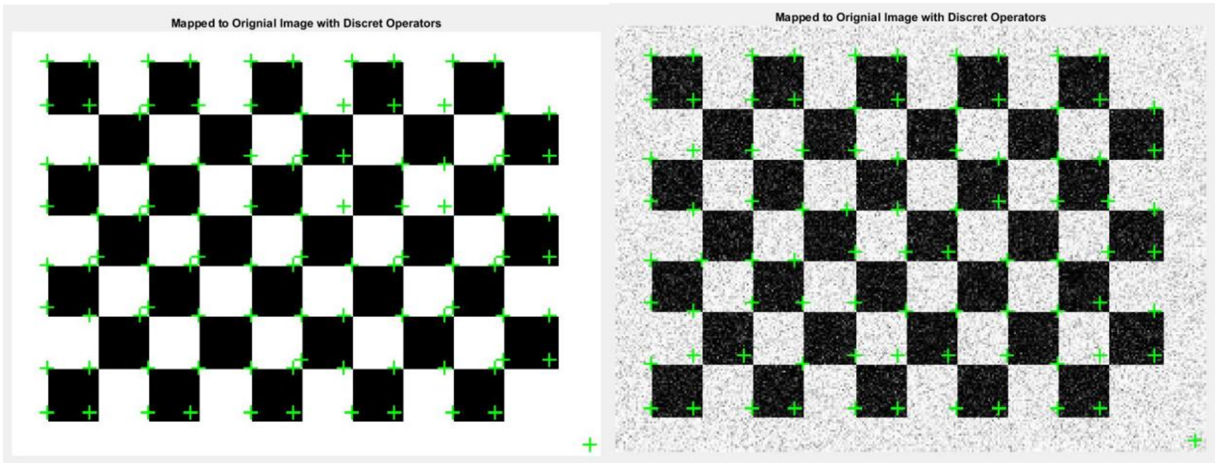
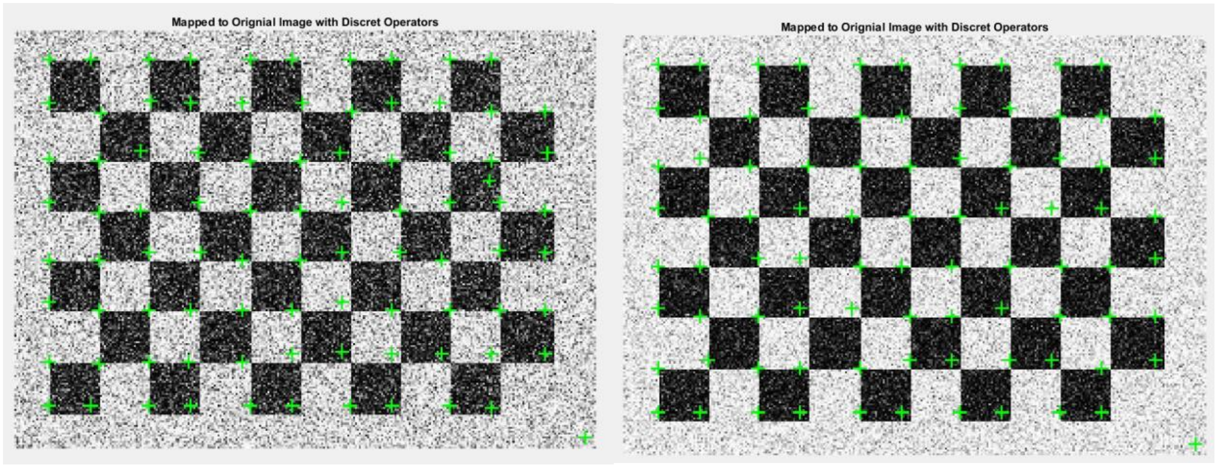
ECSE 683 Assignment 1

Liqiang Ding 260457392

Question 1. CORNER EDGE DETECTOR

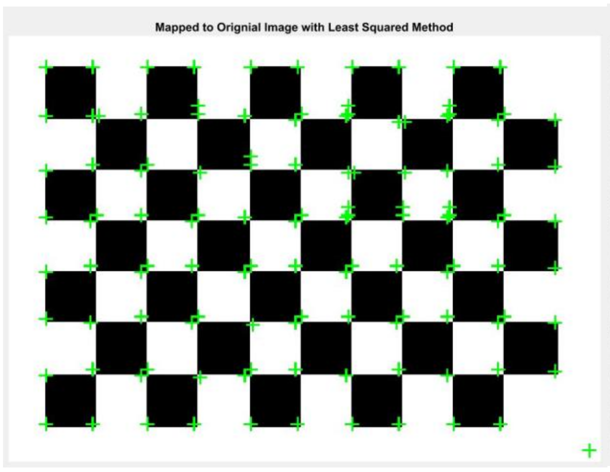
b) Implement, using Matlab or equivalent, the corner detection algorithm discussed in class. Here, build discrete operators to estimate the partial derivatives Ix and Iy. Your operators should also include some smoothing. Compare your error against ground truth for each of the 4 images in the test set.

