Submission of the assignment 3 of Computer Vision course

Xiaoke(Jimmy) Shen

May 18, 2017

1 Assignment problem 1

1.1 Description of the assignment

Show that the derivative of a 1-D signal, $I = I(x)$, amplifies the higher frequency components of the signal. Then, prove that even a slight amount of noise $n = n(x)$, can be responsible for a large difference between the derivative I and I+n (Hint: Assume that n can be written as $\varepsilon sin(x)$ for some very small ε and very large ω). (2 points)

1.2 Submission of the problem 1

For the first order derivation is commonly used to do the edge detection. Meanwhile, for the 2-D image we know that the high pass filter can do the exactly edge detection. This can be the most intuitional explanation for the first part of this question. For more detail, some basic math can be done here:
Suppose the 1-D signal, $I = I(x) = sin(2\pi hx) + sin(2\pi lx)$, where h is the larger than l then the $sin(2\pi hx)$ part will be the signal with the high frequency while the $sin(2\pi lx)$ part will be the signal with the low frequency.

$$\frac{dI}{dx} = 2\pi hcos(x) + 2\pi lcos(x)$$

We can get the conclusion directly from the result above that the higher frequency components of the signal will be amplified by the factor of $2\pi h$ which is greater than the lower frequency which is $2\pi l$

Here is the prove for the second part of this question:

Proof. Assume that n can be written as $\varepsilon sin(x)$ for some very small ε and very large ω, and then

$$\frac{d(l+n)}{dx} = \frac{dl}{dx} + \varepsilon\omega cos(x)$$

Then a large difference between the derivative I and $I+n$ can be achieved as the contribution of the $\varepsilon\omega$ as the scope of $cos(x)$ is from -1 to 1 and then the scope of the $\varepsilon\omega cos(x)$ will be $-\varepsilon\omega$ to $\varepsilon\omega$. Meanwhile, ε is very small and $\omega$is very large we can get a very large $\varepsilon\omega$.

□

2 Assignment problem 2

2.1 Description of the assignment part 2

Show that if you use the line equation $xcos\theta + ysin\theta = \rho$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y). Does the period (or frequency) of the sinusoid vary with the image point (x, y)? (2 points)

2.2 Submission of the assignment part 2

The definition of the Sinusoid is: "A curve similar to the sine function but possibly shifted in phase, period, amplitude, or any combination thereof." based on "http://mathworld.wolfram.com/Sinusoid.html".
For each image point (x, y) , we can have
$xcos\theta + ysin\theta = \rho \Rightarrow xsin(-\theta + \pi/2) + ysin\theta = \rho$, where x and y are some constant value related to the image point(x, y)
Then we can see that the line equation $xcos\theta + ysin\theta = \rho$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space.

As we can have:
$xsin(-(\theta + 2\pi) + \pi/2) + ysin(\theta + 2\pi) = xsin(-\theta + \pi/2) + ysin(\theta)$
we can get 2π is the period of this sinusoid function. As it is not related to the value of both x and y, the period (or frequency) of the sinusoid will not vary with the image point (x, y)

3 Programming Assignment

3.1 Description of the assignment

Your task is to develop a vision system that recognizes lines in an image using the Hough Transform. Such a system can be used to automatically interpret engineering drawings etc. We will call it the "line finder." Three gray-level images are provided to you: hough_simple_1.pgm, hough_simple_2.pgm and hough_complex_1.pgm. (They can be found in blackboard). It is enough that your line finder works on the "simple" images. If the results are good, you can try the "complex" image. Your program will be tested using not only these two images but test images we have taken.
The task is divided into four parts, each corresponding to a program you need to write and submit. Each program must accept arguments in the format specified as
program_name {1st argument}{2nd argument} ... {Nth argument}.

- (a) First you need to find the locations of edge points in the image. Write a program named h1 that locates edges in a gray-level image and generates an "edge" image where the intensity at each point is proportional to edge magnitude:
  h1 {input gray-level image}{output gray-level edge image}.
  For this you may either use the squared gradient operator or the Laplacian. Since the Laplacian requires you to find zero-crossings in the image, you may choose to use the squared gradient operator. The convolution masks proposed by Sobel should work reasonably well. Else, try your favorite masks. 4 points.

- (b) Threshold the edge image so that you are left with only strong edges. You should have a program named h2 that thresholds a gray-level image at a certain threshold value:
  h2 {input gray-level image} {input gray-level threshold} {output binary image}.
  For that you could just rename p1 from the previous programming assignment.

