Anthony Bugatto

EE 481: Image Analysis

Project 2

This project was very challenging for me due to the amount of testing I had to do to figure out how each function worked and how to make the response ideal. Additionally, I had to rewrite all of my code after realizing that my lack of testing functions and forward thought when writing the code was causing segmentation faults and my functions not to work. Overall, beyond having learned about these image analysis algorithms I believe my coding practice improved significantly.

1. In this part we created a synthetic image by starting with a white background image and adding a blue inner square of half size with a yellow square of quarter size within that one (figure 1). Using this synthetic image, I added Gaussian noise for sigma = [3.16, 4.47, 6.32, 10]. While this can be in grayscale I thought it would look better as RGB noise, which wasn't much harder to implement anyway.

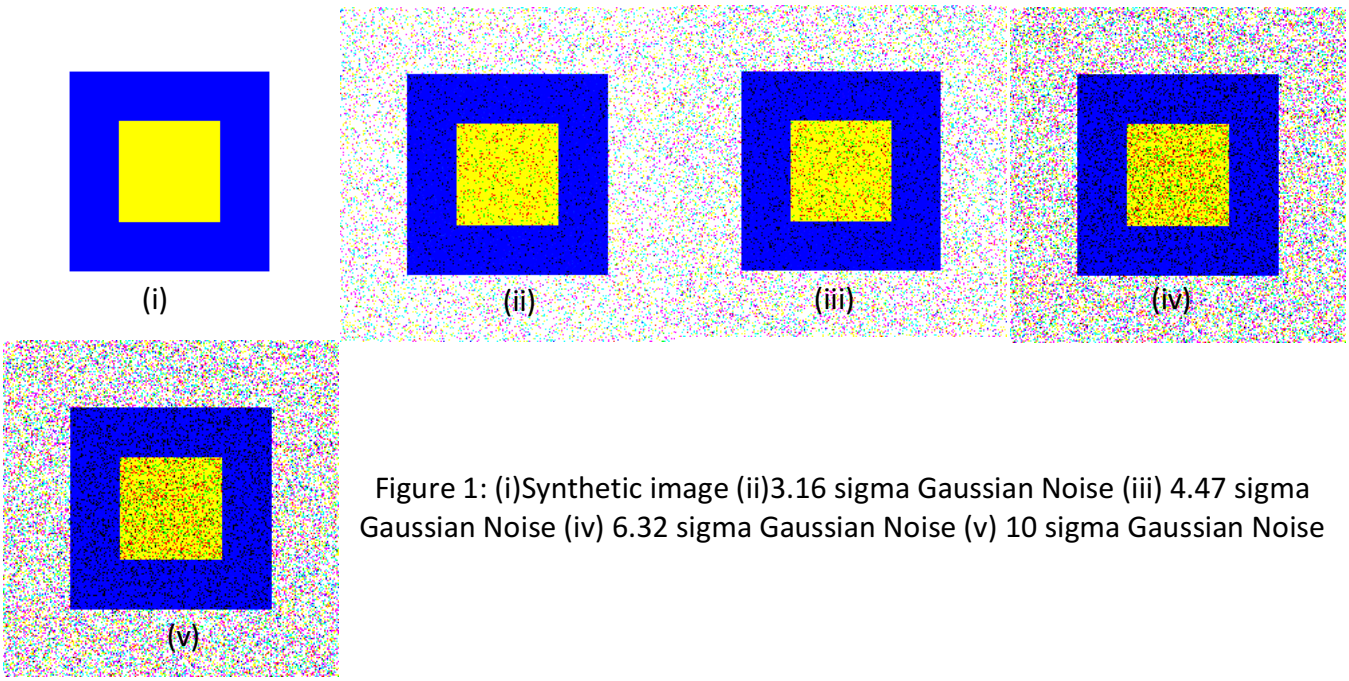

Figure 1: (i)Synthetic image (ii)3.16 sigma Gaussian Noise (iii) 4.47 sigma Gaussian Noise (iv) 6.32 sigma Gaussian Noise (v) 10 sigma Gaussian Noise

