EE 417

COMPUTER VISION

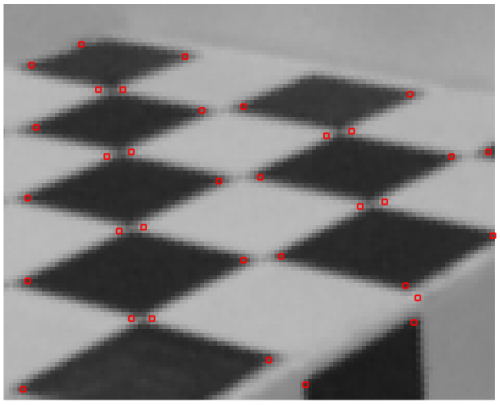
LAB#5 POST-LAB REPORT

Goktug Korkulu
27026
22 November 2022

INTRODUCTION

There are various kind of methods in terms of the corner point detection application. In the past weeks, we have worked on 3 kinds of corner point detection methods under the name of Kanade-Tomasi and Harris algorithms. This week, we will make improvements on those previously examined corner point detection notions and compare the precision ratio of the new approach with ***Harris Corners*** algorithm.

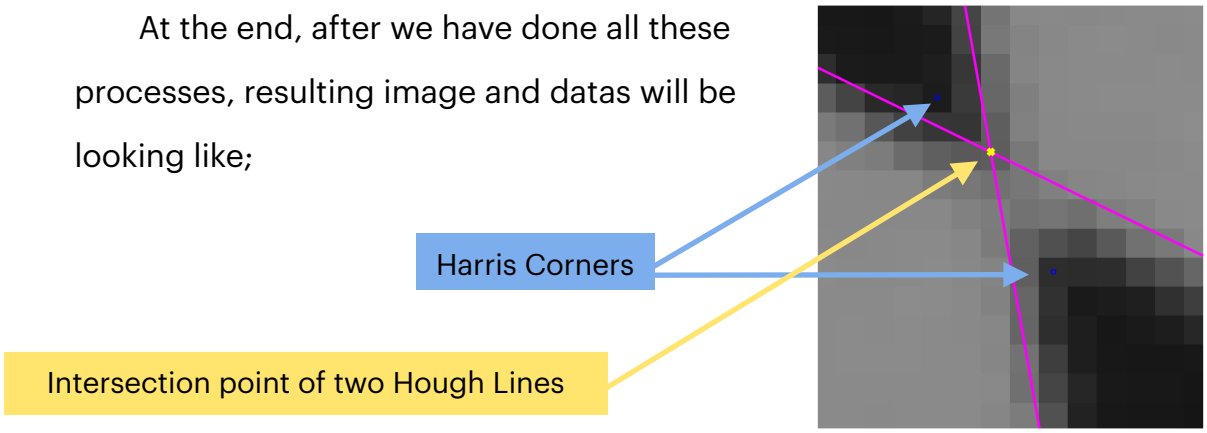
Although the ***Harris Corners*** method gave relatively well results for our purpose of determining the corners, it was not that close to the result we wanted. For example, as seen with the red dots in the example on the left,



the results obtained with the Harris algorithm are not satisfactory enough even if they are close to the real results. For this reason, this week we will work on a different method of finding corner points and compare the error amount with Harris Corner's result at the end.

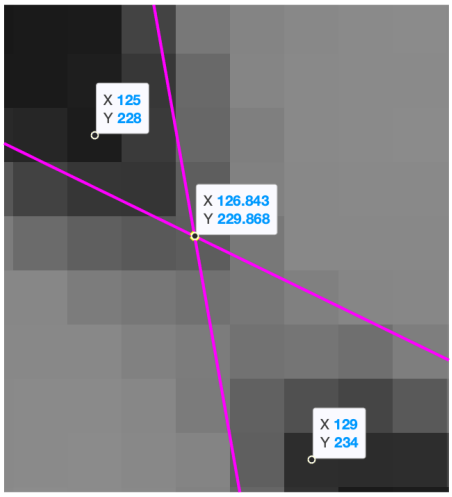
The method we will use in this laboratory will be an advanced version of the ***'Hough Lines'*** method that we have been working on in the past weeks. First, we will detect straight lines in the given image. Next, we will select 2 lines that we believe intersection of them are precise and proper enough; and try to calculate their exact intersection point by utilizing some calculations. After finding the intersection points, we will compare them with the Harris corners we have obtained before and calculate the margin of error. We should also keep in mind that all these studies will be done this week are a preliminary study for the camera calibration methods that we will be working on in the coming weeks.

