EE 417 Computer Vision

Berkant Deniz Aktaş

19515

Post-Laboratory Report

November 23, 2018

Sabancı University

Faculty of Engineering and Natural Sciences

Computer Science and Engineering

Introduction

In the Lab 5, we learned that Harris Corner detector is only capable of detecting pixel valued corners. This may be problematic because some corners may not be exactly integer valued and because of this the detection is not robust. An alternative method has introduced to us for corner detection. This alternative method detects corners based on the intersection of the lines that are deteceted by hough transformation. Since we are working on a polar coordinate system we can find intersection of the lines and mark them on the image.

Preprocessing

I created my own 3D calibration object with using a checkerboard image found on internet.

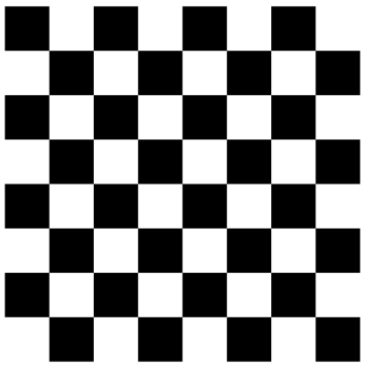


Figure 1 Checkerboard

After printing 2 checkerboards, I sticked it on the wall. When I used the sticked 3D object and tried to find corners etc. I lost a lot of time because of my camera angle. The theta parameters was coming 0, thus division of line equation was not working properly. I changed my 3D object and tried to took photo from different angle.

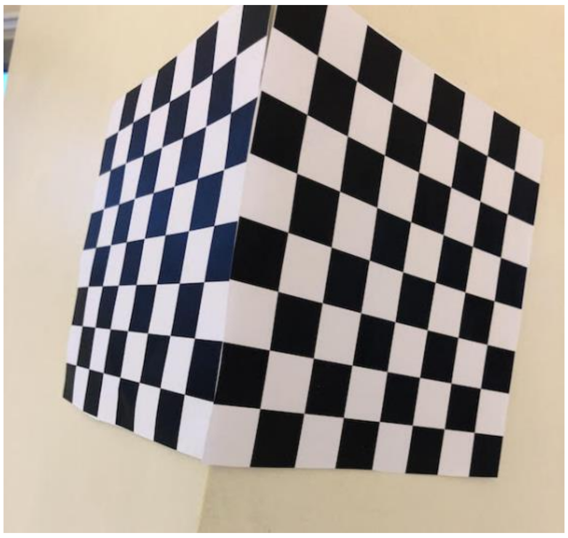


Figure 2. My calibration object

I did not tried to create a perfect 3D object and detect all lines. Rather than that, I just tried to detect 8 corners by using intersection method.