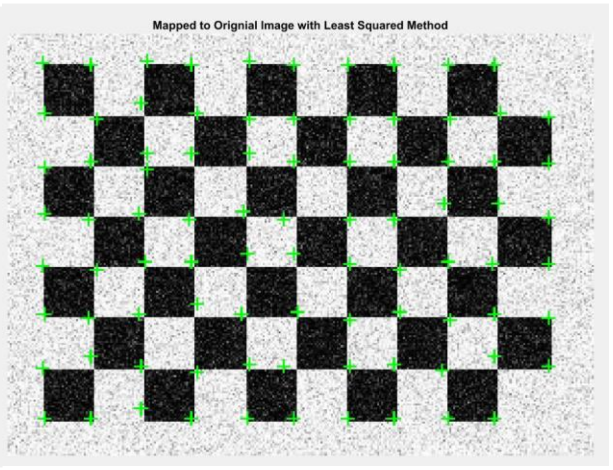

Mapped to Orignial Image with Discret Operators
Mapped to Orignial Image with Discret Operators

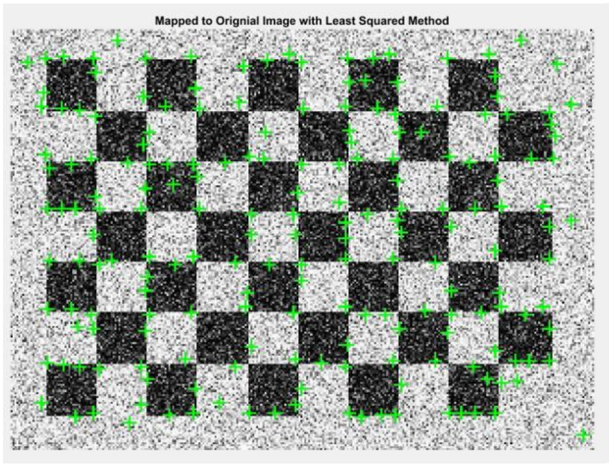

Mapped to Orignial Image with Discret Operators
Mapped to Orignial Image with Discret Operators

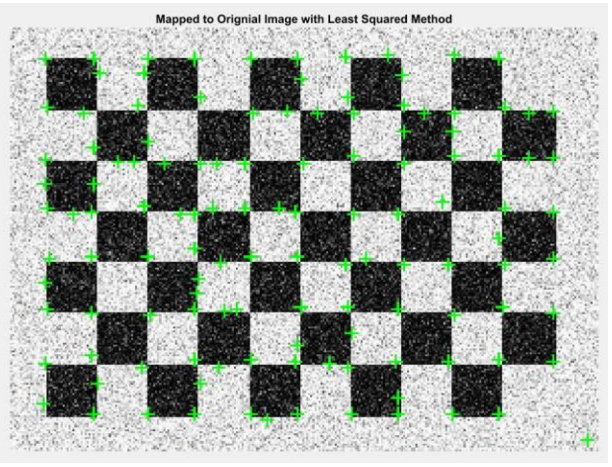
Images above are the output results from using discrete operators to estimate the gradients of the input images. The top left is with zero variance, the top right is with 5% Gaussian variance, the bottom right is with 10% Gaussian variance and bottom left is with 25% Gaussian variance. Detailed error analysis is in the summary section.

c) Repeat Part (b), but this time use least squares to obtain a parameterization of each local neighborhood using a suitable polynomial. The partial derivatives Ix and Iy now reduce to polynomial coefficients.



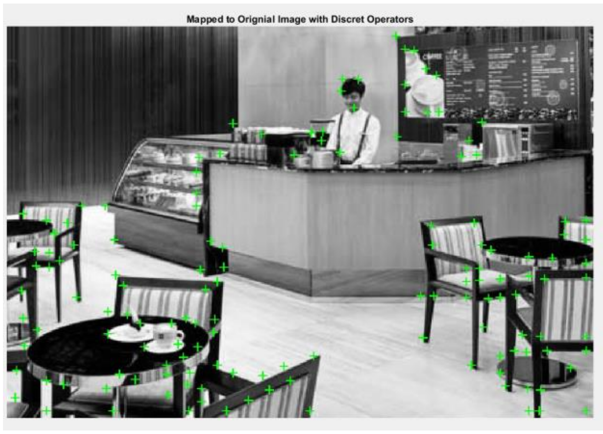


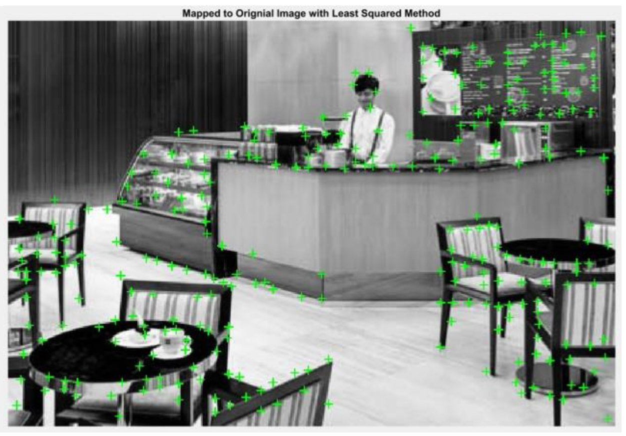




Images above are the output results from using polynomial equation to estimate the gradients of the input images. The top left is with zero variance, the top right is with 5% Gaussian variance, the bottom right is with 10% Gaussian variance and bottom left is with 25% Gaussian variance. Detailed error analysis is in the summary section.