At the end, after we have done all these processes, resulting image and datas will be looking like;



Now, I will write a program called ***"lab5calibprep.m"*** that detects corner points with two different methods that is explained above in detail. Namely;

1. Harris Corners
2. Intersection point of two Hough Lines



Note that, one of the reasons that *Intersection point of two Hough lines* method gives a better accuracy and proper corner than *Harris Corner Algorithm* is that Harris Corner method gives integer pixel accuracy which is less sensitive in terms of finding the exact corner point when intersection point of two Hough Lines method gives sub-pixel accuracy which is way better better in terms of detecting exact corners. In other words, detected Harris Corners are at the center of a pixel all the time. However, intersection points of two hough lines are not necessarily at the center of a pixel. On the right, one example of Harris Corners {(125,228), (129,234)} and intersection of two Hough lines (126.843, 229.868) is given. Here, you can see that Harris corners obtained by Harris Algorithm are in integer format where the intersection point obtained by intersection point of two Hough lines is in double format.

STEP BY STEP IMPLEMENTATION

```
1   img = imread("myCalibObject.jpeg");
2
3   %convert to gray-scale
4   [~,~,ch] = size(img);
5   if (ch==3)
6       img = rgb2gray(img);
7   end
8
9   img_edges = edge(img, "LoG");
10
11  %Hough Transform
12  [H,theta,rho] = hough(img_edges);
13  P = houghpeaks(H,50, 'Threshold',0.5*max(H(:)));
14  houg_lines = houghlines(img_edges, theta, rho, P, 'FillGap',20,'MinLength',60);
15
16  figure;
17  subplot(1,3,1), imshow(img_edges), title("original grayscaled");
18  subplot(1,3,2), imshow(img), title("hough lines");
19  hold on;
20
21  for k = 1:length(houg_lines)
22      xy = [houg_lines(k).point1; houg_lines(k).point2]; %beginning and ending coordinates of lines.
23      plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
24      plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
25      plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');
26  end
27
```

Here, I start by reading input image of mine in the first line. Namely, I used 'myCalibObject' image as input and worked on it entire lab. Original version of the input image can be found at the appendix.

————————GRAYSCALING————————

After that, I have converted my input image to gray-scaled version of it. First, I gathered the channel number information from line#4 and used that information to check whether my image is rgb-scaled or not. If it was, I converted the image to gray-scaled version by utilizing **rgb2gray()** MATLAB built-in function.

————————EDGE DETECTION————————

Line#9 corresponds to the operation for detecting the edges of gray-scaled input image. Here, I have used "***LoG***" edge detection option which is referring to Laplacian of Gaussian operation.

————————HOUGH TRANSFORM————————

I have used **hough()** built-in function in order to get Hough Matrix (*H*), theta(*theta*) and rho(*rho*) values as an outcome. Remember that, **hough()**

computes the Standard Hough Transform (SHT) of the binary image `entered as input`. The `hough` function is designed to detect lines. The function uses the parametric representation of a line: `rho = x*cos(theta) + y*sin(theta)`. The function returns `rho`, the distance from the origin to the line along a vector perpendicular to the line, and `theta`, the angle in degrees between the x-axis and this vector. The function also returns the SHT, H, which is a parameter space matrix whose rows and columns correspond to *rho* and *theta* values respectively.