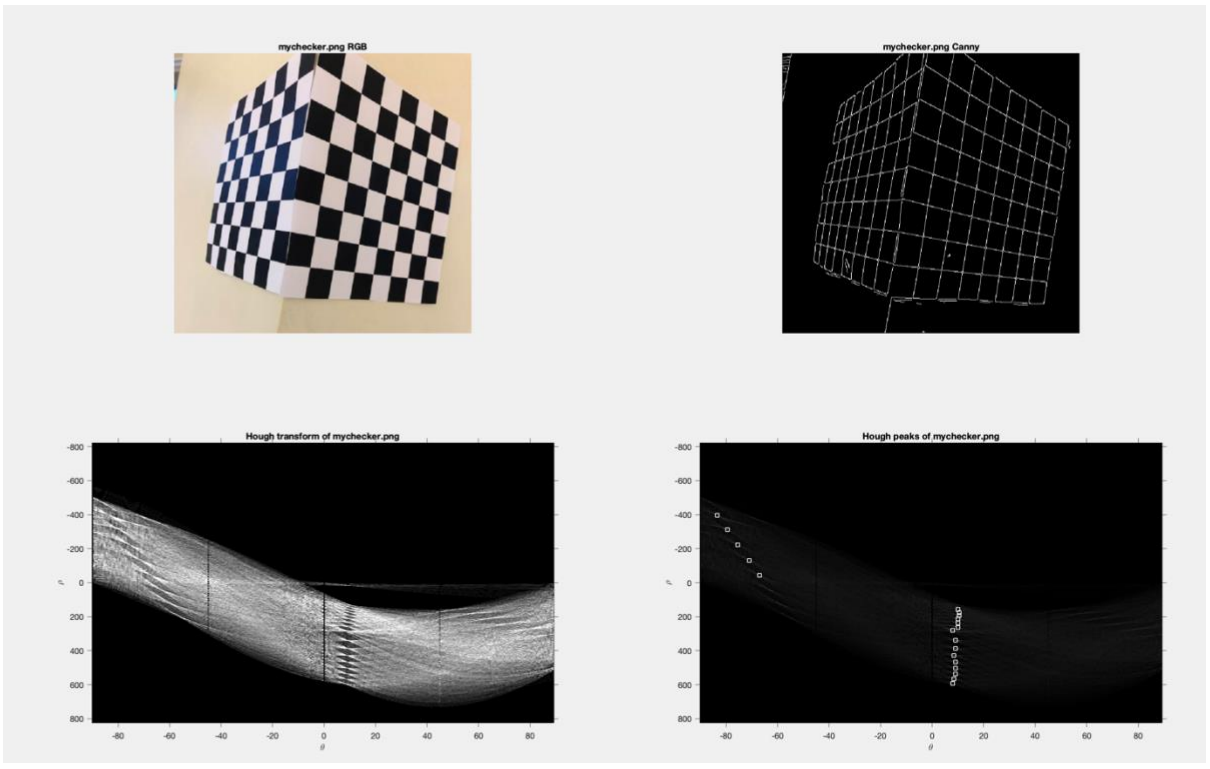


Figure 3. Line detection steps

After succesfully taking the photo and changing the resolution of the object. I used Canny filter to change image to B/W and detect lines. After that I used MATLAB’s hough transformation functions for moving parameters space and finding the peaks for line detection.

Line and Corner Detection

In the beginning the accuracy of the lines detected was really low, there were lots of undetected lines and some extra lines. I played with filters and achieved best result with:

```
P  = houghpeaks(H,20, 'Threshold', 0.01 );
Canny with MATLAB’s built-in parameters
lines = houghlines(img,T,R,P,'FillGap',10,'MinLength',40);
```

