    figure(3);
    imshow(rgbimg);title('Harris Corner Detection');
    hold on;
    C = corner(rgbimg, 'N', 350);
    plot(C(:,1),C(:,2),'bo');
```

Listing 2: MatLab Code for Hough Line Detection- Part 1

```matlab
%Intersect 1
rho1 = lines(35).rho;
theta1 = lines(35).theta;
rho2 = lines(12).rho;
theta2 = lines(12).theta;
X1 = 0:0.1:350;
Y1 = (rho1 - X1*cosd(theta1))/sind(theta1);
plot(X1,Y1,'m');
X2 = 0:0.1:350;
Y2 = (rho2 - X2*cosd(theta2))/sind(theta2);
plot(X2,Y2,'m')
A = [cosd(theta1) sind(theta1); cosd(theta2) sind(theta2)];
rhoMat = [rho1; rho2];
invA = pinv(A);
point = invA * rhoMat;
plot(point(1), point(2),'rx');

%Intersect 2
rho3 = lines(32).rho;
theta3 = lines(32).theta;
rho4 = lines(38).rho;
theta4 = lines(38).theta;
X3 = 0:0.1:350;
Y3 = (rho3 - X3*cosd(theta3))/sind(theta3);
plot(X3,Y3,'m')
X4 = 0:0.1:350;
Y4 = (rho4 - X4*cosd(theta4))/sind(theta4);
plot(X4,Y4,'m')
A2 = [cosd(theta3) sind(theta3); cosd(theta4) sind(theta4)];
rhoMat2 = [rho3; rho4];
invA2 = pinv(A2);
point2 = invA2 * rhoMat2;
plot(point2(1), point2(2),'rx');

 %Intersect 3
rho6 = lines(3).rho;
theta6 = lines(3).theta;
X6 = 0:0.1:350;
Y6 = (rho6 - X6*cosd(theta6))/sind(theta6);
plot(X6,Y6,'m')
A3 = [cosd(theta1) sind(theta1); cosd(theta6) sind(theta6)];
rhoMat3 = [rho1; rho6];
invA3 = pinv(A3);
point3 = invA3 * rhoMat3;
plot(point3(1), point3(2),'rx');
```

Listing 3: MatLab Code for Hough Line Detection- Part 1

```matlab
    %Intersect 4
    rho7 = lines(29).rho;
    theta7 = lines(29).theta;
    rho8 = lines(37).rho;
    theta8 = lines(37).theta;
    X7 = 0:0.1:350;
    Y7 = (rho7 - X7*cosd(theta7))/sind(theta7);
    plot(X7,Y7,'m');
    X8 = 0:0.1:350;
    Y8 = (rho8 - X8*cosd(theta8))/sind(theta8);
    plot(X8,Y8,'m')
    A4 = [cosd(theta7) sind(theta7); cosd(theta8) sind(theta8)];
    rhoMat4 = [rho7; rho8];
    invA4 = pinv(A4);
    point4 = invA4 * rhoMat4;
    plot(point4(1), point4(2),'rx');

    %Intersect 5
    rho10 = lines(13).rho;
    theta10 = lines(13).theta;
    X10 = 0:0.1:350;
    Y10 = (rho10 - X10*cosd(theta10))/sind(theta10);
    plot(X10,Y10,'m')
    A5 = [cosd(theta7) sind(theta7); cosd(theta10) sind(theta10)
       ];
    rhoMat5 = [rho7; rho10];
    invA5 = pinv(A5);
    point5 = invA5 * rhoMat5;
    plot(point5(1), point5(2),'rx');

    %Intersect 6
    rho11 = lines(48).rho;
    theta11 = lines(48).theta;
    rho12 = lines(24).rho;
    theta12 = lines(24).theta;
    X11 = 0:0.1:350;
    Y11 = (rho11 - X11*cosd(theta11))/sind(theta11);
    plot(X11,Y11,'m');
    X12 = 0:0.1:350;
    Y12 = (rho12 - X12*cosd(theta12))/sind(theta12);
    plot(X12,Y12,'m')
    A6 = [cosd(theta11) sind(theta11); cosd(theta12) sind(
       theta12)];
    rhoMat6 = [rho11; rho12];
    invA6 = pinv(A6);
    point6 = invA6 * rhoMat6;
    plot(point6(1), point6(2),'rx');
```

Listing 4: MatLab Code for Hough Line Detection- Part 1