2. There weren't any guidelines as for what picture to use so I decided that the most illustrative example of the algorithm would come again from the grayscale synthetic image. Using sigma = [.5,2], I applied the Laplacian of Gaussian Filter, approximated it by the Difference of Gaussians, and then computed the zero crossing (Figures 2 and 3). TAs you can see in Figure 2 the LoG computes the best edge detection as the lines are both

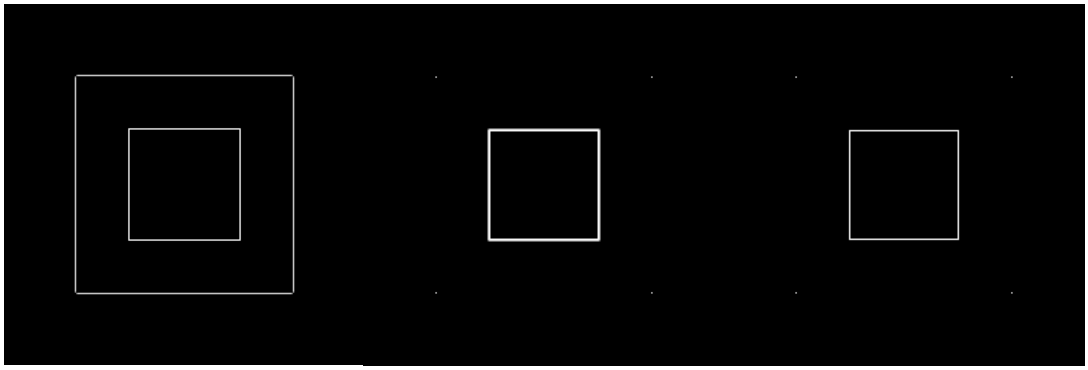

Figure 2: (left) LoG with sigma = .5 (middle) DoG with sigma = .5 (right) Zero Crossing of DoG

thin and intact. The DoG is missing the outer edge in all but the corners and the inner edge is more thick. The Zero crossing of the DoG does notably thin the thick inner edge.

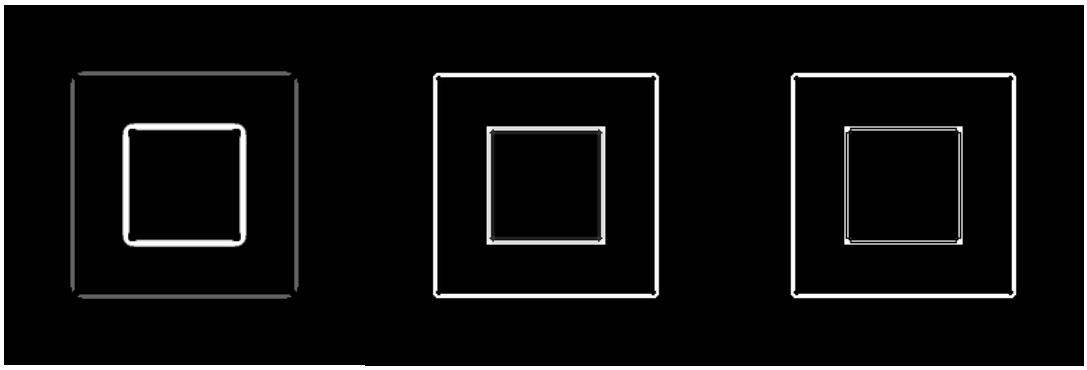

Figure 3: (left) LoG with sigma = 2 (middle) DoG with sigma = 2 (right) Zero Crossing of DoG

For Figure 3 we can see that all three filters do a good job of computing the edges. The LoG noticeably gives the less noticeable outer edges a less prominent lighting and due to the larger sigma the edges both are significantly thicker than Figure 2. The DoG is noticeably thicker than Figure 2 but it does a much better job in that both edges are

retrieved. The zero crossing thins out both edges considerably with the inner edge becoming two edges due to the thickeness. The most interesting thing I found about this process was that the LoG and DoG output a black image and would inly work once I carefully tuned the size of the filter to the sigma and brightened the image a certain extent. It seems like different pictures need different tuning given what I have observed here.