————————HOUGH PEAKS————————

At line#13, I get peak values of Hough matrix by utilizing **houghpeaks()** which identifies peaks in Hough transform. As a deeper functionality, **peaks = houghpeaks(H, numpeaks, Name, Value)** locates peaks in the Hough transform matrix, *H*, generated by the hough function. *`numpeaks`* specifies the maximum number of peaks to identify. The function returns *`peaks`* a matrix that holds the row and column coordinates of the peaks. Also, controls aspects of the operation using name-value pair arguments.
For example, I have set *numpeaks* = 50, *Name* = 'Threshold', and *Value* = 0.5*max(H(:)) which implies the half of the maximum value in the matrix H. Because of these parameters, *P* variable will hold the maximum number of 50 peak value of given Hough Matrix whose values are greater than half of the maximum value of Hough Matrix element.

————————HOUGH LINES————————

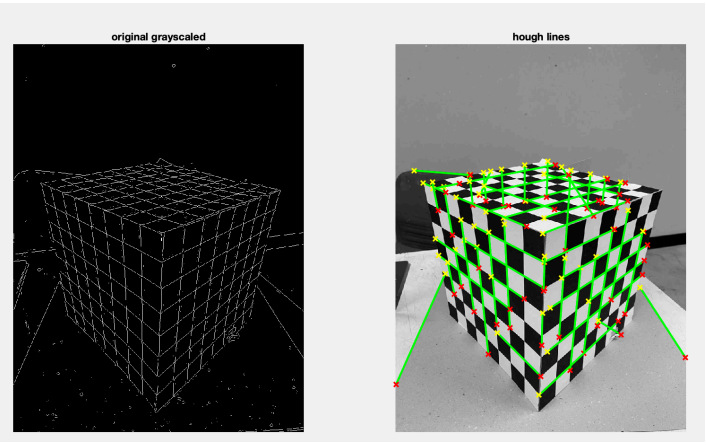
In order to extract line segments based on Hough transform, **houghlines()** built-in function is used. houg_lines = houghlines(img_edges, theta, rho, P, 'FillGap', 20, 'MinLength', 60) extracts line segments in the image *`img_edges`* associated with particular bins in a Hough transform. *`theta`* and *`rho`* are vectors returned by function *`hough`*. *P* is a matrix returned by the *`houghpeaks`* function that contains the row and

column coordinates of the Hough transform bins to use in searching for line segments. The return value *houg_lines* contains information about the extracted line segments. Also, name-value pair arguments are to control various aspects of the line extraction. ['FillGap', 20] is the name-value pair argument which are related to distance between two line segments associated with the same Hough transform bin, specified as a positive number. When the distance between the line segments is less than the value specified, (20, in our case), the *houghlines* function merges the line segments into a single line segment. Also, ['MinLength', 60] is another name-value pair argument which are related with minimum line length, specified as a positive number. *houghlines* discards lines that are shorter than the value specified.

————————PLOTTING THE LINES————————

Between line#'s 16-19, we plot the grayscale image as well as the original input image. The most important command in this pair of codes is holding on the images so that we will process some other operation on top of these images later.

Now, it is time to plot the hough lines that was extracted on previous steps. Here, we plot the beginning coordinate, end coordinate and the line in between those two points one by one for each line segment retrieved from *hough_lines* variable. Starting coordinate is yellow, end point is red, and the line in between is green colored.



What we have done so far is on the left. Note that, although lines look nasty and complicated, this result is enough for our initial purpose that will be talked about later in this report.

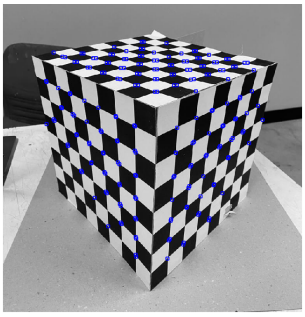
——————FIND THE HARRIS CORNERS——————

```
27
28  img_corners = corner(img, 'harris');
29  plot(img_corners(:,1), img_corners(:,2), 's', 'MarkerSize', 4, 'MarkerEdgeColor', 'blue', 'LineWidth', 1);
30
31
```

Now, we will imply the Harris Corner detection method by using **corner()** method. C = corner(I, method) detects corners in image `I` using the specified `method` and returns their coordinates in matrix `C`. In our case, img_corners = corner(img, 'harris') method detect corners in image *img* using h*arris* method and returns the coordinates in the matrix *img_corners*.