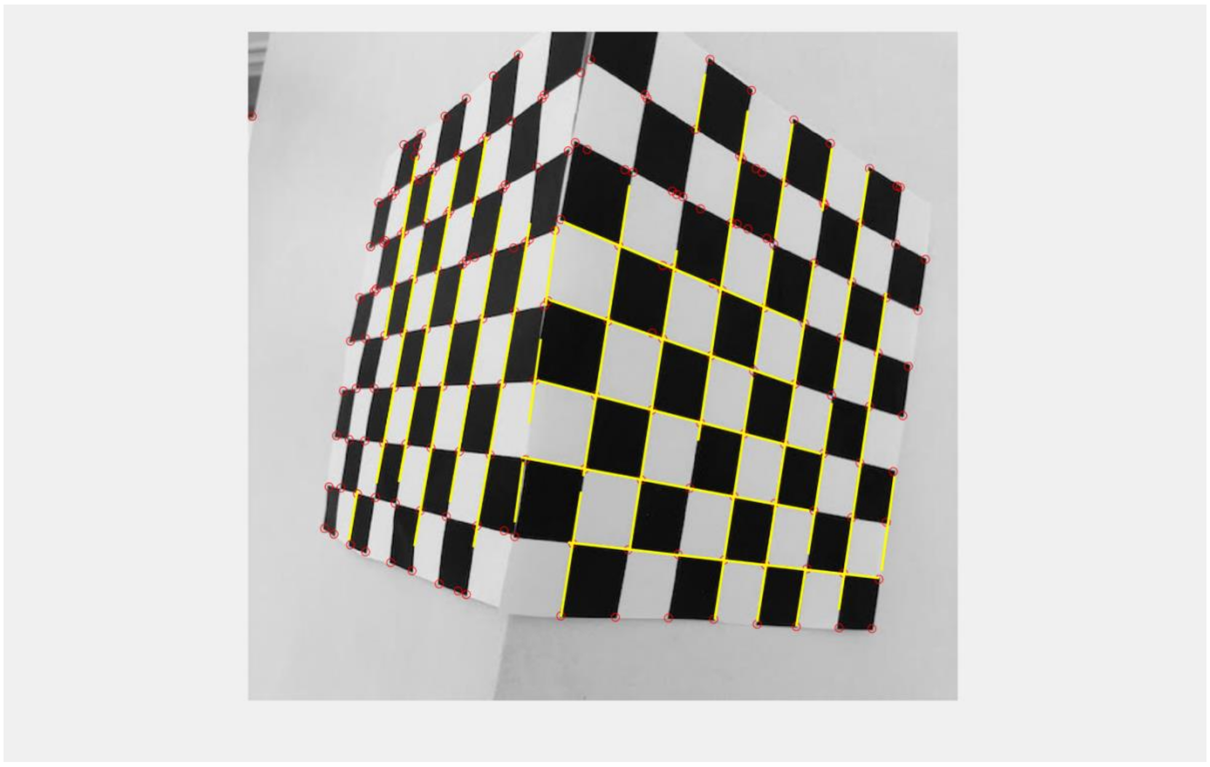


Figure 4. Lines and Corners

After finding lines I used Harris Corner detector to detect corners. Figure 4 shows detected lines in yellow color and corners in red color. One could use better created image and different parameters to achieve more robust solution for detecting lines and corners.

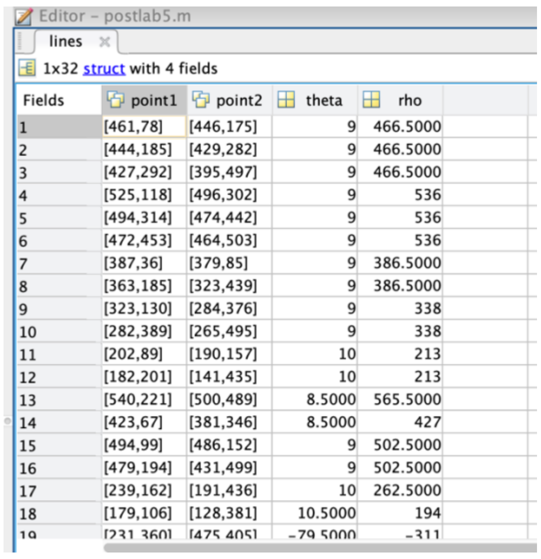
Alternative Corner Detection

As its stated before problem with Harris is non-integer valued corners. In this part I will try to explain how to find them and compare with Harris Corner Detector.

We can describe a line equation with y = mx + n. In parameter space we have n and m for our axis but it s a problem because m can be infinity. So, idea is using a polar coordinate system. In polar coordinate system a line can be defined as:

$$\mathrm{x}\cos(\theta) + y\sin(\theta) = \rho$$

If we move parameter space the axes will be θ (theta) and ρ (rho). After finding the lines with using hough transformation. Houghlines function returns the start, end, theta and rho values for us.