```
    %Intersect 7
    rho13 = lines(36).rho;
    theta13 = lines(36).theta;
    rho14 = lines(14).rho;
    theta14 = lines(14).theta;
    X13 = 0:0.1:350;
    Y13 = (rho13 - X13*cosd(theta13))/sind(theta13);
    plot(X13,Y13,'m')
    X14 = 0:0.1:350;
    Y14 = (rho14 - X14*cosd(theta14))/sind(theta14);
    plot(X14,Y14,'m')
    A7 = [cosd(theta13) sind(theta13); cosd(theta14) sind(
       theta14)];
    rhoMat7 = [rho13; rho14];
    invA7 = pinv(A7);
    point7 = invA7 * rhoMat7;
    plot(point7(1), point7(2),'rx');

    %Intersect 8
    rho15 = lines(64).rho;
    theta15 = lines(64).theta;
    rho16 = lines(23).rho;
    theta16 = lines(23).theta;
    X15 = 0:0.1:350;
    Y15 = (rho15 - X15*cosd(theta15))/sind(theta15);
    plot(X15,Y15,'m');
    X16 = 0:0.1:350;
    Y16 = (rho16 - X16*cosd(theta16))/sind(theta16);
    plot(X16,Y16,'m')

    eucliddif1 = sqrt((point(1)-149)^2 + (point(2)-117)^2);
    disp(Euclidean distance between Corner Detection 1:);
    disp(eucliddif1);

    eucliddif2 = sqrt((point2(1)-62.2)^2 + (point2(2)-483.3)^2);
    disp(Euclidean distance between Corner Detection 2:);
    disp(eucliddif2);

    eucliddif3 = sqrt((point3(1)-298)^2 + (point3(2)-101)^2);
    disp(Euclidean distance between Corner Detection 3:);
    disp(eucliddif3);
```

Listing 5: MatLab Code for Hough Line Detection- Part 1

```
    eucliddif4 = sqrt((point4(1)-61)^2 + (point4(2)-200)^2);
    disp(Euclidean distance between Corner Detection 4:);
    disp(eucliddif4);

    eucliddif5 = sqrt((point5(1)-150)^2 + (point5(2)-200)^2);
    disp(Euclidean distance between Corner Detection 5:);
    disp(eucliddif5);

    eucliddif6 = sqrt((point6(1)-43)^2 + (point6(2)-442)^2);
    disp(Euclidean distance between Corner Detection 6:);
    disp(eucliddif6);

    eucliddif7 = sqrt((point7(1)-151)^2 + (point7(2)-359)^2);
    disp(Euclidean distance between Corner Detection 7:);
    disp(eucliddif7);

    eucliddif8 = sqrt((point8(1)-41)^2 + (point8(2)-236)^2);
    disp(Euclidean distance between Corner Detection 8:);
    disp(eucliddif8);