3. Using the stent picture, we first enhance the image, add a LoG filtered original image, blur the image, subtract the original, and add the difference to the original image. As we can see in Figure 4, the enhanced image brightens the image a ton. The enhanced image

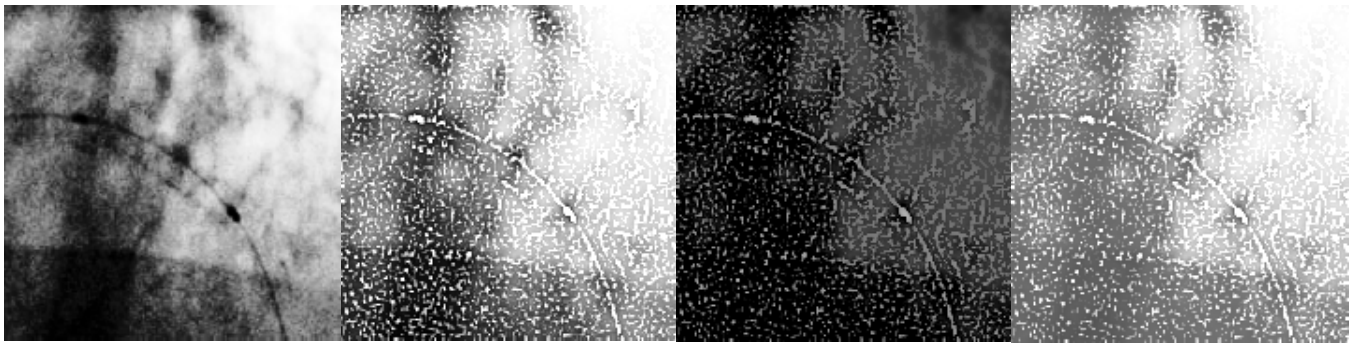

Figure 4: (i) The enhanced stent image (ii) the enhanced image + the LoG of the original image (iii) The original image subtracted by the blurred image (iv) The difference in iii added to the original image

added to the LoG of the original seems to sharpen the image. The blurred image and original image difference seems to darken the background while keeping the edges intact. Finally, the difference added to the original image sharpens the image while lightening the contrast seen in the previous images.

4. The derivative of Gaussian takes the form of an infinitely many transformations corresponding to the infinity directions the gradient can point as well as the gradient

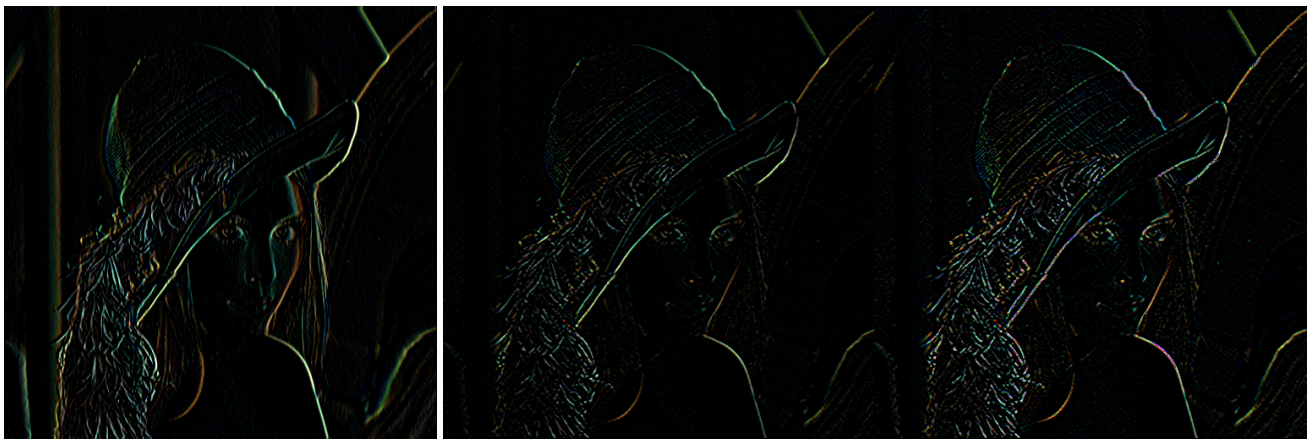

Figure 5: For sigma = 1: (Left) the dX gradient image (Middle) the dY gradient image (Right) the gradient magnitude