d) Compare the performance of the two implementations on a real image, find_the_corners.jpg





Summary:

As it can be seen from results, the higher Gaussian variance, the higher amount of noise is added to the original checker board image. This makes it harder to detect any corners without changing the threshold value. In face, if we don't decrease the threshold values as more noise gets involved, the method would miss most of corners. Therefore, the actual threshold values for the discrete operator are [50, 5, 0.5, 0.5], and the actual threshold values of the polynomial equation are [1.5, 0.25, 0.1, 0.1], in order to maintain accurate recognition performance.

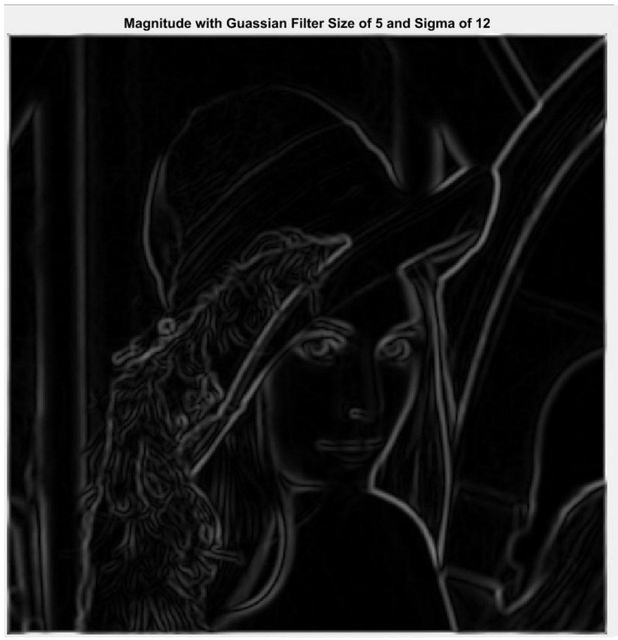
In terms of accuracy, the measured coordinates of the true corners {[43,41], [68,41], [68,41], [93,41], [143,41], [43,65], [43,91], [43,115], [43,141], [43,167], [68,66], [92,66]}. In both methods, most of green labels are inserted at the pixel around the corners instead of the exact locations of corners. This makes sense because the gradients in X and Y directions are "ill defined", whereas gradients have two different values in regions around corners. This results in labeling corners with some offsets.

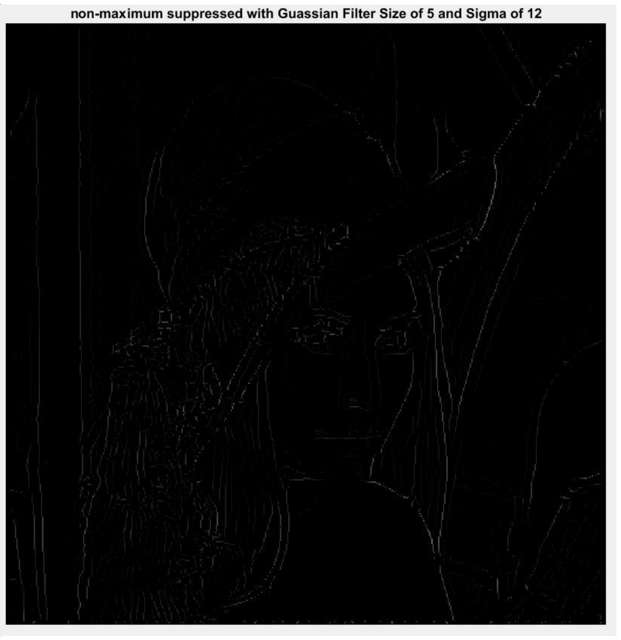
Both algorithm performs very well for correctly detecting the corners, and labeling them without missing any of corners when the variance is low. However, when the Gaussian noise variance reaches to 10% and above, the polynomial equation method starts labeling incorrectly in regions where no corners exist. On contrast, there is only one label inserted incorrectly by using discrete operator when the Gaussian variance reaches to 25%.