end
```

As details of the algorithm and the code are provided above, it would be reasonable to discuss results of sub-accurate corner detection and Harris corner detection. Figure 1 depicts the original calibration image that is two checkerboard images on different planes.

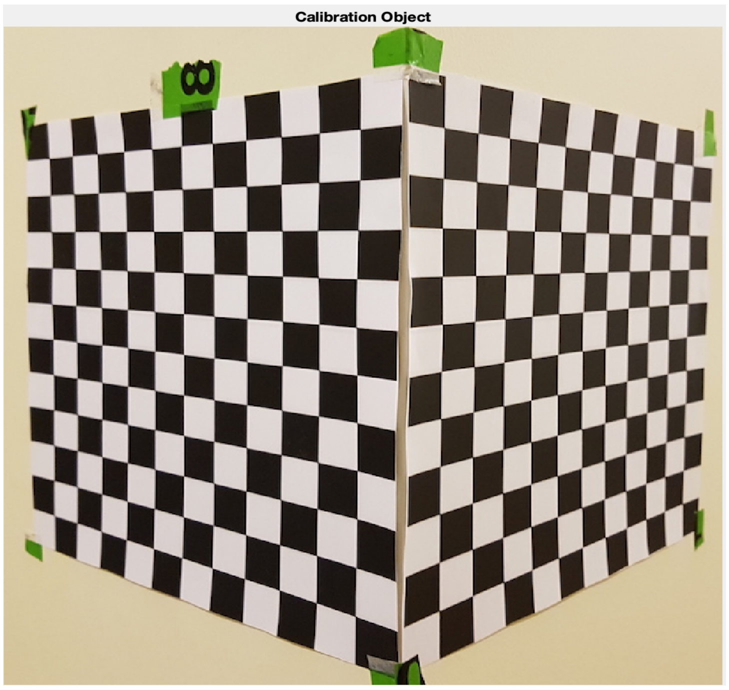


Figure 1: Original Image

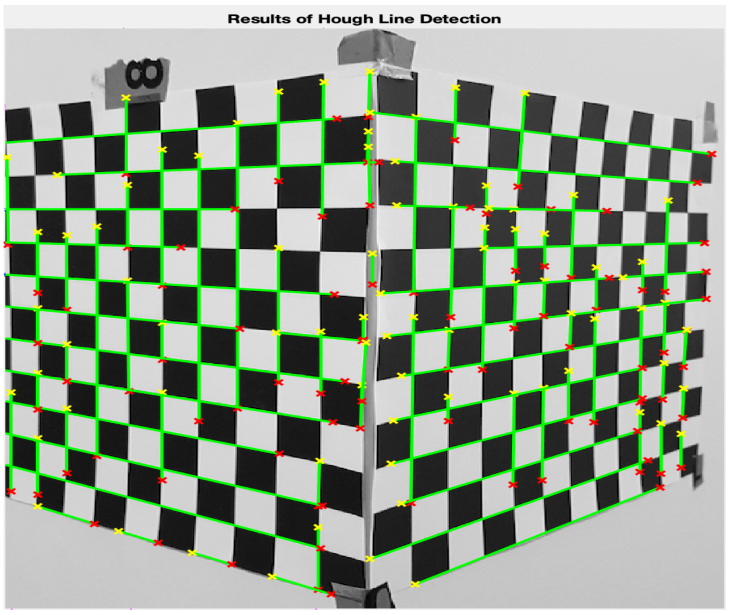


Figure 2: Hough Line Detection with 500 Peak Points and Threshold = 100

Figure 2 demonstrates the greyscale version of the image. In addition, results of Hough Line Detection can be seen. Parameters for the Hough Lines detection are 500 peak points and Threshold = 100. Start of the lines can be seen as yellow marks and end of the lines are the red marks. Intersection of these lines result in sub accurate corner detection. To detect 8 intersection, normally 16 different lines must be chosen. However, in our case there one line may have more than one