Lab #5 Post-Lab Report

pciftcioglu@sabanciuniv.edu

December 2020

1 Introduction

In this lab, we tried to do operation for camera calibration preparation. Camera calibration is an essential operation to understand how to objects have been placed relative to each other, relative to camera and the world to make transitions between 2D image plane, 3D world coordinate system and camera coordinates.

For the post lab experiments, it has been conducted a camera calibration preparation on a calibration object which has been created by me and the corners and lines are extracted based on that object. The function lab5calibprep.m function has been utilized for this purpose and the actual image used can be found in the attachment.

2 In lab Implementations

In the lab session, we implemented the function lab5calibprep.m that has been extracted from a template provided to us and the methods utilized as the following: First we made the image gray scale as we always do and applied a proper edge detection with using the built-in function **edge(img_gray,'canny')** where img_gray is the gray scale version of the original image and the used method was Canny edge detector. After that, we performed the Hough Transform, extracted peaks and detected Hough Lines with again using built-in MATLAB functions. And 600 has been chosen for peaks count with a threshold **ceil(0.5*max(H(:)))**. Then lines have been extracted with FillGap = 5 and MinLength = 4 used as the parameters which corresponds to the bin that is separated less than 5 and disregarding the lines shorter than 4 respectively. Then we visualized the lines with their start and end points on the original image as can be seen in Figure 2 as an illustration in the post lab section. Then we selected two intersecting lines where the intersect point would correspond to a corner from the image and stored the beginning and end points of those lines in two different arrays. then we extracted theta and rho values for every line in the line array and chose the appropriate values that are belong to those lines with using another built-in function ismember() (see Appendix for implementation and parameters) and stored the corresponding values in different arrays which will be used for line equation later on. Finally, we plotted the intersection points of the two lines we selected by solving their equations based as two unknown variable equation system. Additionally, we also found the corners using another operation which is the Harris corners to compare which one resulted in better corners. The method implemented using a built-in function **corner(img_gray,"harris")** where again img_gray corresponds to the original but gray scale version of the image. Then we also plotted the Harris Corners as well as the extraction we implemented with Hough Lines.

3 Post Lab Experiment

First, to use as a calibration object, I printed out 2 checker board images and placed them onto a wall corner in some place in my room with a 90 degree angle and took a picture of it from an arbitrary angle. After that, I lowered the resolution of the image because when I applied the smoothing operation and hough transform on the original one, the methods were failing to work as accurate. One possible reason for that is might be because there are a lot of pixels assigned to corners and where the lines should cross and that caused a lot of scatter around where should lines cross and and corners detected. And even though after lot of trials with different thresholds and edge detecting methods and different smoothing operations, I could not came further in terms of accuracy; thus, I decided to lower the image resolution. The image I took can be seen in the Figure 1 below.

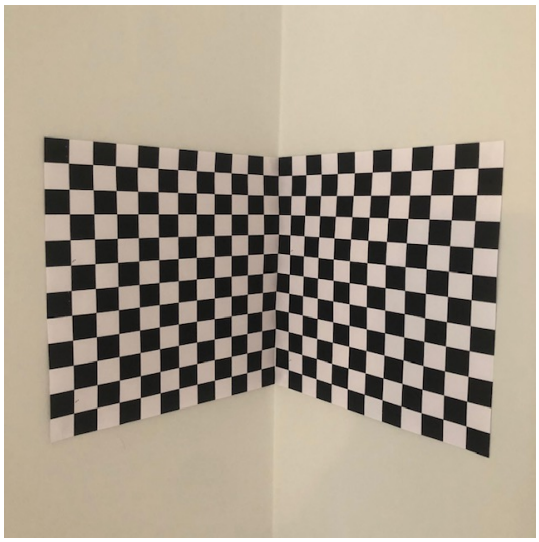


Figure 1: Calibration Object

For the rest, the function implemented in the lab has been used with the slight differences as the following. In Canny edge detection, the threshold and the standard deviation value has been selected as 0.9 and 0.8 respectively, based on the trials which resulted in optimum in those values. And 500 Hough peaks has been selected to determine the lines and **FillGap** and **MinLength** parameters has been assigned as 12 and 35 respectively based on the appropriate relative scale of the pixels in the new calibration object and the resulting lines can be seen from the Figure 2.