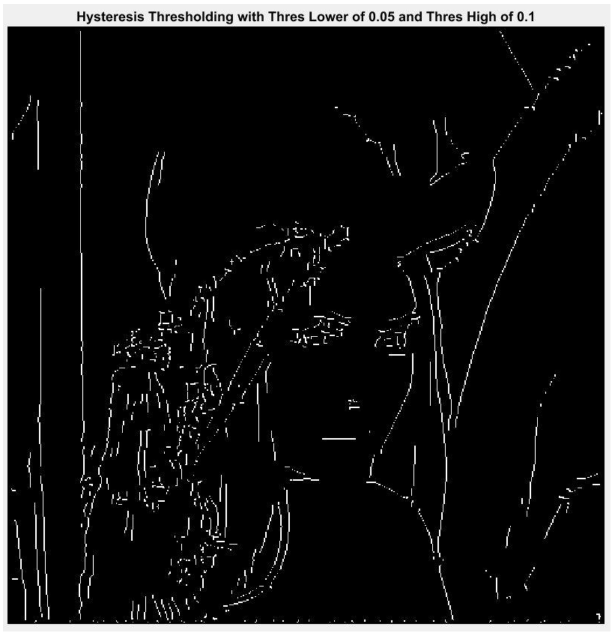
The major difference between two approaches is the way to calculate the gradients in X and Y directions of the input image. The discrete operator uses a filter size of 3 to convolute the input in both directions to get Ix and Iy. In the polynomial parameterization method, a window size of 5 is used to scan the input image to parameterize the entity values in terms of x and y. After the parameterization, a polynomial equation, I(x, y), with 6 terms can be generated. Then, equations of Ix and Iy are calculated based on the coefficients of the polynomial equation.

Question 2. CANNY EDGE DETECTOR

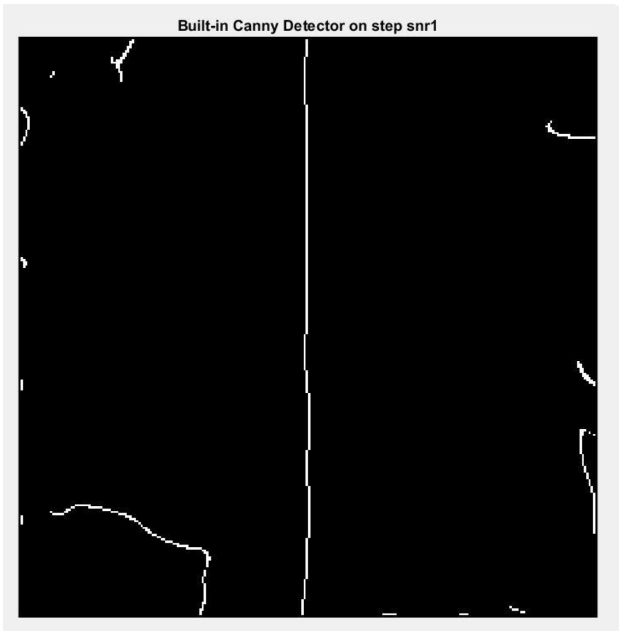
i) Run your edge detector on image lena.png. Show intermediate and final images: the norm of the image gradient, the image after non-maximum suppression, and the image after hysteresis thresholding.