Editor – postlab5.m

lines

1x32 struct with 4 fields

Fields	point1	point2	theta	rho
1	[461,78]	[446,175]	9	466.5000
2	[444,185]	[429,282]	9	466.5000
3	[427,292]	[395,497]	9	466.5000
4	[525,118]	[496,302]	9	536
5	[494,314]	[474,442]	9	536
6	[472,453]	[464,503]	9	536
7	[387,36]	[379,85]	9	386.5000
8	[363,185]	[323,439]	9	386.5000
9	[323,130]	[284,376]	9	338
10	[282,389]	[265,495]	9	338
11	[202,89]	[190,157]	10	213
12	[182,201]	[141,435]	10	213
13	[540,221]	[500,489]	8.5000	565.5000
14	[423,67]	[381,346]	8.5000	427
15	[494,99]	[486,152]	9	502.5000
16	[479,194]	[431,499]	9	502.5000
17	[239,162]	[191,436]	10	262.5000
18	[179,106]	[128,381]	10.5000	194
19	[231,360]	[475,405]	-79.5000	-311

Figure 5. start, end theta and rho values of lines.

Figure 5 shows the values in lines. Now we need to look Figure 4. and find the intersecting yellow lines. In my example I used the lines with index 2, 26, 15, 9, 8, 16 for detecting intersection points. As I stated before my previous attempt was failure because after extracting everything without looking theta value, I failed to detect corners properly because some lines had theta equals to 0.

```
theta2 = lines(2).theta;
rho2 = lines(2).rho;
theta26 = lines(26).theta;
rho26 = lines(26).rho;
theta14 = lines(14).theta;
rho14 = lines(14).rho;
theta9 = lines(9).theta;
rho9 = lines(9).rho;
theta8 = lines(8).theta;
rho8 = lines(8).rho;
theta16 = lines(16).theta;
rho16 = lines(16).rho;
theta25 = lines(25).theta;
rho25 = lines(25).rho;
```

After gathering the values, we can find the formula of point y as follows:

$$y = \frac{\left(\rho - xcos(\theta)\right)}{\sin(\theta)}$$

```
y2 = (rho2-x2*cosd(theta2))/sind(theta2);
y26 = (rho26-x26*cosd(theta26))/sind(theta26);
y14 = (rho14-x14*cosd(theta14))/sind(theta14);
y9 = (rho9-x9*cosd(theta9))/sind(theta9);
y8 = (rho8-x8*cosd(theta8))/sind(theta8);
y16 = (rho16-x16*cosd(theta16))/sind(theta16);
y25 = (rho25-x25*cosd(theta25))/sind(theta25);
```

And we can find intersection of two lines with:

$$[x, y] = [\begin{matrix}\cos(\theta 2) & \sin(\theta 1) \\ \cos(\theta 2) & \sin(\theta 2)\end{matrix}]^{-1} * [\begin{matrix}\rho 1 \\ \rho 2\end{matrix}]$$

After finding all intersection points we can mark the corners on the image with corresponding intersected lines.

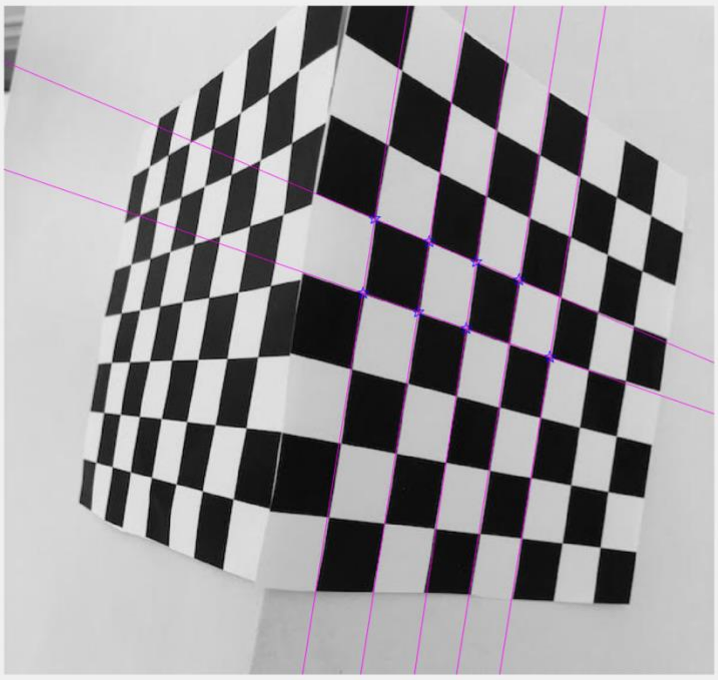


Figure 6. Intersection of lines as corners

Discussion

We can see that the corners that Harris Detector detected are different than intersection of line corners.