- (c) Next, you need to implement the Hough Transform for line detection. Write a program named h3 that generates an image of the Hough Transform space of a given binary edge image: h3 {input binary edge image} {output gray-level Hough image} {output Hough-

voting-array}. The brightness of each pixel (voting bin) in the output gray-level Hough image should be proportional to the number of votes it receives. The last output filename will store the votes of the voting array in a representation of your choice. Note that the gray-level output will be used just for visualization.
As discussed in class, the equation $y = mx + c$ is not suitable as it requires the use of a huge accumulator array. So, use the line equation$xcos\theta + ysin\theta = \rho$. Note that θ must lie between 0 and π, and very large values of ρ correspond to lines that lie outside the image. You can use these constraints to limit the size of the accumulator array. The resolution of the array must be selected carefully. Low resolution will not give you sufficient accuracy in estimated parameters, and very high resolution will increase computations and reduce the number of votes in each cell (or bin). You may want to vote for small patches rather than points in the accumulator array. 4 points

- (d) Write a program named h4 that finds lines in the image from its Hough Transform space, using a given threshold, and draw the detected lines on a copy of the original scene image: h4 {input original gray-level image} {input Hough-voting-array} {input Hough threshold value} {output gray-level line image}. Make sure that the lines you draw are clearly visible irrespective of pixel brightness values (dark or bright). Straight lines in the image will produce blurred "areas of brightness" in the Hough space not single points, as predicted by the theory. The theoretical model uses the approximation that lines are infinitely long and infinitely thin - which is not exactly true in practice. The suggested approach to deal with this problem is to first threshold the Hough space image by cutting out all pixels below the given threshold value, and then segment the result into a number of "areas of brightness", each area presumably corresponding to a particular straight line (for that you could adapt the code developed in the previous assignment). Then in order to calculate the parameters of the line, you would have to find the "center" of its bright area. Notice that the approach we used when calculating positions of binary objects will probably not work, as different points in the Hough space will have different brightness (number of votes), and should have different contribution into the position of the "center". You may want to use a weighted average based on points? intensities to locate the "center" (similar to how you would locate the center of mass of a non-homogenuous 2 body). 8 points

- (e) Bonus Question: Note that the above implementation does not detect end-points of line segments in the image. Modify h4 to make it prune the detected lines so that they do not extend beyond the objects they belong to. If your are successful, you get 4 bonus points!
  As usual, you should submit a README file specifying what threshold values you used.

3.2 Submission

3.2.1 code submission

The code is written based on python 3.5
And the code related to h1, h2, h3, h4 are given in with the file name: assignment3_h1.py, assignment3_h2.py, assignment3_h3.py, assignment3_h4.py. The bonus part is given in the assignment3_h5.py file.

The results are shown in the following sessions.

3.2.2 Submission for the hough simple 1 image

The results of the hough simple 1 are shown in this section. From the results the main conclusion that:

- we can get is by using the canny edge detector(X direction Sobel filter, Y direction Sobel filter, Non max compression), we can detect the edge pretty well.
- The hough transform can well detects the lines in the image
- If the resolution of the hough space is too low, then the location of the edge will get bad result because of the low accuracy.

