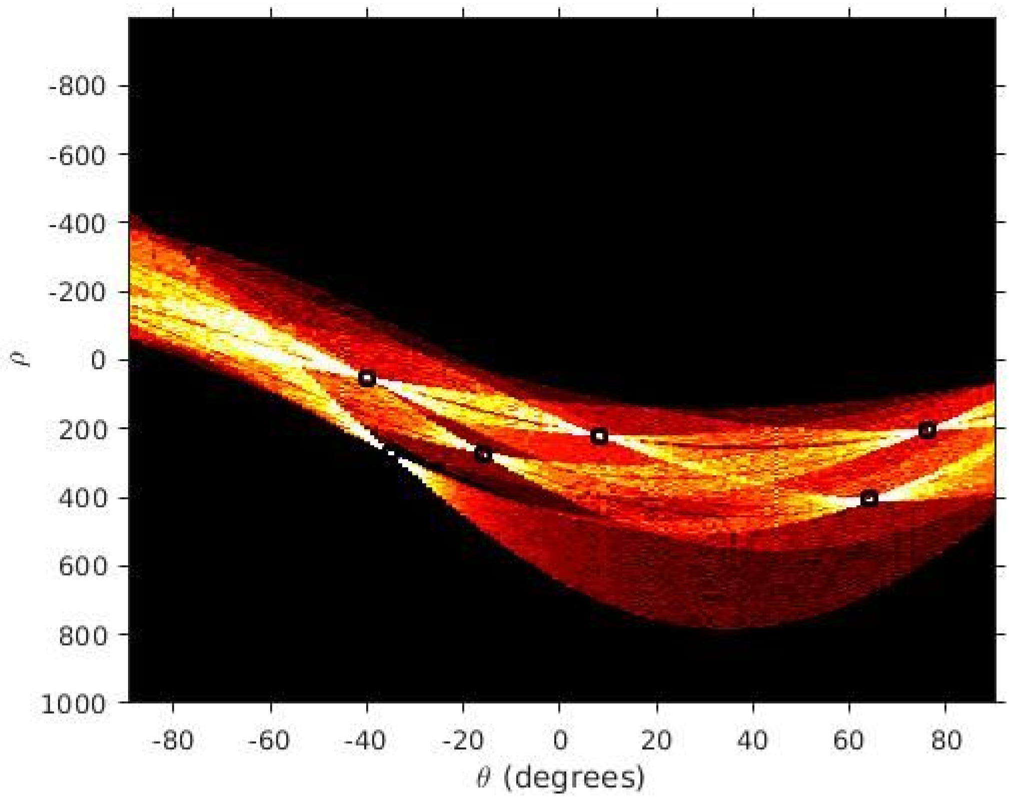
CSCI 5561: Programming Assignment 2 Report