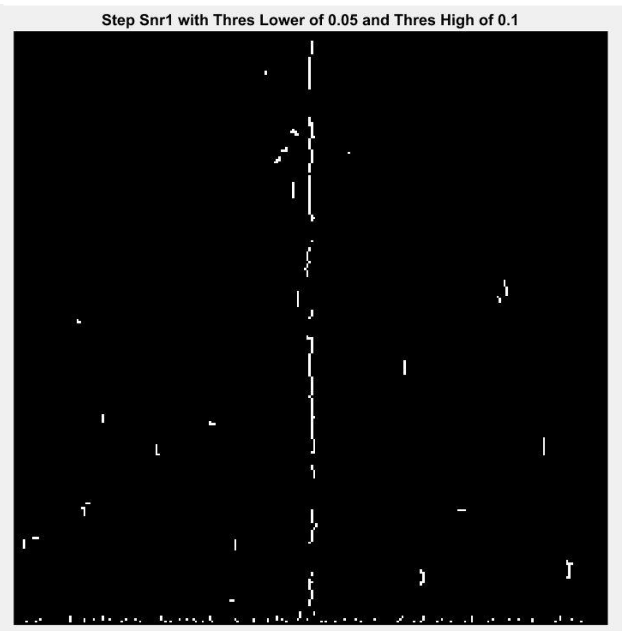


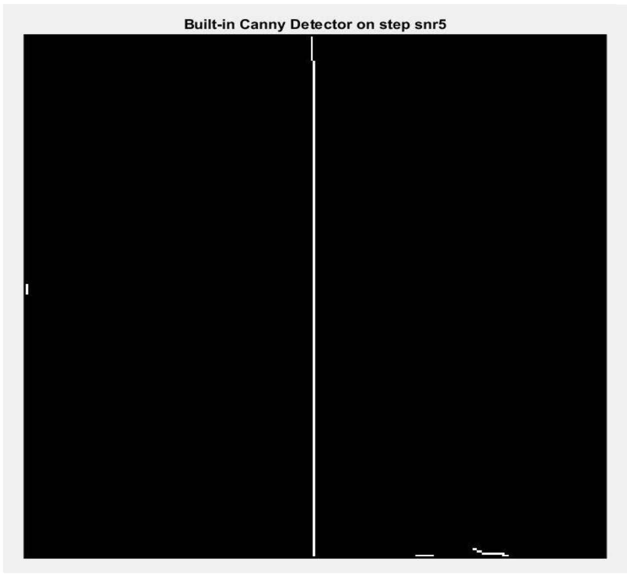

non-maximum suppressed with Guassian Filter Size of 5 and Sigma of 12

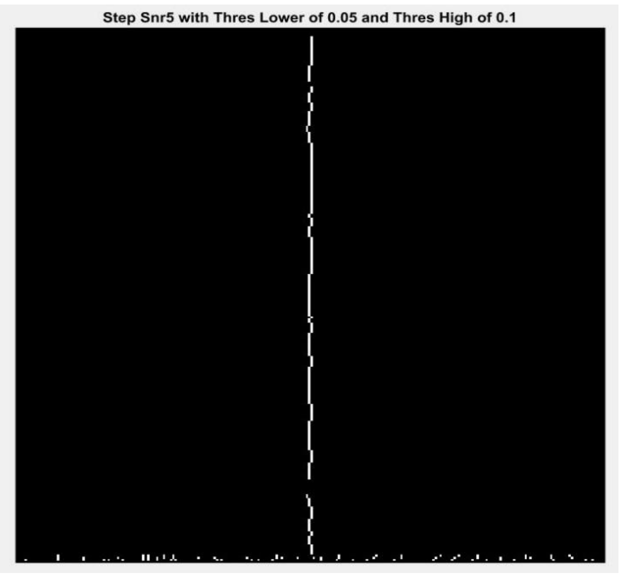

Hysteresis Thresholding with Thres Lower of 0.05 and Thres High of 0.1

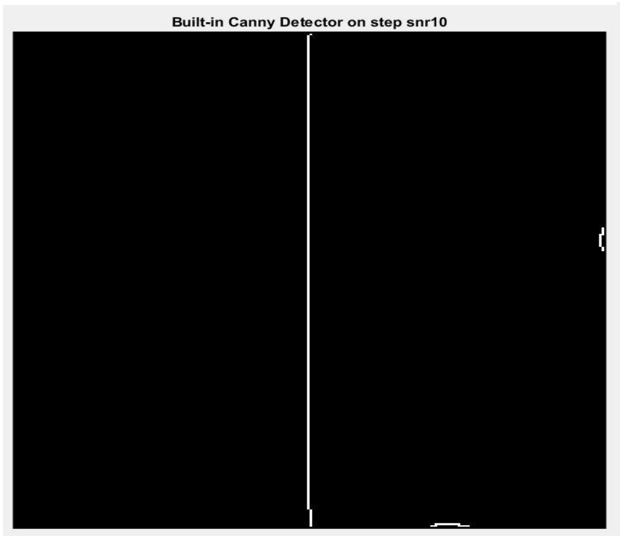
ii) Evaluate your algorithm on edge location performance using synthetic images (snr1.png through snr100.png), and compare it to the built-in Matlab function. The number of edge pixels should be the same as the image height. Good detectors should produce the most edge pixels along the central line and the fewest in other areas. For each image, present side-by-side results generated by your code and those generated by Matlab command "edge(I,’canny’,...)". Try and use parameters, which produce the best results. Provide a brief discussion of your results.