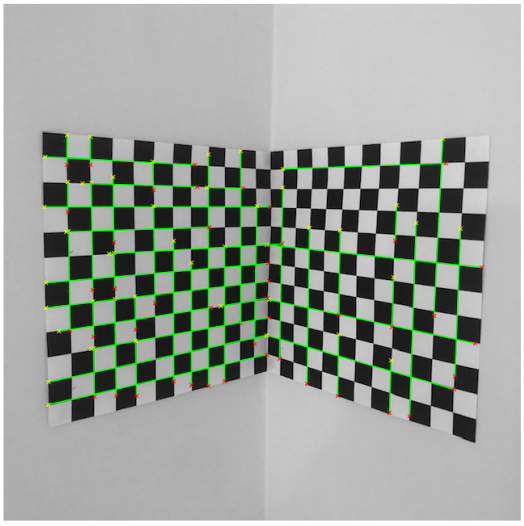


Figure 2: Hough Lines

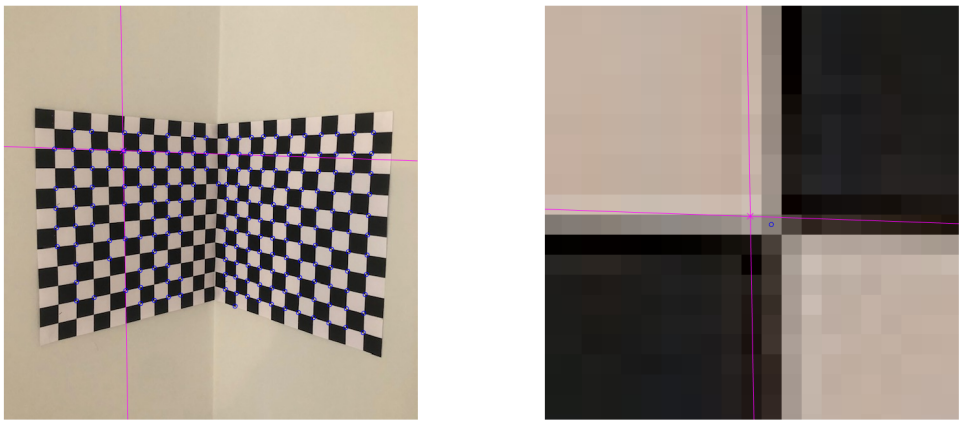
Then, manually selecting points from the image figure in MATHLAB I picked following crossing lines where tuples indicates beginning and end points as the following:

- 1. Point: Beginning Points: (85, 196), (98, 223), End Points: (186, 231), (240, 227).
- 2. Point: Beginning Points: (36, 215), (51, 252), End Points: (137, 296), (330, 252).

- 3. Point: Beginning Points: (81, 282), (53, 346), End Points: (83, 405), (328, 322).
- 4. Point: Beginning Points: (544, 168), (326, 299), End Points: (533, 387), (592, 327).
- 5. Point: Beginning Points: (417, 281), (326, 299), End Points: (415, 386), (592, 327).
- 6. Point: Beginning Points: (326, 411; 511 272 , End Points: (558, 478), (503, 511).
- 7. Point: Beginning Points: (53, 346), (137, 309), End Points: (328, 322), (138, 353).
- 8. Point: Beginning Points: (52, 190), (108, 165), End Points: (237,200), (108, 200).

The resulting corners with the lines selected will be listed with their visuals from the distant perspective and zoomed in versions as well as with the Harris Corners for comparison. I put visualized them in different images to avoid confusion but all the points should be listed together for further implementations since it needs at least 8 points for calibration. Magenta colored marker with shape X indicates the corner we calculated and the blue spheres indicates the Harris Corners.

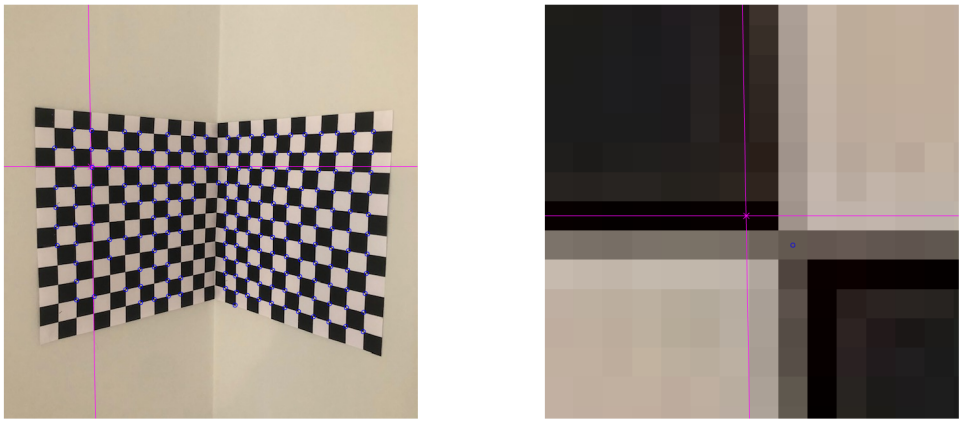
- 1. Point:



(a) First Corner Point Lines (b) First Corner Point and Harris Corner

Figure 3: First Point

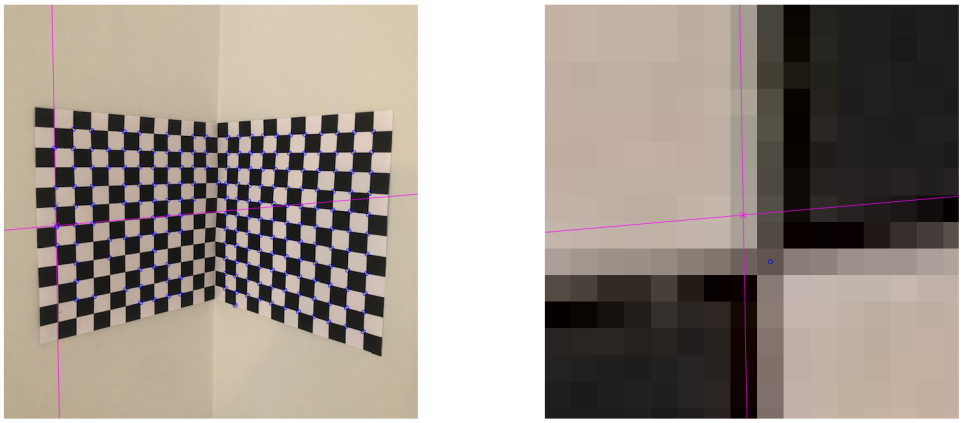
- 2. Point:



(a) Second Corner Point Lines (b) Second Corner Point and Harris Corner

Figure 4: Second Point

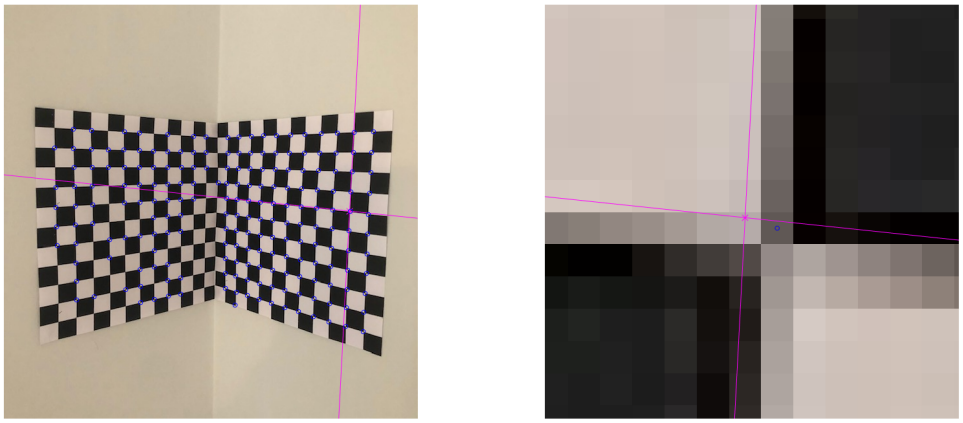
- 3. Point:



(a) Third Corner Point Lines (b) Third Corner Point and Harris Corner

Figure 5: Third Point

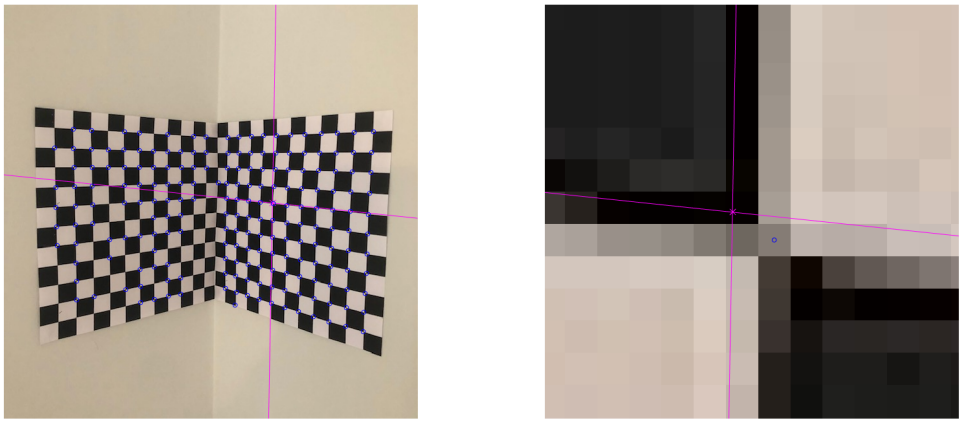
- 4. Point:



(a) Fourth Corner Point Lines (b) Fourth Corner Point and Harris Corner

Figure 6: Fourth Point

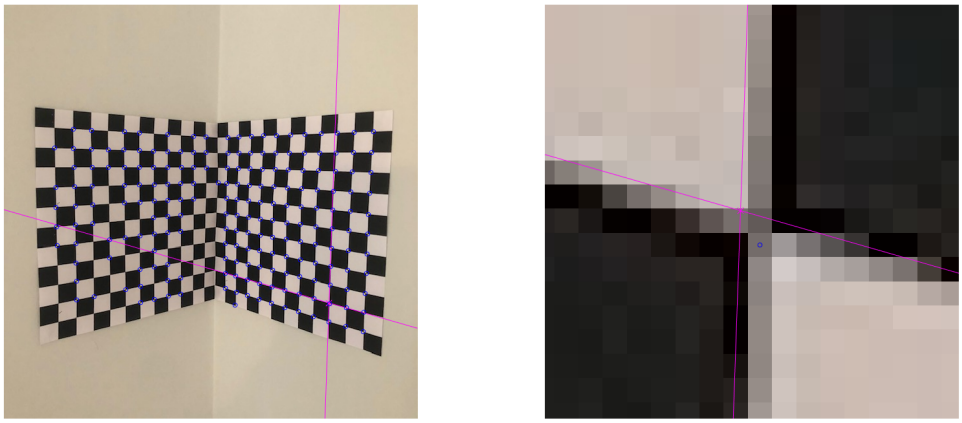
- 5. Point:



(a) Fifth Corner Point Lines (b) Fifth Corner Point and Harris Corner

Figure 7: Fifth Point

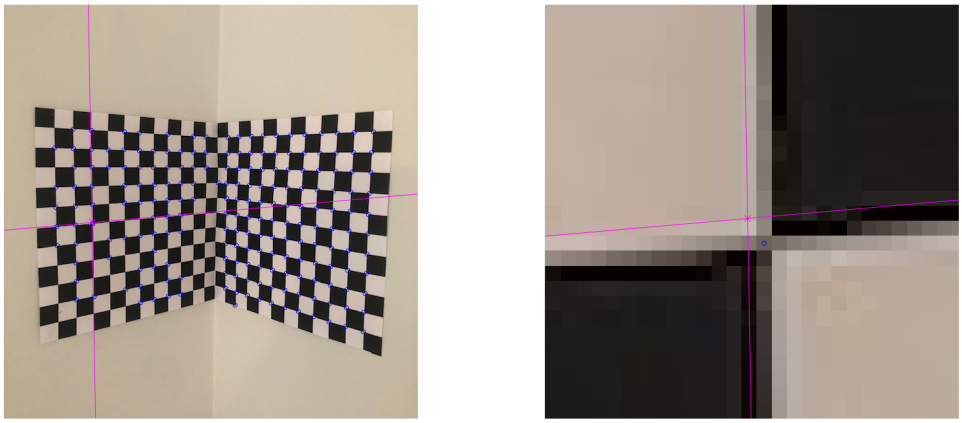
- 6. Point:



(a) Sixth Corner Point Lines (b) Sixth Corner Point and Harris Corner

Figure 8: Sixth Point

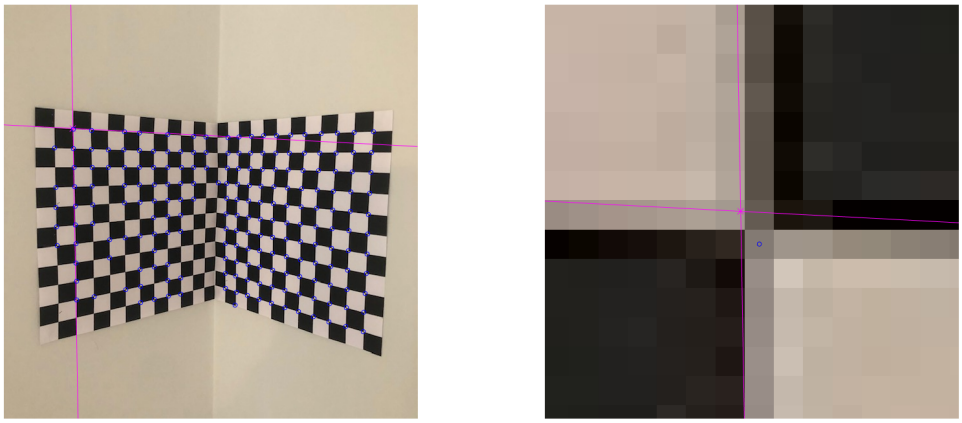
- 7. Point:



(a) Seventh Corner Point Lines (b) Seventh Corner Point and Harris Corner

Figure 9: Seventh Point

- 8. Point:



(a) Eighth Corner Point Lines (b) Eighth Corner Point and Harris Corner

Figure 10: Eighth Point

As can be seen from the results above, Harris corners resulted in a more accurate corners than our calculated ones in all of them if we consider the corner point as the diagonal pixel point in between two different black (or white) boxes. However, the method also accomplished a quite good result with being mistaken only couple of pixel points in most of them which is even too hard to observe when looked globally.

It is possible to indicate that the accuracy might have affected by the calculations based on the used method's sensitivity so further trials should be done considering different threshold values for further implementations so the better results might be observed. But for the current experiment Harris corners has done a pretty good job contrary in the results we have observed during the lab which resulted in better corners when we implemented the results. However, it is also known that the examples provided has been given based on the optimal values.

Thus, as I also observed with my own experiments in terms of resolution of the calibration object's image and thresholds for Hough and edge detector, every image has its unique pixel values and orientation so it should be considered and evaluated individually in order to get the best results for each different calibration object.

Appendix

```
clear all; close all; clc;

img = imread('calibrationObjectSelfMadeHigherRes.jpg');
[row, col, ch]=size(img);
if ch==3
    img_gray = rgb2gray(img);
end

% Returning image is Black and White
img_edge = edge(img_gray,'canny',0.9,0.8);
%imshow(img_edge);

%% HOUGH TRANSFORM - Extract Lines
[H,theta,rho] = hough(img_edge);
P  = houghpeaks(H,500,'threshold',ceil(0.5*max(H(:))));
line = houghlines(img_edge,theta,rho,P,'FillGap',12,'MinLength',35);

%% PLOT HOUGHLINES
figure
%subplot(1,2,1), imshow(img)
subplot(1,1,1), imshow(img_edge)
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 = [185 196; 98 223]; % Beginning points
Lines_E =  [186 231; 240 227]; % End points

% Lines_B = [136 215; 51 252]; % Beginning points
% Lines_E =  [137 296; 330 252]; % End points

% Lines_B = [81 282; 53 346]; % Beginning points
% Lines_E =  [83 405; 328 322]; % End points

% Lines_B = [544 168; 326 299]; % Beginning points
% Lines_E =  [533 387; 592 327]; % End points

% Lines_B = [417 281; 326 299]; % Beginning points
% Lines_E =  [415 386; 592 327]; % End points

% Lines_B = [326 411; 511 272 ]; % Beginning points
% Lines_E =  [558 478; 503 511 ]; % End points
%
% Lines_B = [53 346; 137 309]; % Beginning points
% Lines_E =  [328 322 ; 138 353]; % End points

% Lines_B = [52 190; 108 165]; % Beginning points
% Lines_E =  [237 200; 108 200]; % End points

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

Thetas = [];
Rhos = [];
for p = 1:2
    for k=1:length(line)
        if (ismember(line(k).point1, Lines_B(p,:),'row') && ismember(line(k).point2, Lines_E(p,:)
    ,'row'))
            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 = x_h;
   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(1);Rhos(2)];
A = [cosd(Thetas(1)) , sind(Thetas(1));
     cosd(Thetas(2)) , sind(Thetas(2))];
C = A\b;
%% HARRIS CORNERS
harris = corner(img_gray,"harris");
plot(C(1), C(2), 'x', "MarkerSize",8,'Color', 'magenta');
%% PLOTTING CORNERS FOR COMPARISON
plot(harris(:,1),harris(:,2),'bo',"MarkerSize",4);
```