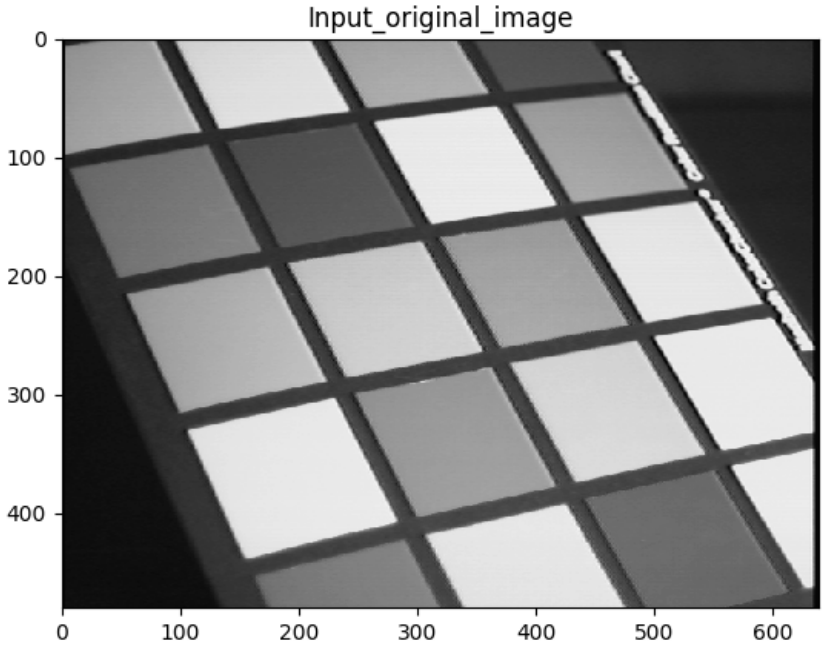

Figure 1: Input original image

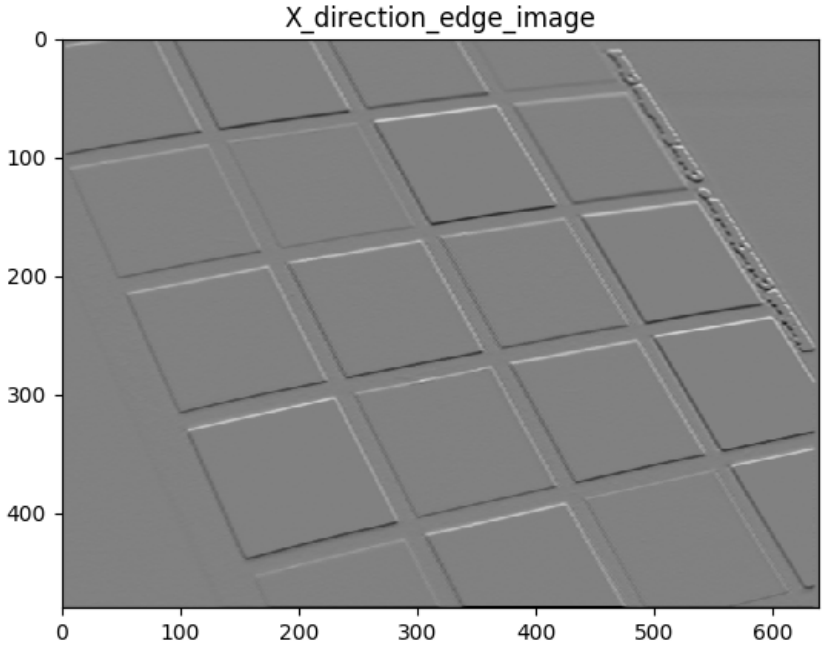

Figure 2: X direction edge image

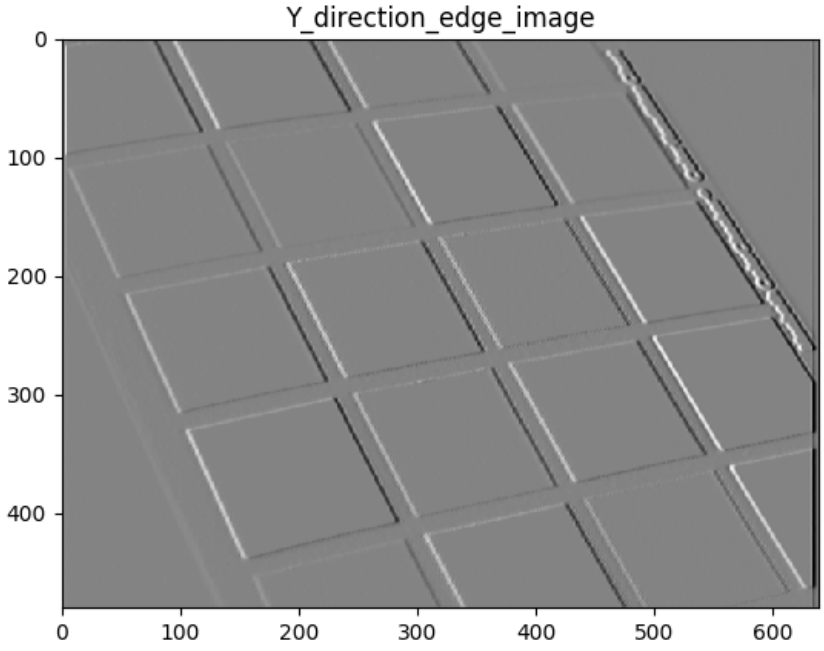

Figure 3: Y direction edge image

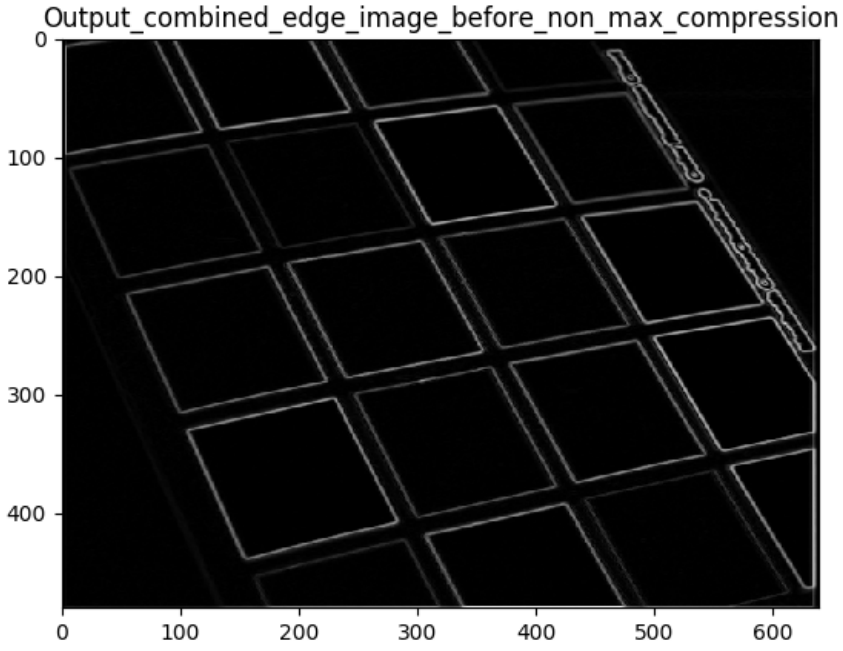