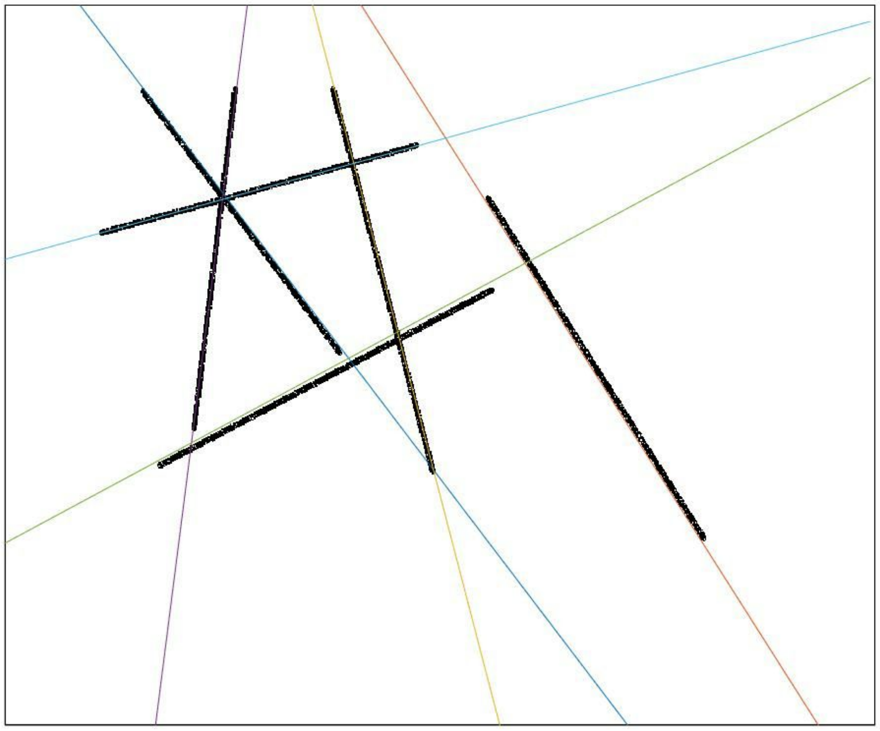
Arjun Varshney
varsh007@umn.edu

In this report, we implement Hough Transform method to detect Lines and ellipses. We use our thinned edge detected image as an input to the Hough Transform detection, which was done earlier in Assignment 1.

1) **Line Detection:**
Detecting lines uses the parametric space (m,b or rho, theta) to plot the curves and collinear points define a peaks. If there are multiple lines in an image, we should see that many peaks in the hough space. In the given programming assignment, we are given an image with multiple lines, where Hough transform uses an accumulator array of Rho and Theta parametric space (using equation of a line in polar coordinates). The result of Hough Transform for detecting lines is as shown below,

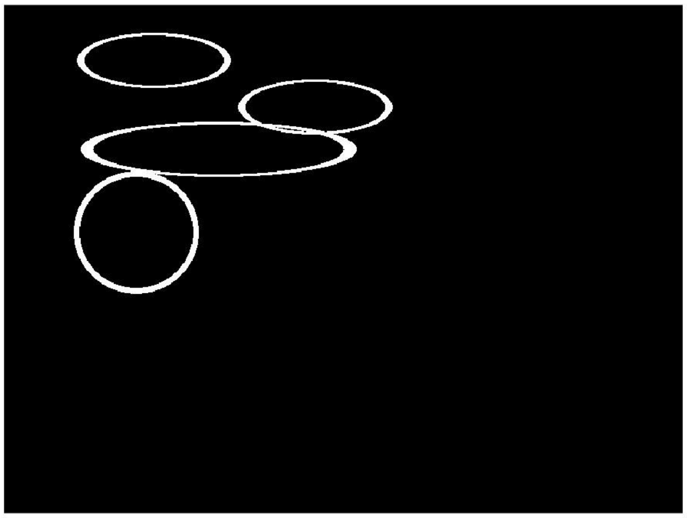


In the given figure above, we detect the peaks in the Hough Space. One limitation I saw here was to fix the size of the threshold beyond which we can call the votes as peaks. Either the user can input the no. of lines in the image beforehand or detecting threshold would be cumbersome task for various images. While matlab uses 0.5 * max(hough), I have used 0.4* max(Hough) due to the number of lines in the image to be detected. We can have different algorithms to find the threshold for that (similar to edge detection thresholding technique). Next is superimposing the detected lines on the original image, which is done using the traditional values of m and b in the polar coordinates. Figure shown below is the result of superimposing the lines on the original image.