After that, we plot each corner points one by one by several settings which are related with physical attributes of each point.

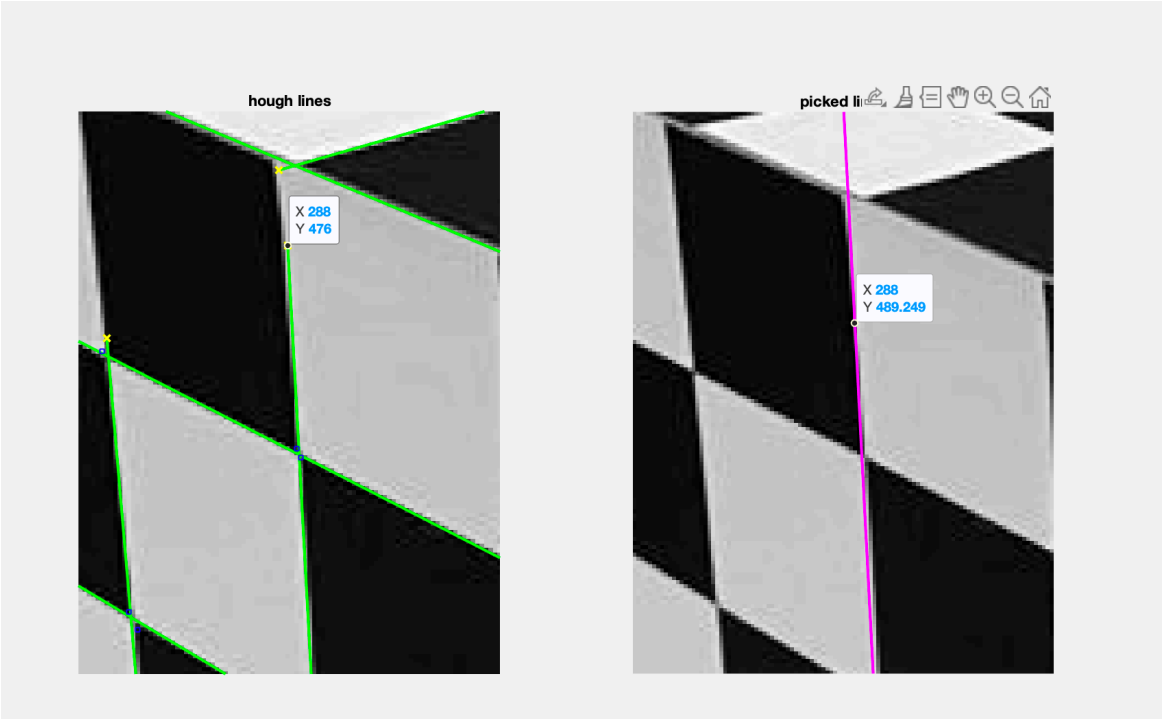
At the end, what we get is on the right. The detected corners are labeled as blue circled figures.



——EXTRACTING 8 CORNER POINTS by INTERSECTIONS OF HOUGH LINES—

Now, we will choose 8 different intersection points manually from the line segments that were plotted before.

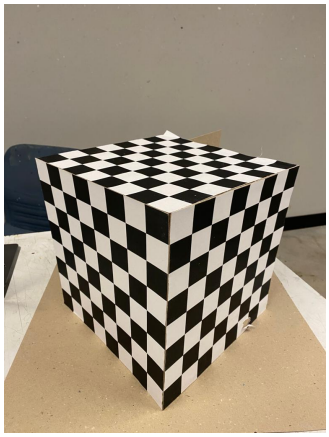
========ERROR========



The left one is the line segment extracted from hough lines. The right one is the one calculated by mathematical equation using the left one's rho and theta values. The computation part is the problematic one I believe and still cannot solve yet. Still working on it and will figure it out ASAP.

CONCLUSION

APPENDIX



THE ORIGINAL INPUT IMAGE