magnitude and gradient direction. In this part we took the gradient images in X and Y and took the magnitude using sigma = [1,3,5]. In Figure 5 we can see the for sigma = 1 we have very light edges on all three images. The X gradient is much more detailed than the Y gradient which must mean there is more change in the x direction. The magnitude

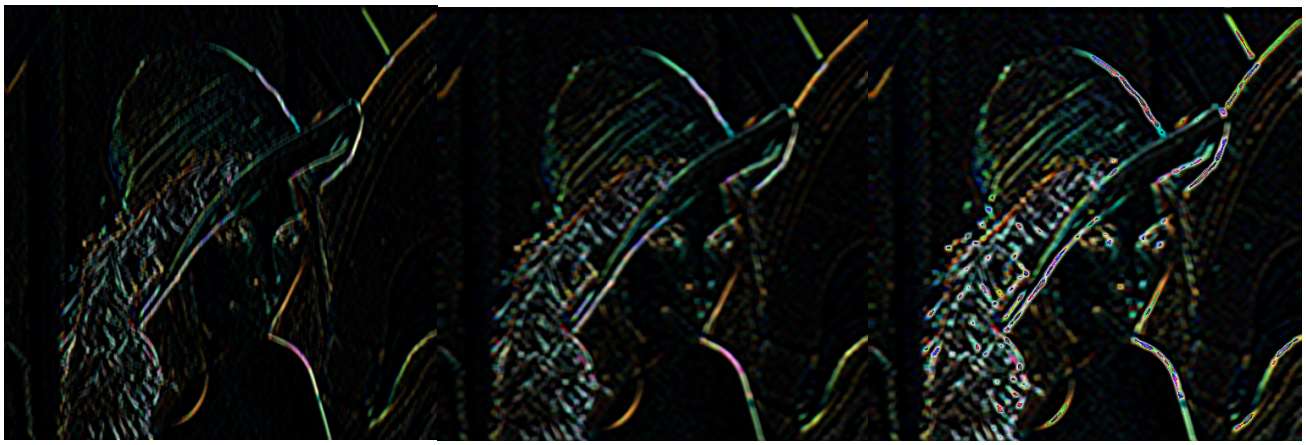

Figure 6: For sigma = 3 (Left) the dX gradient image (Middle) The dY gradient image (Right) The gradient magnitude

is the norm of the X and Y gradients and looks as such. In figure 6 we can see that when sigma = 3 there is much more detail in the Y gradient than the X gradient and that the magnitude is very detailed compared to sigma = 1. In figure 7 we can see that sigma = 5 regresses and makes it such that the X gradient is much more detailed than the Y