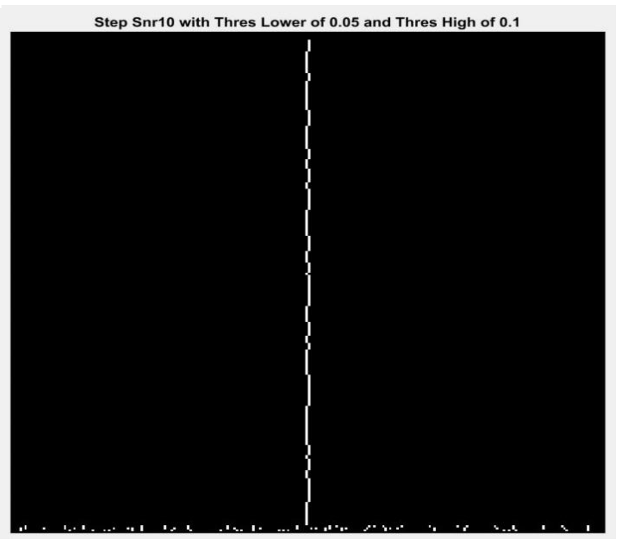

Built-in Canny Detector on step snr1

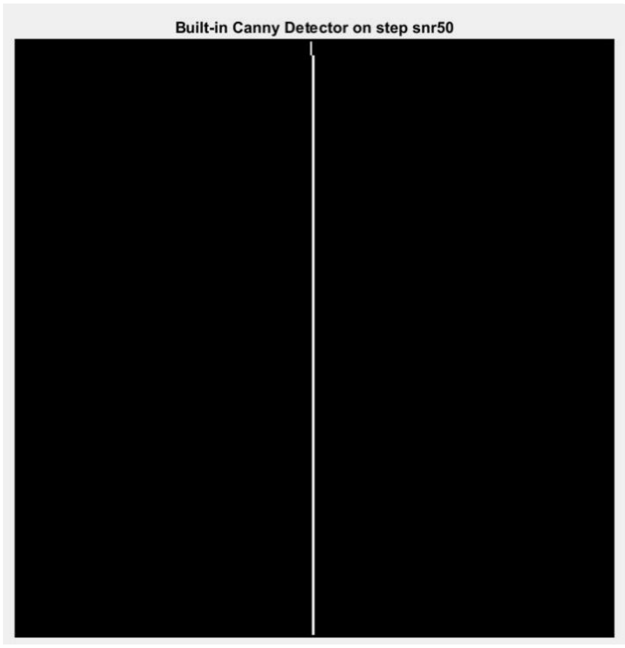

Step Snr1 with Thres Lower of 0.05 and Thres High of 0.1

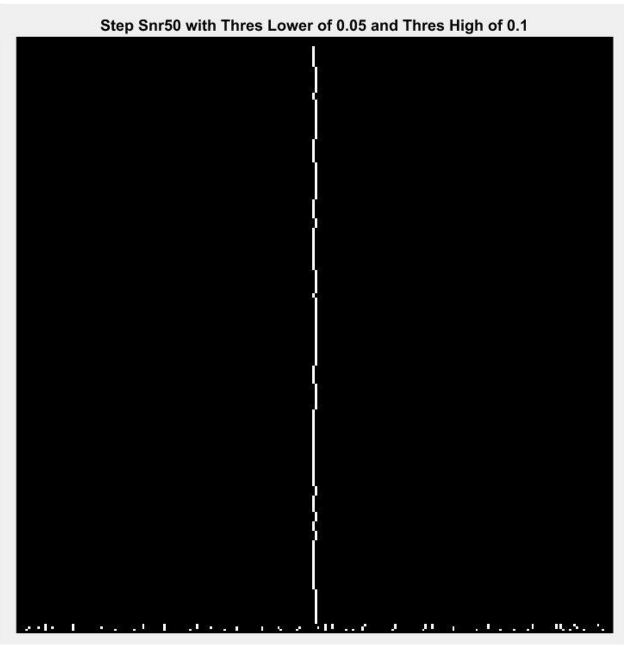

Built-in Canny Detector on step snr5

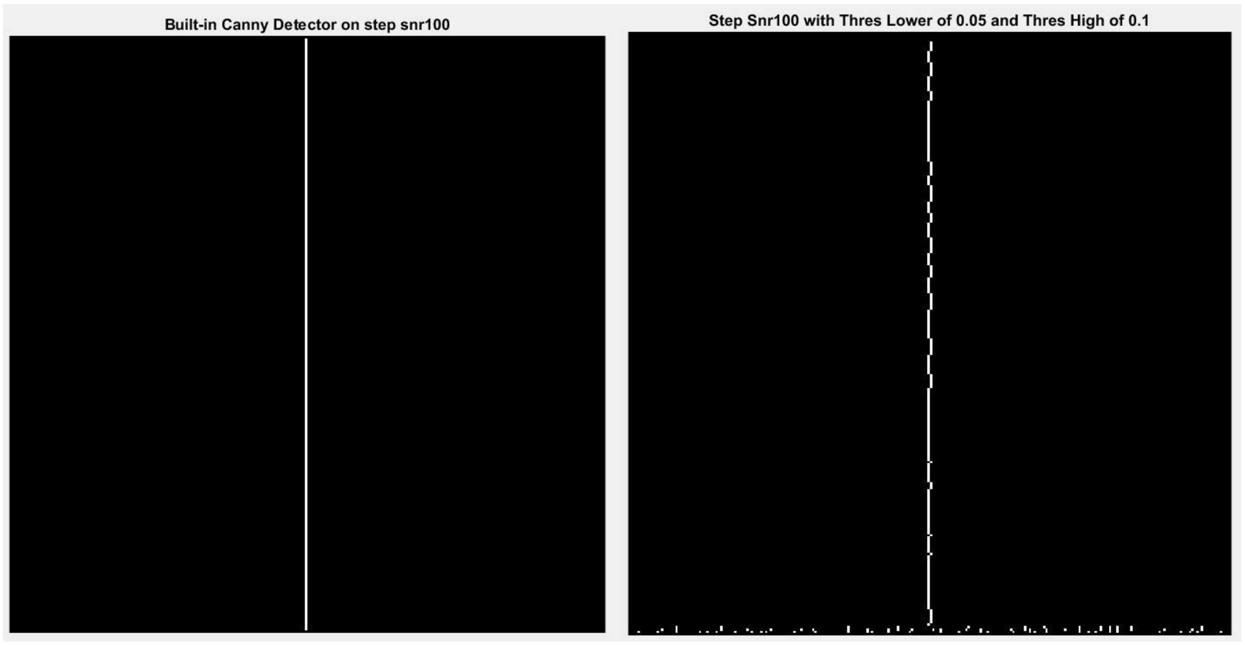

Step Snr5 with Thres Lower of 0.05 and Thres High of 0.1


Built-in Canny Detector on step snr10


Step Snr10 with Thres Lower of 0.05 and Thres High of 0.1


Built-in Canny Detector on step snr50


Step Snr50 with Thres Lower of 0.05 and Thres High of 0.1

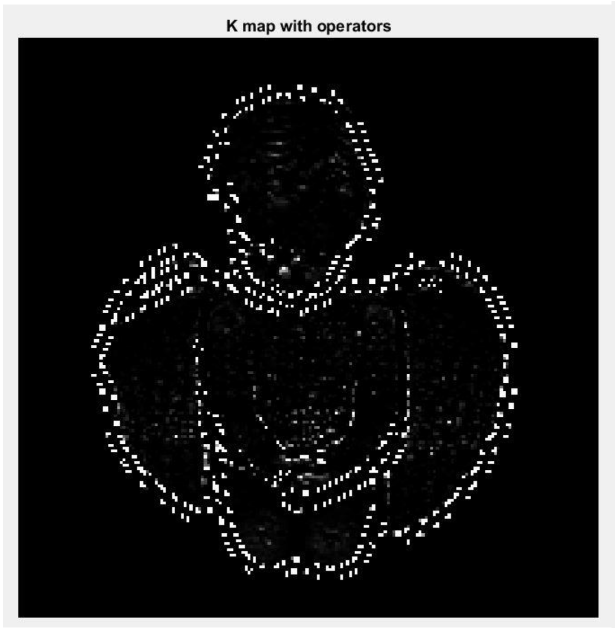

Built-in Canny Detector on step snr100
Step Snr100 with Thres Lower of 0.05 and Thres High of 0.1

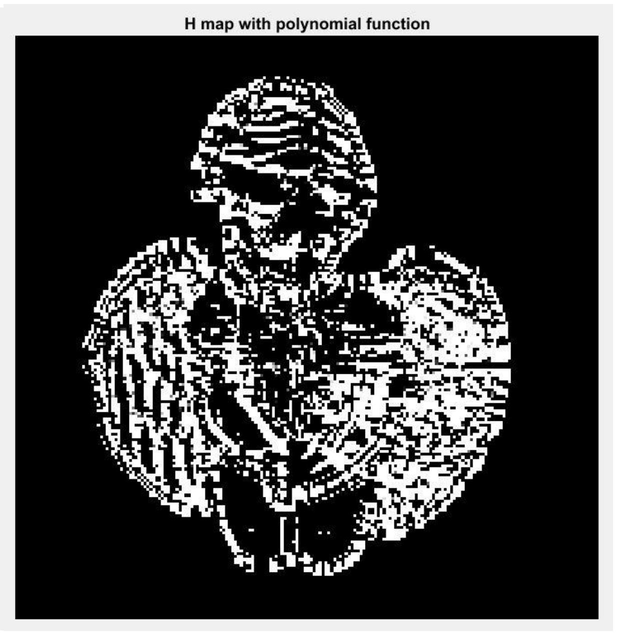
Summary:

The above images are the output results from Matlab built-in canny edge detector and manually implemented canny edge detector. Both approaches use the same set of threshold values and the sigma for Gaussian smoothing filter. Based on the result, the Matlab built-in function is able to handle the