Figure 4: Output combined edge image before non max compression

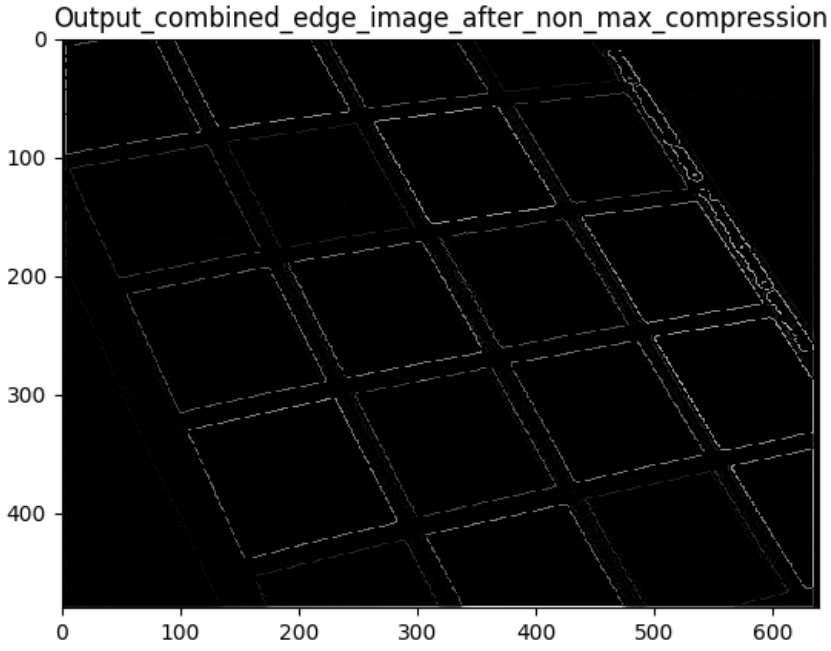

Figure 5: Output combined edge image after non max compression

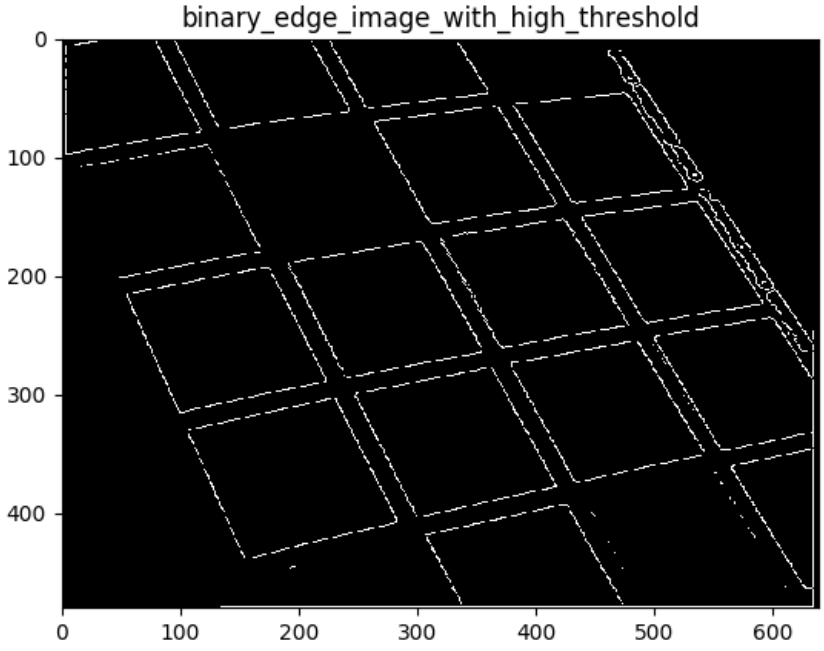

Figure 6: binary edge image with high threshold