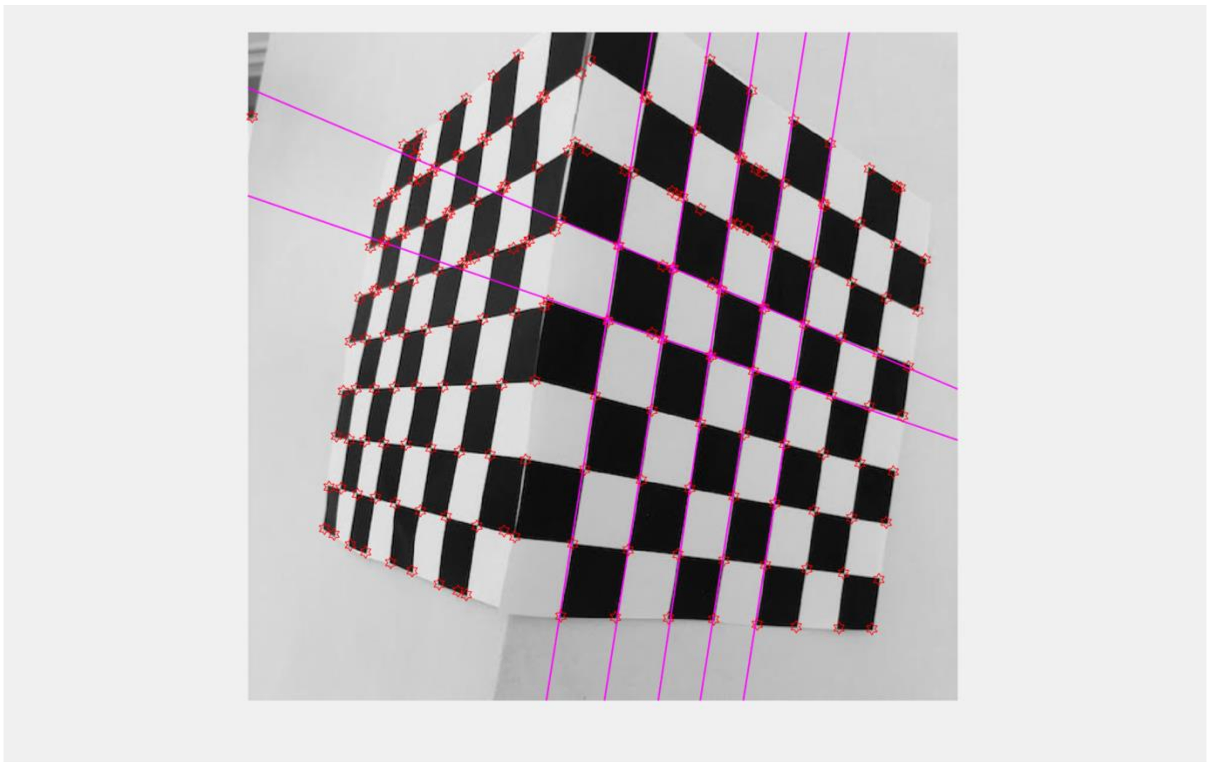


Figure 7. All in one picture.

If we zoom on Figure 7 we can see the difference of corners detected.