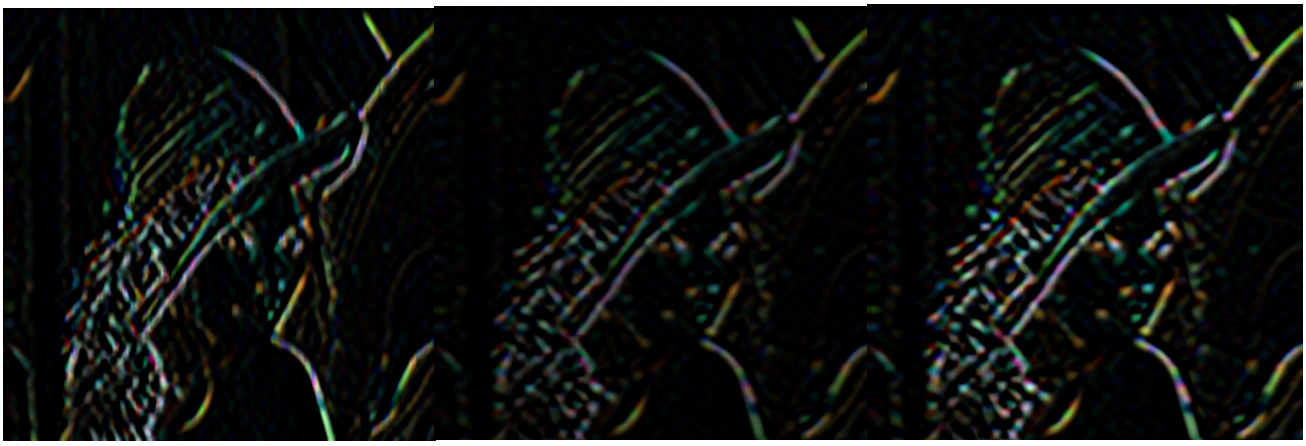

Figure 7: For sigma = 5 (Left) the dX gradient image (Middle) The dY gradient image (Right) The gradient magnitude

gradient. Additionally, we can see that the gradient magnitude for sigma = 5 is more detailed than sigma = 3 but less detailed than sigma = 1.

5. For the hough transform I spent many hours trying to figure out why I had a segmentation fault for the accumulator loading loop both times I tried coding it. I never figured it out and at the time of writing I feel I should just use the library functions and turn in a working copy and figure it out for the midterm. Using the library functions for Canny(), which I thought would be more reliable than my own functions, and

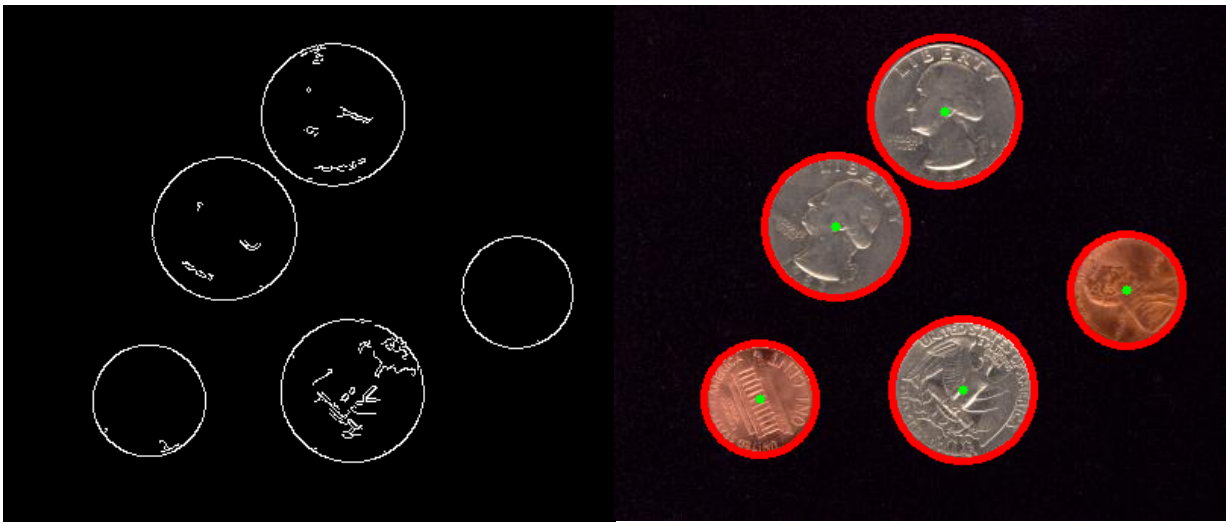

Figure 8: (Left) Canny output (Right) Hough Circle Transform output

HoughCircles() I quickly got a working output. It took a couple tries to figure out which threshold values would find all the circles and figure 8 shows the eventual output.