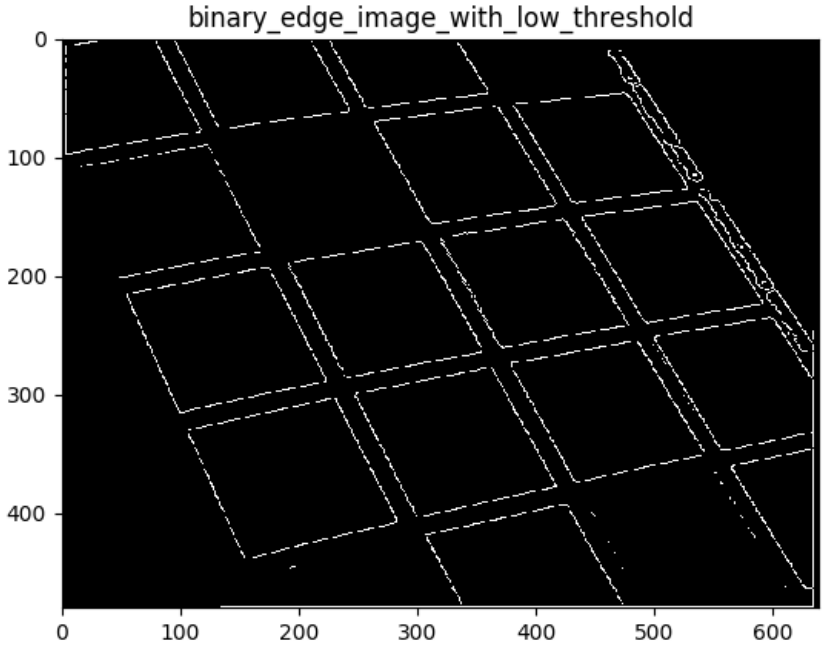

Figure 7: binary edge image with low threshold

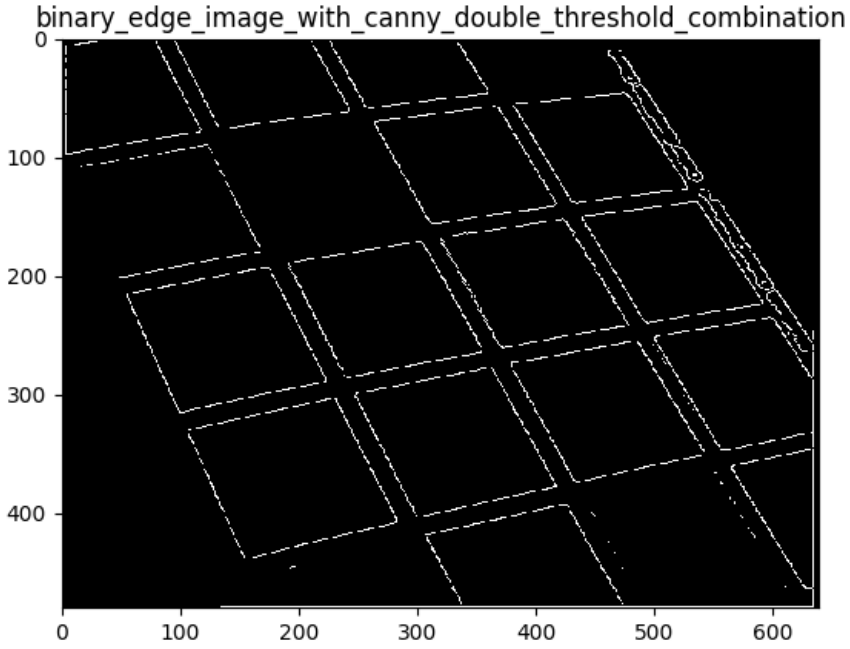

Figure 8: binary edge image by combining the low threshold and the high threshold based on the canny edge detection algorithm.