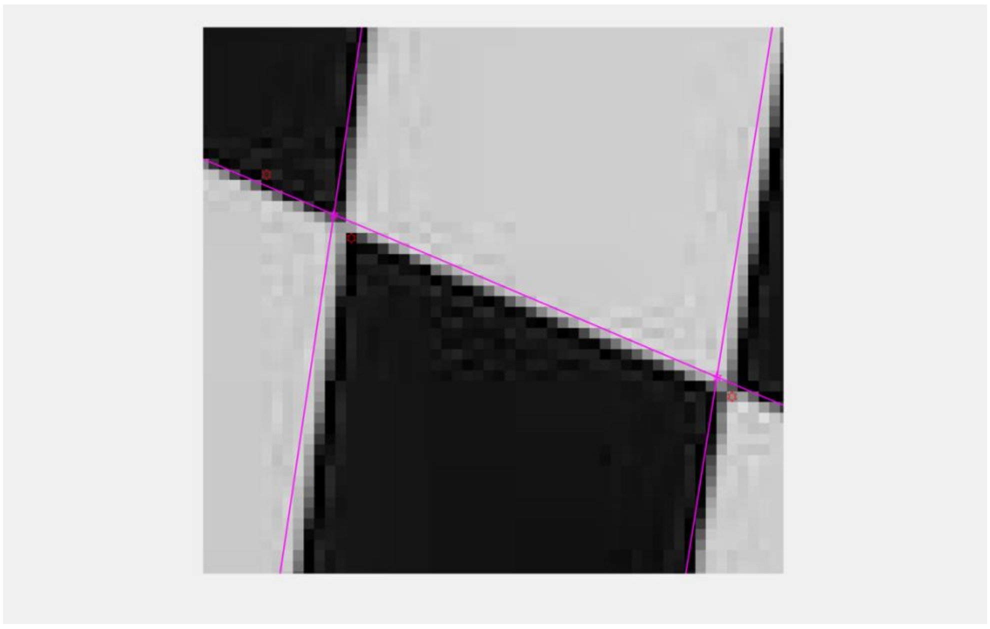


Figure 8. Zoom of Figure 7

We can see the differene on Figure 8. the intersection and the detected corners are not the same and there is a distance between them. According to my opinion it will be better to use alternative method if the lines are easily detected. Imagine a automated system capturing frames from a video every second. It will be tough to detect every line for every frame. If edge is not detected it means no corners detected for alternative method. So Harris detector works better on that case. On the other hand, Image we have got a one picture and we are doing image processing for detecting corners. If we can detect all lines without problem, trying lots of parameters, we can use alternative method instead of Harris because the results will be more realistic. For calibration, I would suggest Harris. We can try Kanade-Tomasi corner detection algorithm and compare it, maybe it can give better results in camara calibration.

Codes used in this post-lab:

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

imgGrey = img;
img = edge(img, 'Canny');

imgEdges = img;

[H,T,R] = hough(img,'RhoResolution',0.5,'Theta',-90:0.5:89);
%[H,T,R] = hough(img);
 P  = houghpeaks(H,20, 'Threshold', 0.01 );

subplot(2,2,1);
imshow(imgRGB);
title('mychecker.png RGB');
subplot(2,2,2);
imshow(imgEdges);
title('mychecker.png Canny');
subplot(2,2,3);
imshow(imadjust(rescale(H)),'XData',T,'YData',R,...
      'InitialMagnification','fit');
title('Hough transform of mychecker.png');
```

```
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(gca,gray);
subplot(2,2,4);
imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
title('Hough peaks of mychecker.png');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color','white');


pause(1);
lines = houghlines(img,T,R,P,'FillGap',10,'MinLength',40);
figure, imshow(imgGrey), hold on



theta2 = lines(2).theta;
rho2 = lines(2).rho;
theta26 = lines(26).theta;
rho26 = lines(26).rho;
theta14 = lines(14).theta;
rho14 = lines(14).rho;
theta9 = lines(9).theta;
rho9 = lines(9).rho;
theta8 = lines(8).theta;
rho8 = lines(8).rho;
theta16 = lines(16).theta;
rho16 = lines(16).rho;
theta25 = lines(25).theta;
rho25 = lines(25).rho;