intersection such that obtaining 8 intersection points require less than 16 different lines. Lines that are used for sub accurate corner detection are depicted in the table above the Listing part in terms of their start points, end points, ρ, θ and nearest detected Harris corner point.

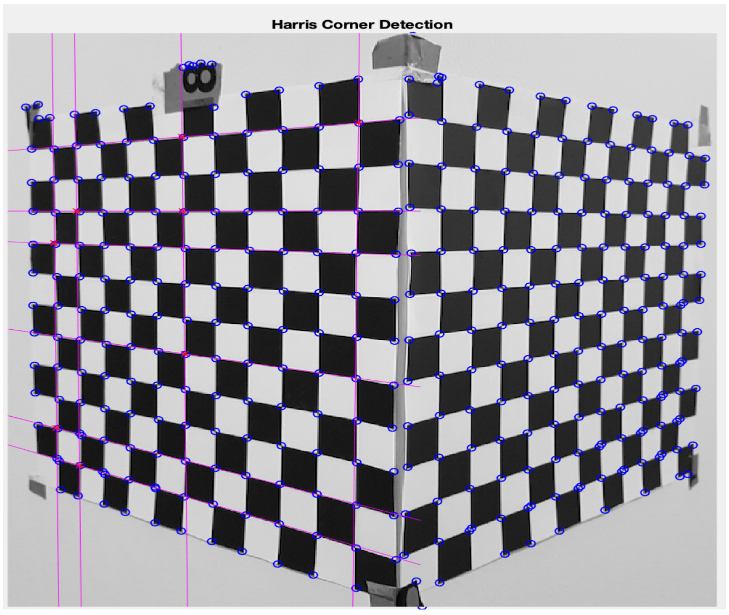


Figure 3: Sub accurate corner detection by intersection of lines and Harris Corners

Figure 3 depicts Harris Corners detection as blue marks and sub accurate corner detection as intersection of magenta lines with red marks. As both corner detection algorithms are applied, results of both algorithms become arguable. To see the difference of accuracy between two detected corners, zooming in one of the detected corners by both of the algorithms would be reasonable. Therefore, Figure 4 and 5 depict a specific detected corner point and corresponding sub accurate and Harris corner detection coordinates. Both of the detection