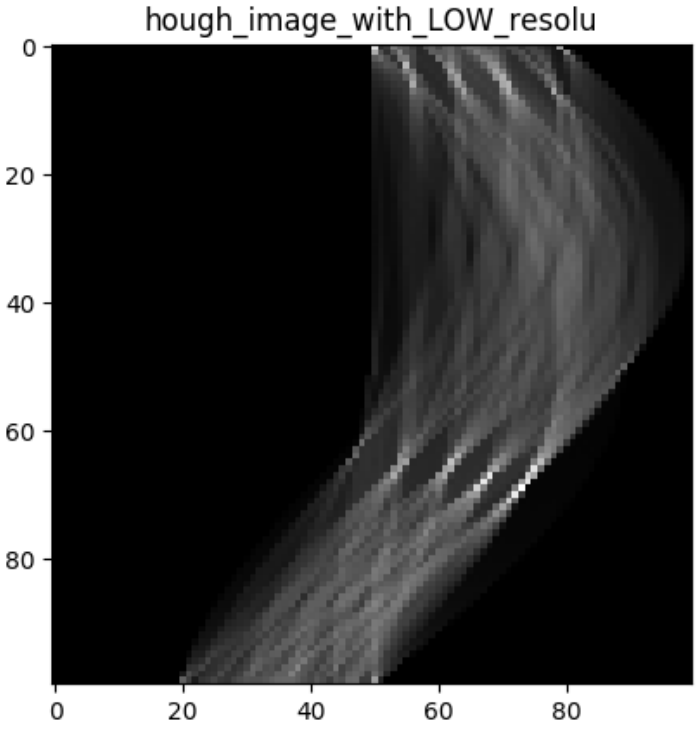

Figure 9: hough image with LOW resolution

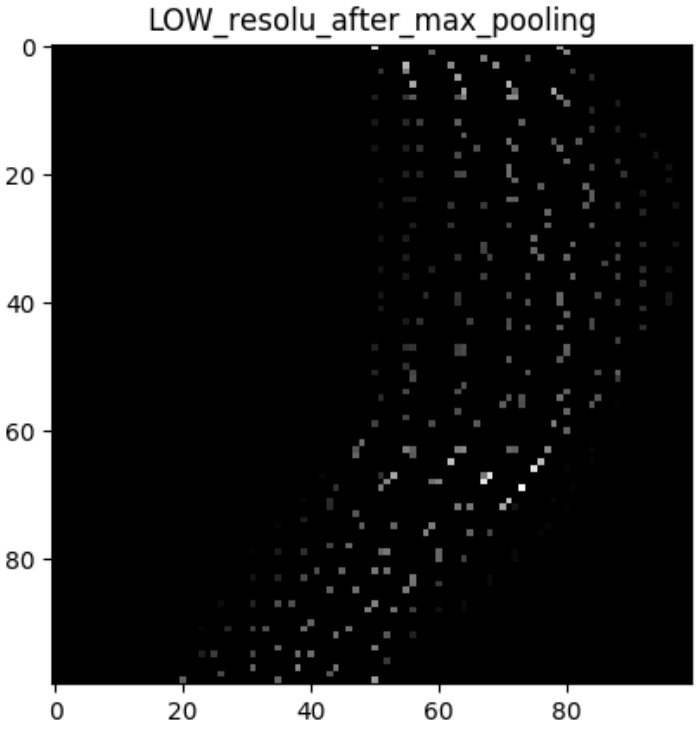

Figure 10: hough image with LOW resolution after max pooling

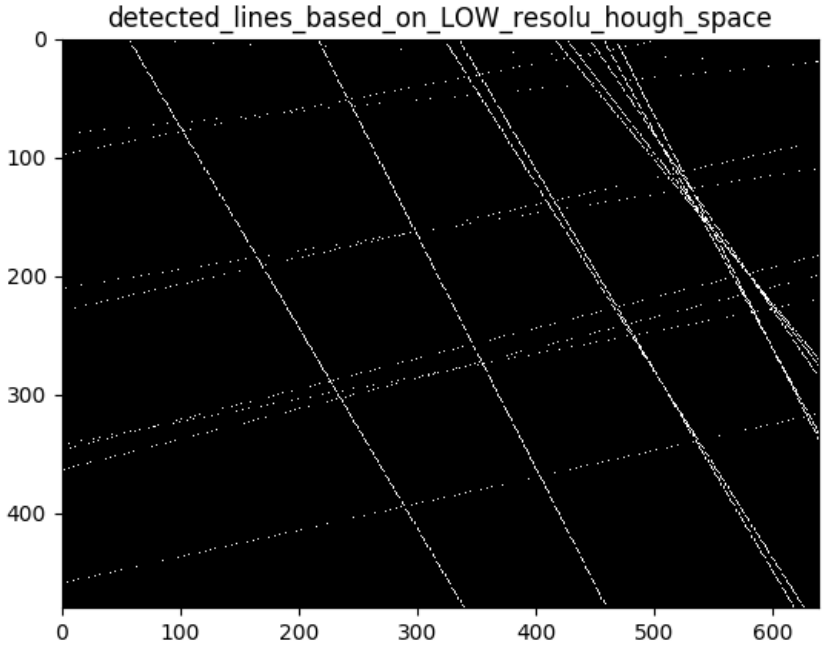

Figure 11: Detected lines based on LOW resolution hough space image

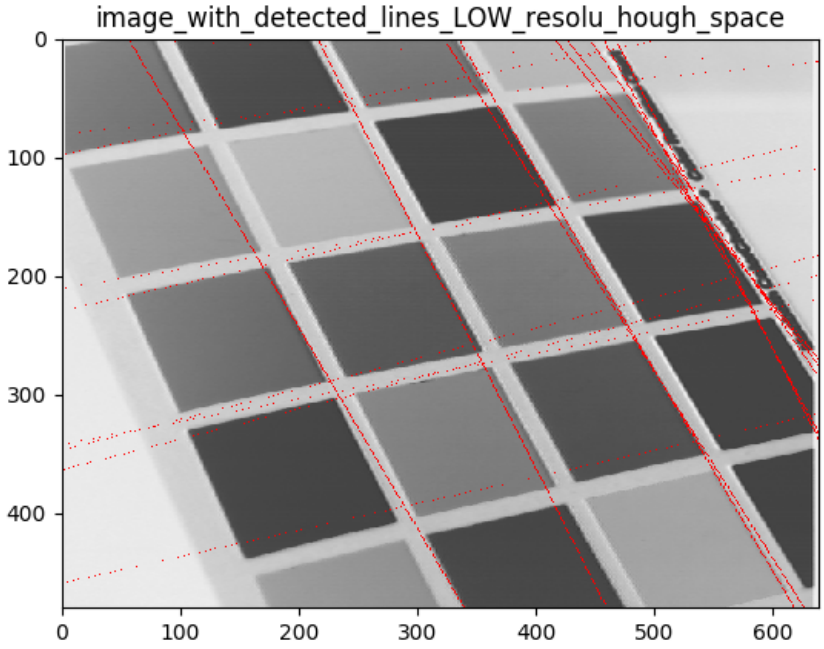

Figure 12: Combine the original image with the detected lines based on LOW resolution hough space image