edge detection task accurately. Similarly, the manual approach is able to detect and localize the center edge accurately as well. However, when it connects every dot together, some parts of the line are not reconstructed straightly enough. In terms of the performance on the image boundaries, the built-in function performs better by successfully avoiding the small threads. On contrast, the manual approach labels some of the threads on the output image. As the variance increases, both methods get affected negatively, whereas the manual method performs worse. Based on this result, some improvements can be done on hysteresis thresholding step of the manual approach, in order to connect dots more accurately.

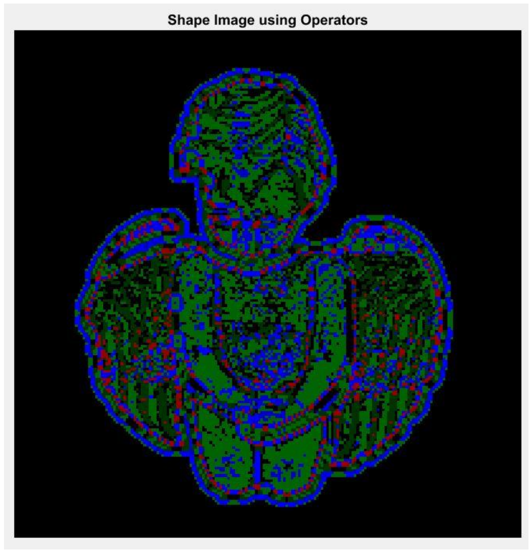
Question 3. CURVATURE RECOGNIZATION

i) Begin by implementing operators for Ix, Iy, Ixx, Iyy, and Ixy using a 2D Gaussian as a basis. The optimal value of σ will have to be inferred from the data. With these in place, compute the H-K map for the angel data set. Display your results accordingly.