coordinates are close but different. Their euclidean distance will be a reliable measure to see magnitude of the difference. It is clear that sub accurate corner detection gives better corner detection results than Harris Corner detection.

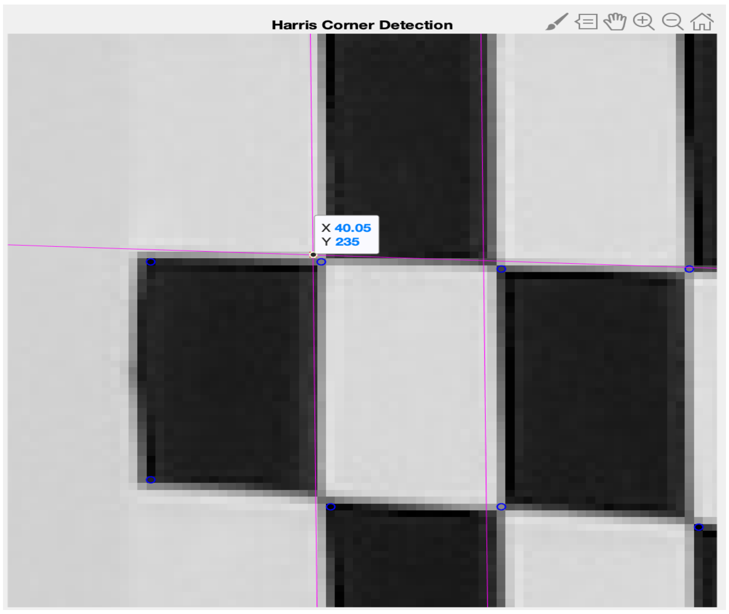


Figure 4: Sub accurate corner detection with corresponding coordinates

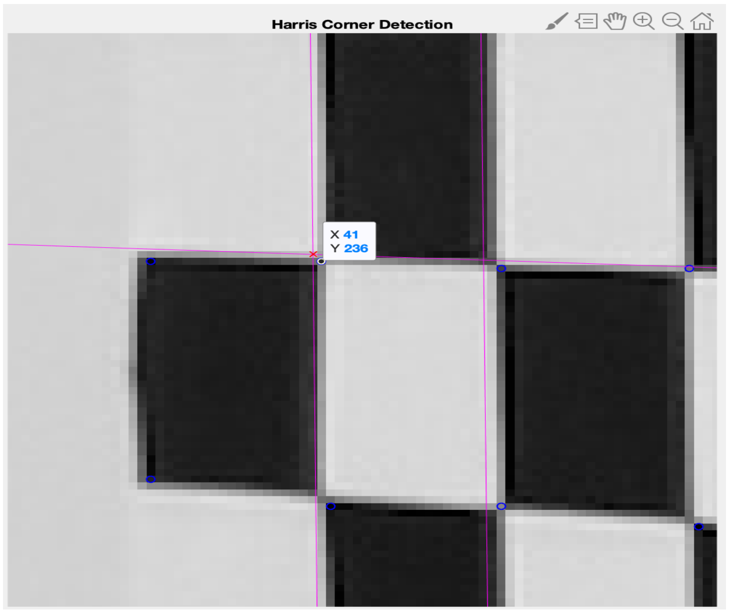


Figure 5: Harris corner detection with corresponding coordinates

Figure 6, 7, 8, 9 and 10 provide other Harris Corners and different sub accurate corners as intersection of two lines. Figure 7 depicts again the clear distinction of both algorithms corner detection accuracy. On these figures, main goal is to create different point of views to see singular or multiple corner detections in terms of improving the understanding of difference between the two given corner detection algorithms.

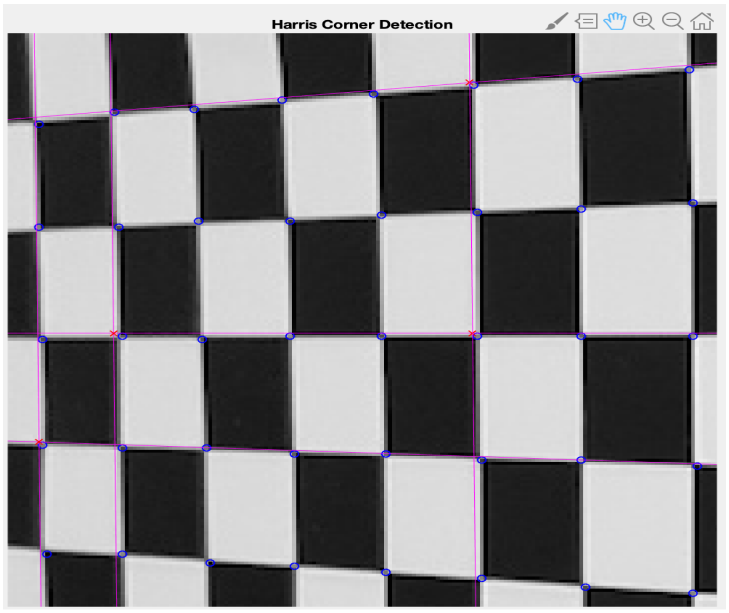