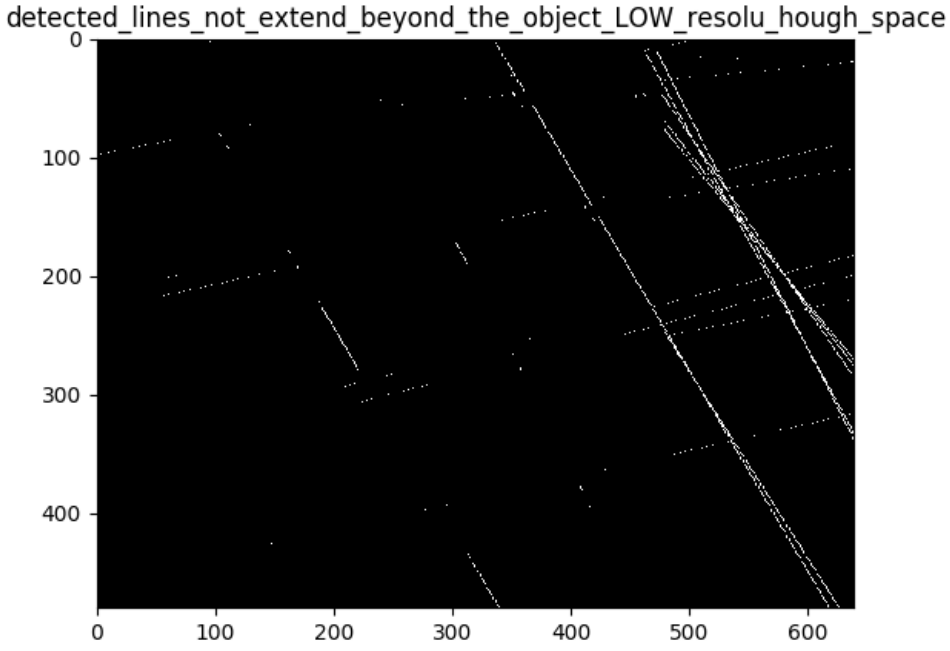

Figure 13: Detected lines not extend beyond the object based on LOW resolution hough space image

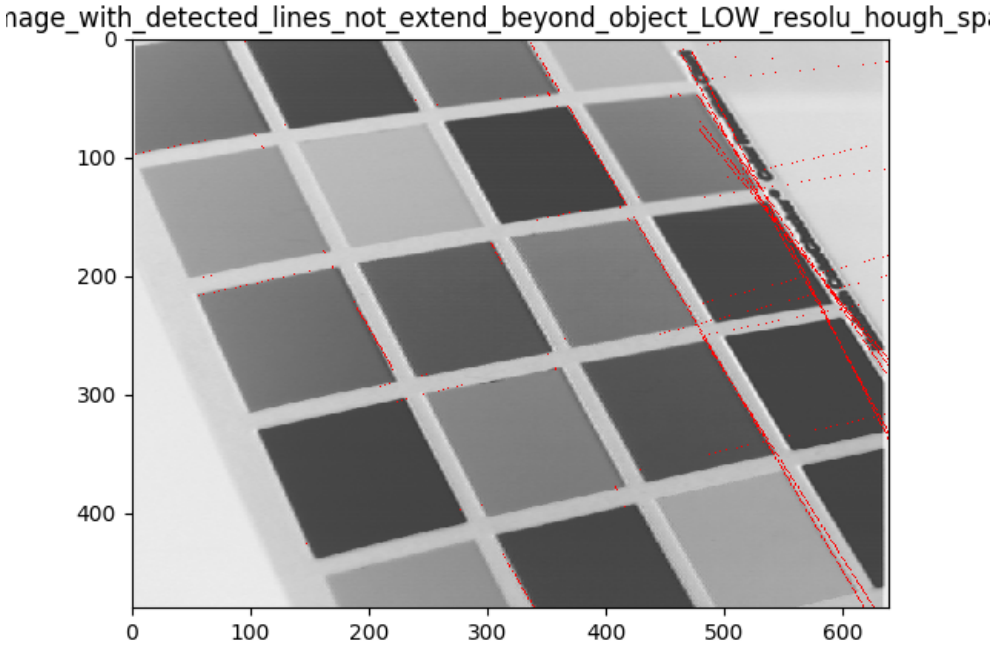

Figure 14: Combine the original image with the detected lines not extend beyond the object based on LOW resolution hough space image

