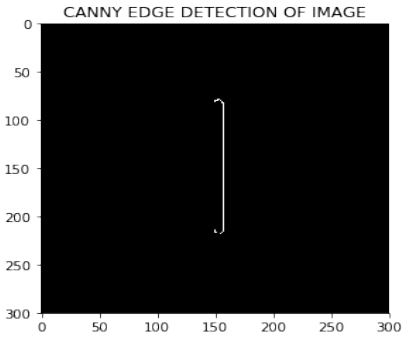
PLEASE NOTE! I was granted a 24-hour deadline extension by Professor Wu.
I cc'd the TAs in the email conversation.

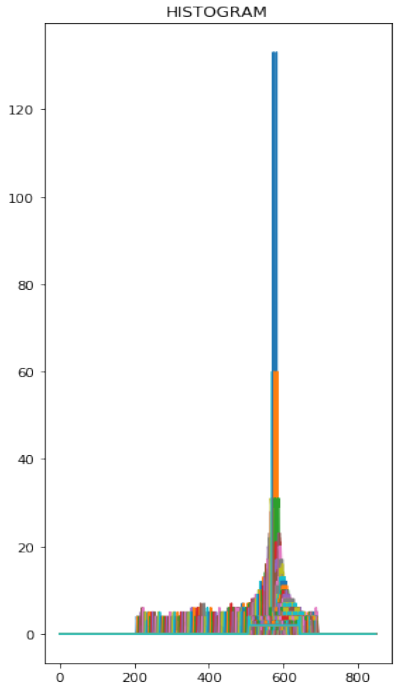
For my results, you can open ***main.ipynb*** via Jupyter Notebook. You will see everything laid out neatly. An example function to run is **plot_hough()**, which takes in the **hough()** of an array from the **Canny Edge Detector** function. Throughout the notebook, you can see the functions I use along with their corresponding graphs.

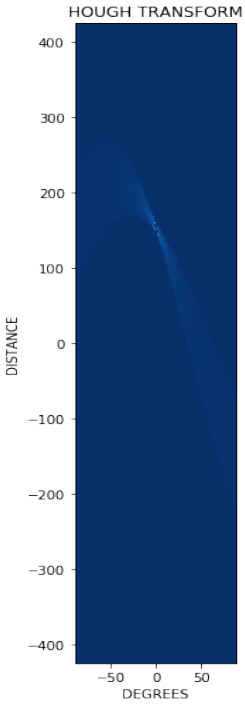
This week we studied Hough Transform, which is a feature extraction technique to find imperfect instances of an object by voting. Essentially, we take an input image, convert that into an edge image by using any edge detector that we previously talked about (in this program I use the Canny Edge Detector), and then detect the lines of this image. Hough Transform can now be applied to more than just simple lines, and can also apply to, say, circles, depending on the equation we give it, but the general idea in this assignment was to simply detect straight lines in an image, whether it be binary or a picture. Once our detector detects an edge point, it iterates over all possible theta and rho values to define the occurrences in a feature space.

A Cartesian line has the equation y = mx + b where 'm' is the slope of the line and 'b' is the y-intercept. A line in the Polar-coordinate system has the equation ρ = x*cos(θ) + y*sin(θ), where 'ρ' is rho, the maximum distance of the whole line, and 'θ' is theta, the angle from the origin of the line. When we ultimately plot the results of our Hough Transform, we will see a number of polar lines in the angles vs. distance graph that show the count intensity, where the greatest count (or brightest pixels), indicate both the distance of a major line along with its angle. The only problem I had was that I ran out of time, and **could not solve for the y-intercept**. While I found the distance and angle of each major line, I could not plot it onto a second graph because I kept getting errors in finding the y-intercept. I do understand the equation, where m = $\frac{-\cos(\theta)}{\sin(\theta)}$, and b = $\frac{\rho}{\sin(\theta)}$, but again, I just kept getting errors in plotting.

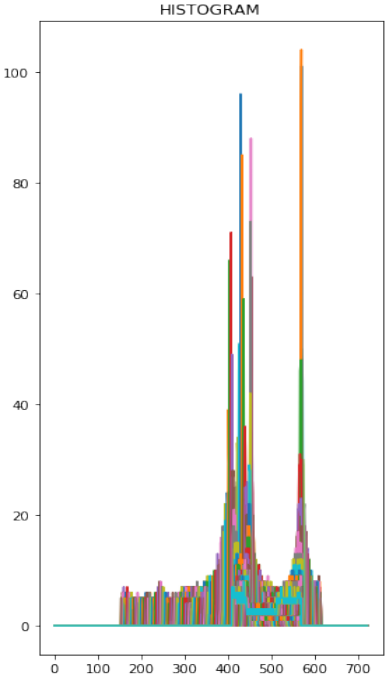
Regardless, I understand how it works well enough. The main objective was to calculate the accumulator array, which requires extracting the edges and for each degree, create a dictionary of cumulative addition of the sum of the rho index with x-index and cosine of the degree, and y-index and sine of the degree. I also plotted a histogram of the results. The algorithm requires the following steps: (1) Detect every edge (done via Canny), (2) create a range of theta and rho based on the image, (3) accumulate the number of rho values and number of columns equal to the number of theta values, (4) vote to find the nearest rho value per edge point, and then find those local maxima to show the most prominent lines. If the number of desired lines is provided ahead of time, that makes our job simpler. As an end result, I got the following images, including creating my own test image titled 'untitled'.