 x2 = 0:0.1:750;
 x26 = 0:0.1:750;
 x14 = 0:0.1:750;
 x9 = 0:0.1:750;
 x8 = 0:0.1:750;
 x16 = 0:0.1:750;
 x25 = 0:0.1:750;

 y2 = (rho2-x2*cosd(theta2))/sind(theta2);
 y26 = (rho26-x26*cosd(theta26))/sind(theta26);
 y14 = (rho14-x14*cosd(theta14))/sind(theta14);
 y9 = (rho9-x9*cosd(theta9))/sind(theta9);
 y8 = (rho8-x8*cosd(theta8))/sind(theta8);
 y16 = (rho16-x16*cosd(theta16))/sind(theta16);
 y25 = (rho25-x25*cosd(theta25))/sind(theta25);

 A1 = [ cosd(theta26) sind(theta26); cosd(theta2) sind(theta2)];
 RhoVector1 = [rho26 ; rho2];

 A2 = [ cosd(theta26) sind(theta26); cosd(theta14) sind(theta14)];
 RhoVector2 = [rho26 ; rho14];

 A3 = [ cosd(theta26) sind(theta26); cosd(theta9) sind(theta9)];
 RhoVector3 = [rho26 ; rho9];

 A4 = [ cosd(theta26) sind(theta26); cosd(theta8) sind(theta8)];
 RhoVector4 = [rho26 ; rho8];

 A5 = [ cosd(theta25) sind(theta25); cosd(theta16) sind(theta16)];
 RhoVector5 = [rho25 ; rho16];

 A6 =  [ cosd(theta25) sind(theta25); cosd(theta14) sind(theta14)];
 RhoVector6 = [rho25 ; rho14];

 A7 = [ cosd(theta25) sind(theta25); cosd(theta9) sind(theta9)];
 RhoVector7 = [rho25 ; rho9];

 A8 = [ cosd(theta25) sind(theta25); cosd(theta8) sind(theta8)];
 RhoVector8 = [rho25 ; rho8];



pts1 = inv(A1)*RhoVector1;
pts2 = inv(A2)*RhoVector2;
pts3 = inv(A3)*RhoVector3;
pts4 = inv(A4)*RhoVector4;
pts5 = inv(A5)*RhoVector5;
pts6 = inv(A6)*RhoVector6;
pts7 = inv(A7)*RhoVector7;
pts8 = inv(A8)*RhoVector8;

 hold on;
 plot(pts1(1),pts1(2),'mp','MarkerSize',15);
 plot(pts2(1),pts2(2),'mp','MarkerSize',15);
 plot(pts3(1),pts3(2),'mp','MarkerSize',15);
 plot(pts4(1),pts4(2),'mp','MarkerSize',15);
 plot(pts5(1),pts5(2),'mp','MarkerSize',15);
 plot(pts6(1),pts6(2),'mp','MarkerSize',15);
 plot(pts7(1),pts7(2),'mp','MarkerSize',15);
 plot(pts8(1),pts8(2),'mp','MarkerSize',15);


plot(x2,y2,'LineWidth',2,'Color','m');
 plot(x26,y26,'LineWidth',2,'Color','m');
 plot(x14,y14,'LineWidth',2,'Color','m');
   plot(x9,y9,'LineWidth',2,'Color','m');
 plot(x8,y8,'LineWidth',2,'Color','m');
  plot(x16,y16,'LineWidth',2,'Color','m');
   plot(x25,y25,'LineWidth',2,'Color','m');



 C  =  corner(imgGrey);
  hold on;
plot(C(:,1),C(:,2),'rh', 'MarkerSize', 15)

for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
  %plot(xy(:,1),xy(:,2),'LineWidth',0.8,'Color','yellow');
   end
```