Figure 6: Sub accurate corner detection by intersection of lines and Harris Corners(Different View)

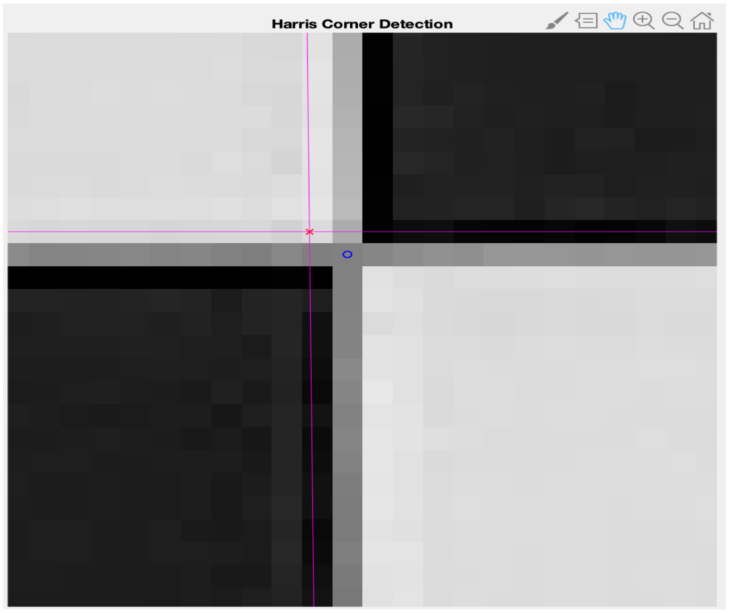


Figure 7: Sub accurate corner detection by intersection of lines and Harris Corners(Different View)

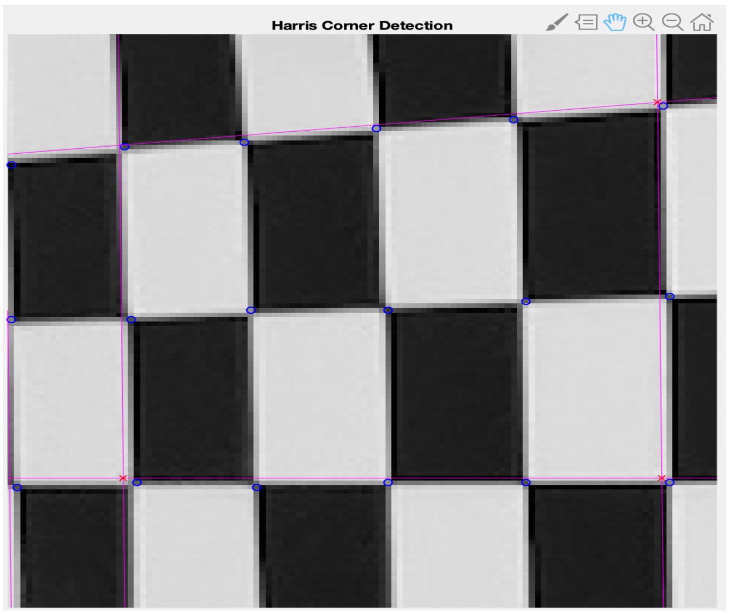


Figure 8: Sub accurate corner detection by intersection of lines and Harris Corners(Different View)

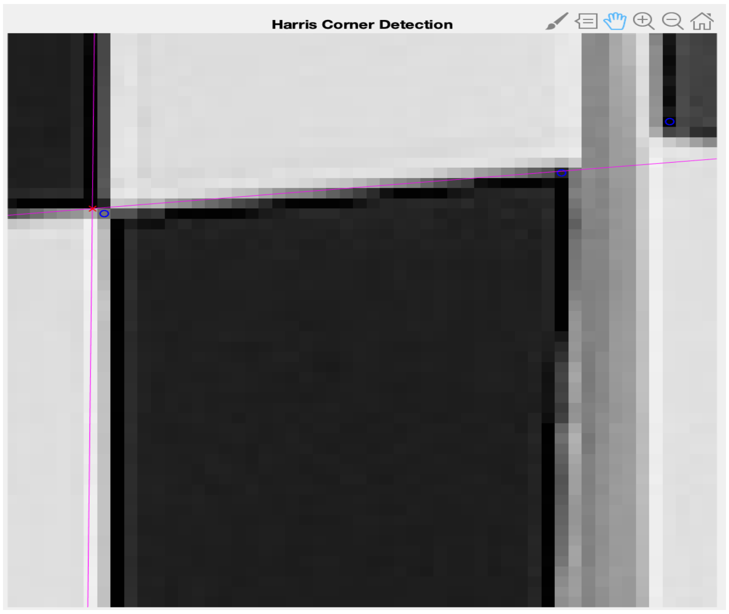


Figure 9: Sub accurate corner detection by intersection of lines and Harris Corners(Different View)

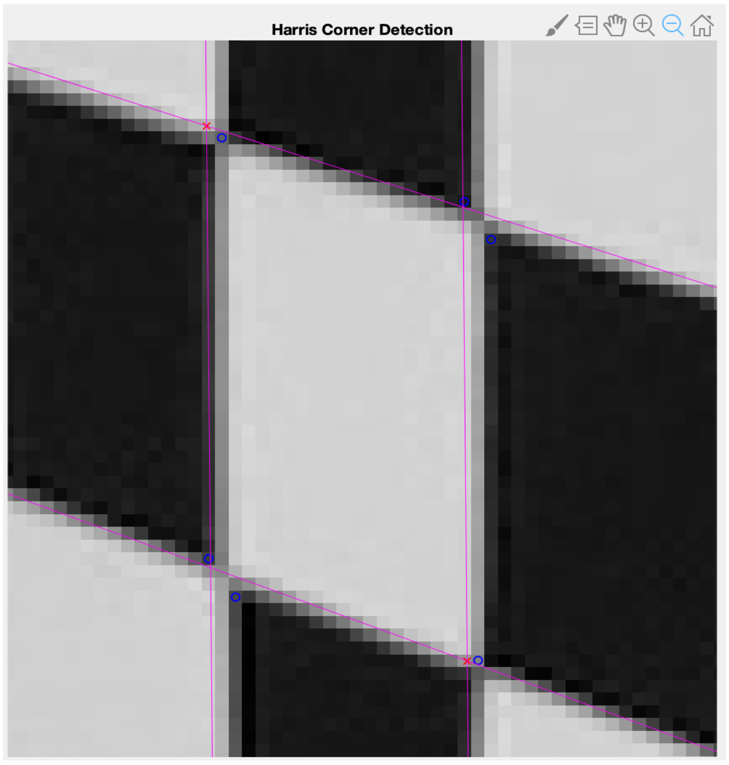
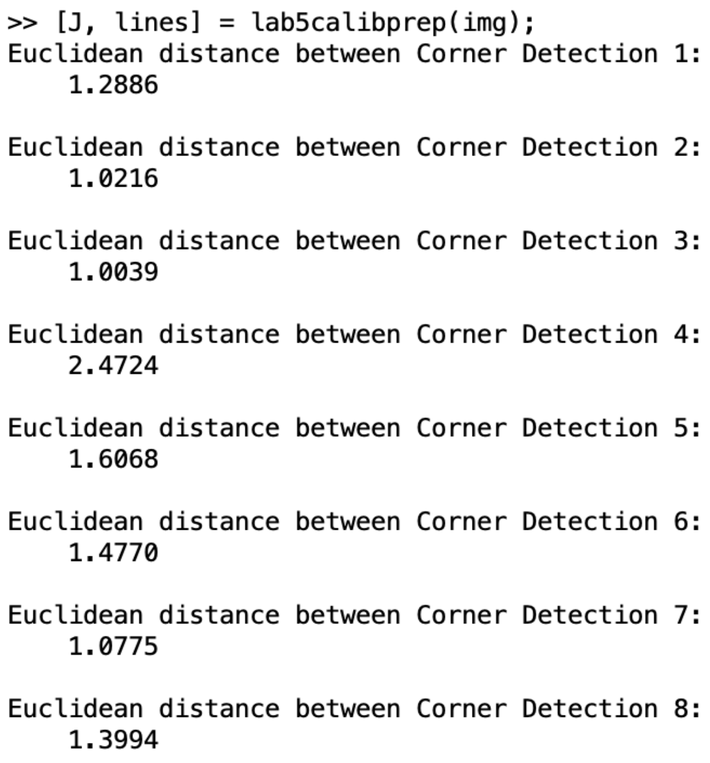


Figure 10: Sub accurate corner detection by intersection of lines and Harris Corners(Different View)

To conclude, Figure 11 provides all the 8 euclidean magnitudes between sub accurate detected corners and Harris corners. Following formula is used as s stands for sub accurate corner points and h stands for Harris corners:

$$d(x,y) = \sqrt{(x_s - x_h)^2 + (y_s - y_h)^2}$$

As it can be seen all distances are between 1 and 2 proving again sub accurate corner detection to be more accurate. However, this result also provides that Harris Corner detection ensures accuracy as well, but in case of certain corner detection sub accurate corner detection provides best results.



```
>> [J, lines] = lab5calibprep(img);
Euclidean distance between Corner Detection 1:
    1.2886

Euclidean distance between Corner Detection 2:
    1.0216

Euclidean distance between Corner Detection 3:
    1.0039

Euclidean distance between Corner Detection 4:
    2.4724

Euclidean distance between Corner Detection 5:
    1.6068

Euclidean distance between Corner Detection 6:
    1.4770

Euclidean distance between Corner Detection 7:
    1.0775

Euclidean distance between Corner Detection 8:
    1.3994
```

Figure 11: Sub accurate corner detection by intersection of lines and Harris Corners(Different View)