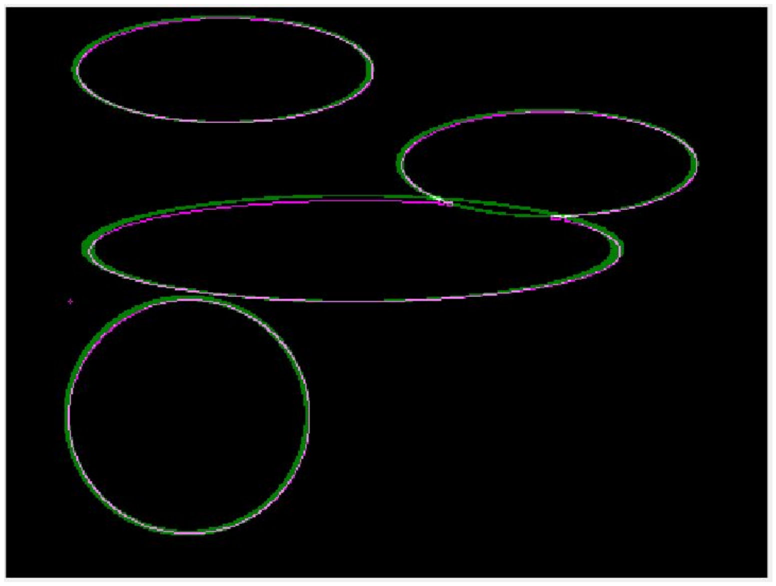


2) **Ellipse Detection:** For this, I have used the paper by Y. Xie[1], where we have the concept of using 1-dimensional array where voting is done for semi-minor axes. We use the concept of three points(each point is an edge pixel) where we calculate the semi-major axes (a) using first two points and calculate semi-minor axes value using third point. We detect ellipse for votes more than min_votes set at the beginning of the algorithm. The detected ellipses are drawn and then matched pixels are deleted in the edge pixels array to eliminate any more possible detection because we assume that we found the best ellipse.
There were several observable points while implementing this algorithm. I have used constraints in this assignment like keeping alpha =0 as we see that the ellipses' major axes are parallel to x-axis and to reduce more number of detected ellipses which are not the best. Moreover, voting for third point gives many ellipses, so we calculate the best of the best ellipses for each selected second point (x2,y2), which gives the best fit ellipse. This can be taken further to select the best-best fit ellipse if we find the ellipse for each (x1,y1) too, but that would require an input from the user saying how many ellipses are present in the image (which is an overhead and no one would want that). The figures are shown below:

[1] Yonghong Xie and Qiang Ji, *A New Efficient Ellipse Detection Method*, Intemational Conference on Pattern Recognition, pp.957-960, vol. 2002.



The figure above shows the detected ellipses. Here, when I draw I use the equation of the ellipse as x^2/a^2 + y^2/b^2 and calculate its value of the image space, with the given detected ellipses parameters (x0,y0, a, b). I have used an estimated epsilon of 0.1 here in the figure (and 0.05 too for experiments), which would broaden the ellipse to delete pixels from the edge pixels array. These detected ellipses are then superimposed on the edge-thinned image as shown below.



Zoomed image:

Things to ponder for ellipse:

1) The theta values differ by 1 in the code. If I change it to 0.5 or lesser, we see multiple lines having similar votes, due to the reason that our Hough Space is larger with minimal difference, and the edges for a single line are actually two. So microscopically, we have two lines very near to each other. So, if our Hough Space is very large, we are bound to get those lines detected. Otherwise, the lines would generally be single as we are rounding off R to some extent with

discrete values of theta. Here the threshold used is 0.4* max(hough) which is slightly lesser than default value of threshold used by MATLAB function houghpeaks to detect those lines.

2) The ellipse algorithm has many constraints. We are applying more and more constraints or conditions to reduce the ellipse space from being detected. Here in this assignment, we know that all the ellipses are parallel to x-axis (semi-major axes parallel to x-axis) so I have put a condition of detecting only those kind of ellipses to reduce the computation time. We can get similar results without having that constraint too but we have numerous ellipses with slight angles (alpha = -0.05) which also gets required minimum votes. This further makes the things complex, as for each first point we now have to calculate the best ellipse which would get maximum votes, but we don't know when the best ellipse is found and then when to delete the pixels to reduce computation. I think if gradient direction comes into play here, it would help us know if we are at the local maximum for getting the best ellipse. A good algorithm can be devised here for this.