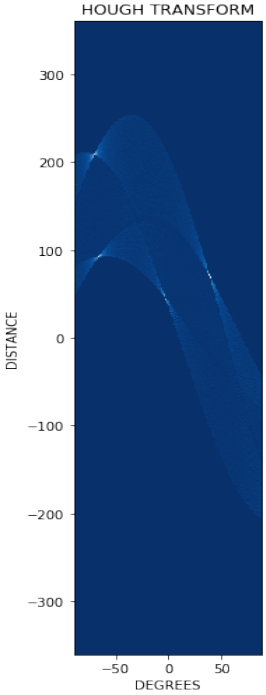

CANNY EDGE DETECTION OF IMAGE

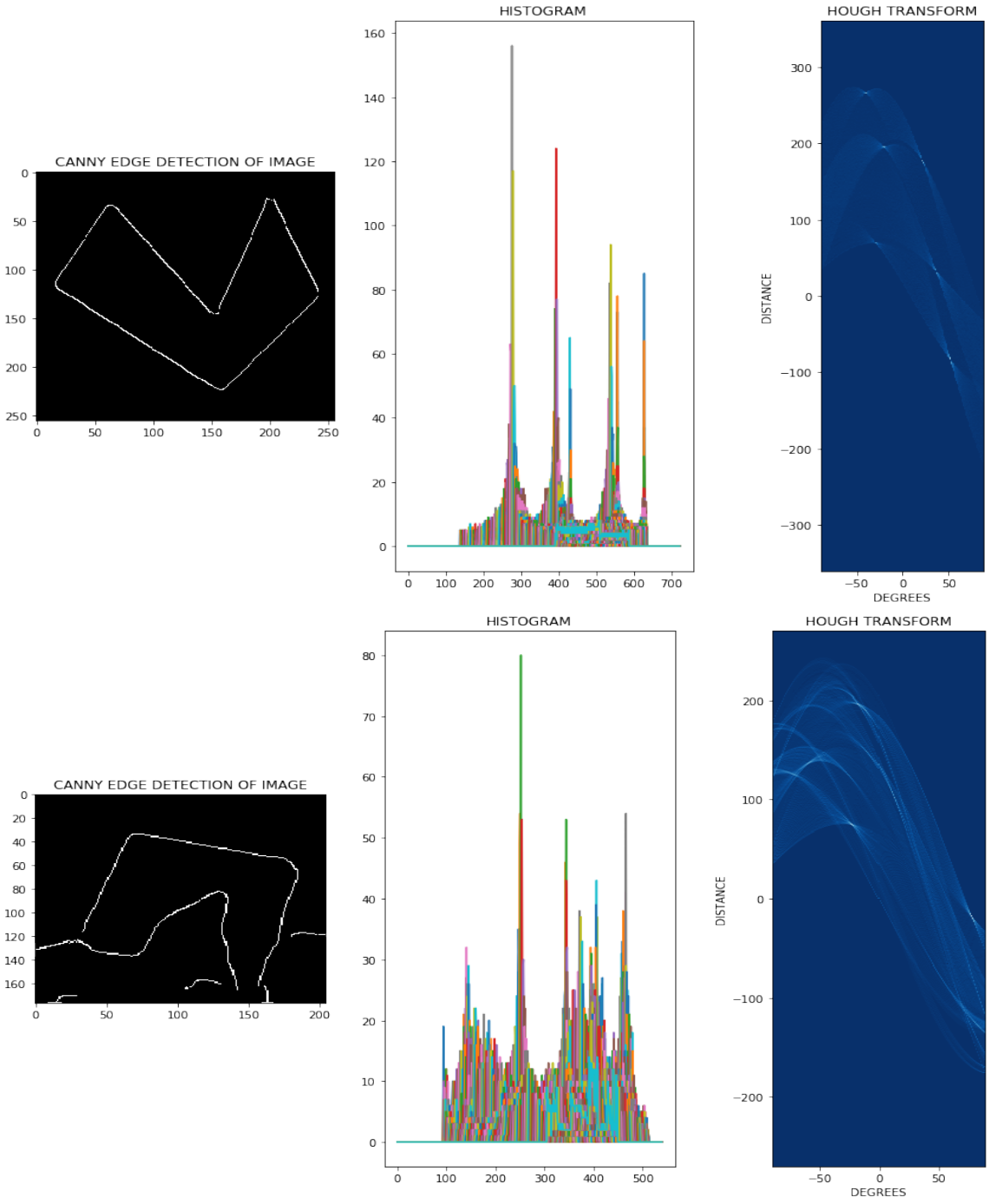

HISTOGRAM


HOUGH TRANSFORM
DISTANCE
DEGREES


CANNY EDGE DETECTION OF IMAGE

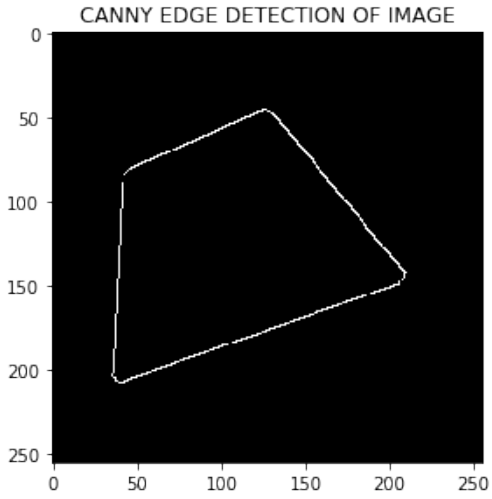

HISTOGRAM

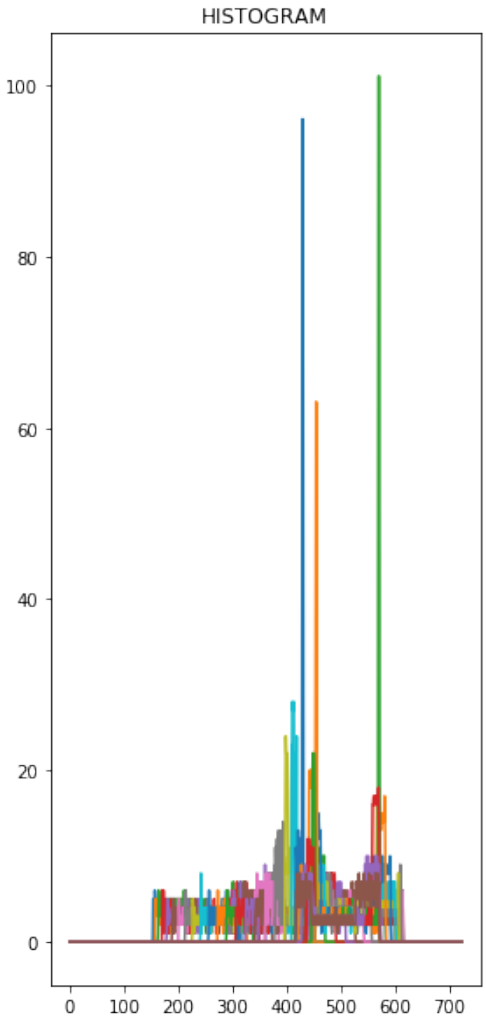

HOUGH TRANSFORM
DISTANCE
DEGREES

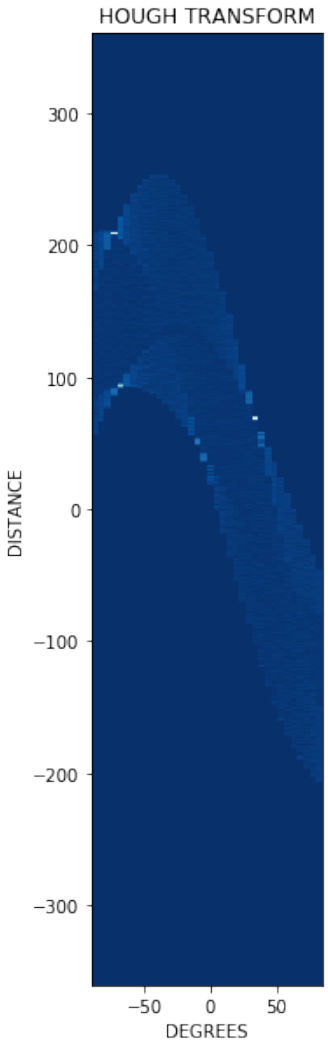


As you can see, the Canny Edge detector shows some lines on the left, of which the straight ones are the lines we're most interested in. The middle images are histograms denoting the local maxima (peaks) of the accumulator matrix. The images on the right are the Hough Transform results applied to the image on the left, where the brightest points (significant intersections) are the result of overlapping rho by theta points from the polar coordinate space. If you look at the 2nd image on the page above (test.bmp), you'll see that there are 4 bright points in the Hough Transform space, as well as 4 local maxima peaks in the histogram. This corresponds to the 4 straight lines detected in our canny edge image. Again, I ran out of time to calculate the y-intercept of each line, but I did have the angle and distance of each line collected.

I did test different quantization in the parameter space, and got the following image:







There's a very different histogram and Hough transform result (also quite blurry), where there only seem to be 3 or so noticeable local maxima. It's best to quantize with the largest available range per degree in my opinion. Regardless, we can get around this quantization problem by presetting a number for the # of lines we want to detect in the image.