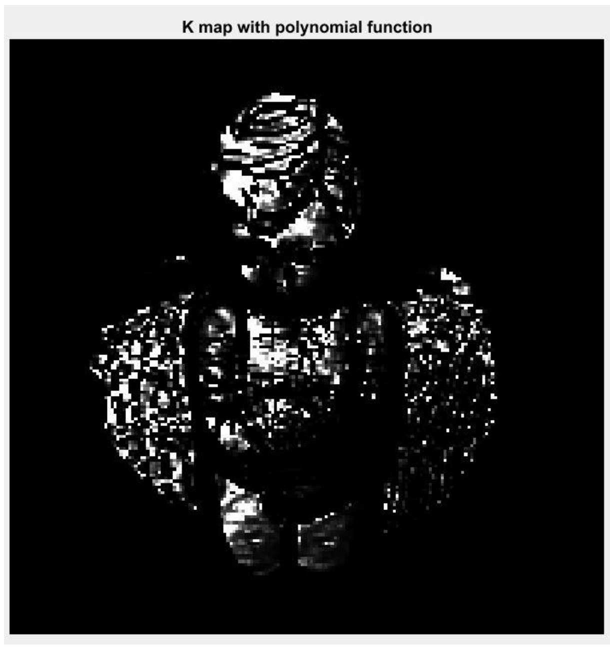


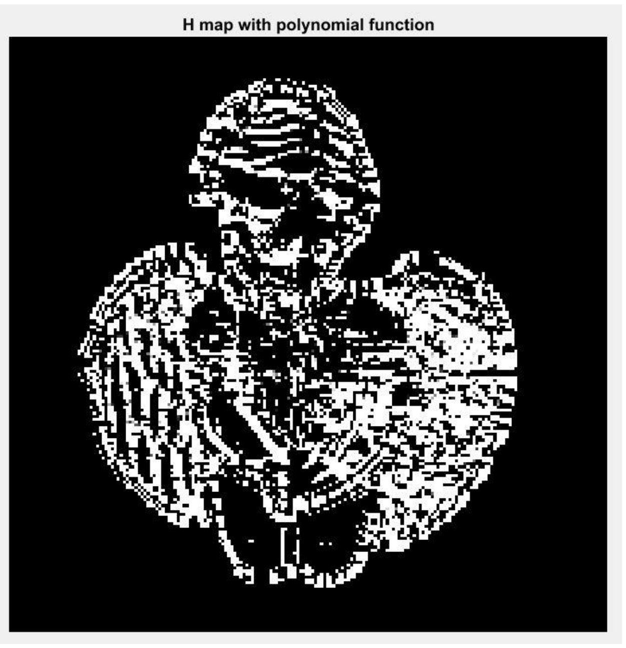
ii) Compute the shape label image, S, by assigning a shape label si to each pixel.



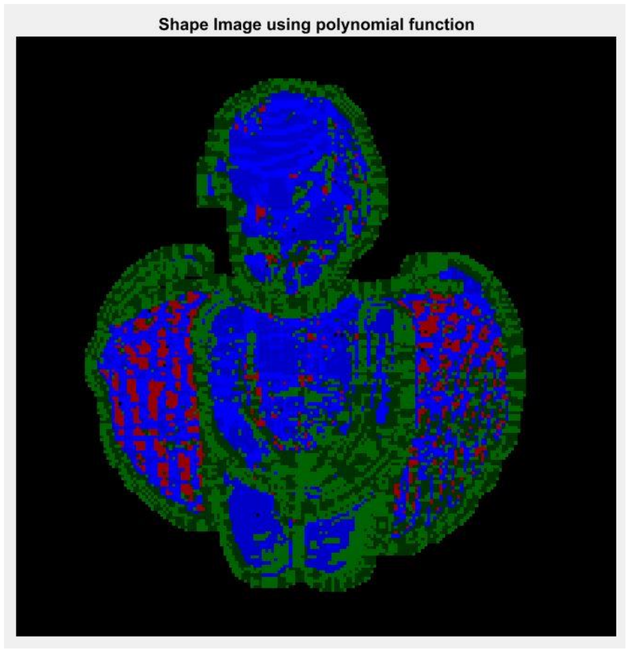
Legend: [(Black, Plane), (Dark Green, concave cylindrical), (Green, convex cylindrical), (Red, concave elliptic), (Light Blue, convex elliptic), (Blue, hyperbolic)]

iii) Now repeat Part (ii) using a least-squares approximation to estimate the values of a polynomial h(x,y) = ax2 + bxy + cy2 + dx + ey + f at each coordinate. You need to figure out the appropriate sampling neighborhood. From the polynomial coefficients, determine H and K. Generate the H-K map for the angel data set. Display your results and compare against those obtained in Part (i).





iv) Repeat Part (ii) using the H-K map obtained using least squares. Compare against the shape label image you obtained initially.



Legend: [(Black, Plane), (Dark Green, concave cylindrical), (Green, convex cylindrical), (Red, concave elliptic), (Light Blue, convex elliptic), (Blue, hyperbolic)]