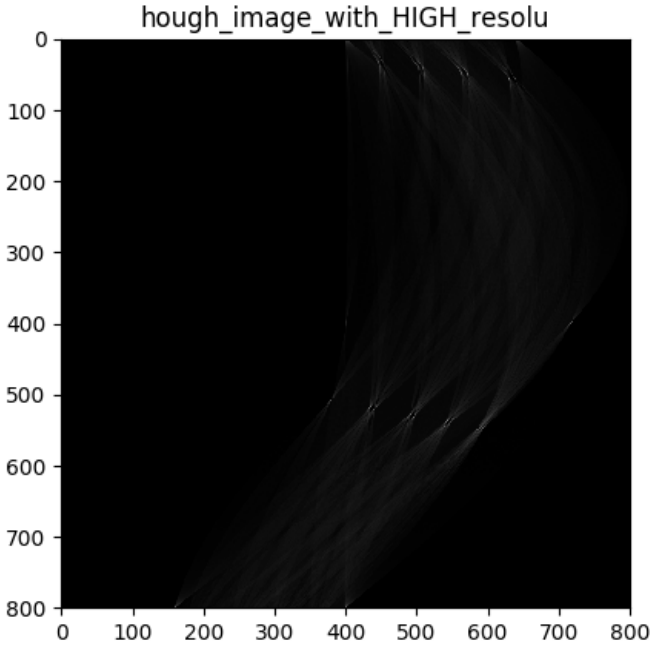

Figure 15: hough image with HIGH resolution

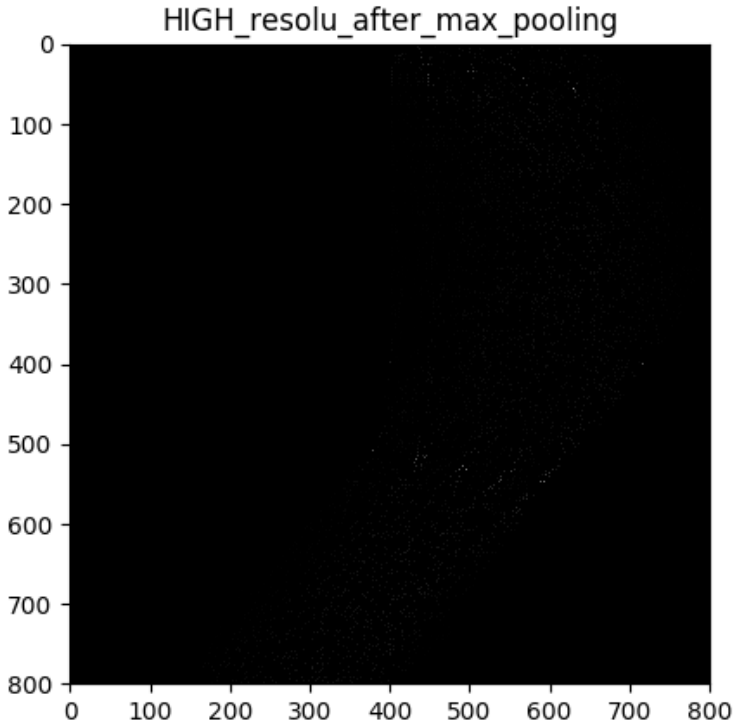

Figure 16: hough image with HIGH resolution after max pooling

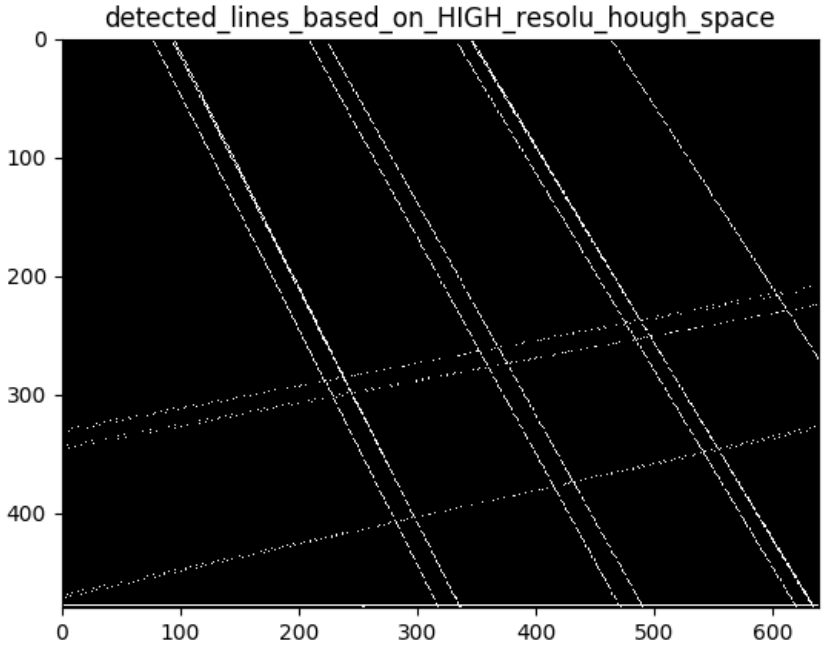

Figure 17: Detected lines based on HIGH resolution hough space image

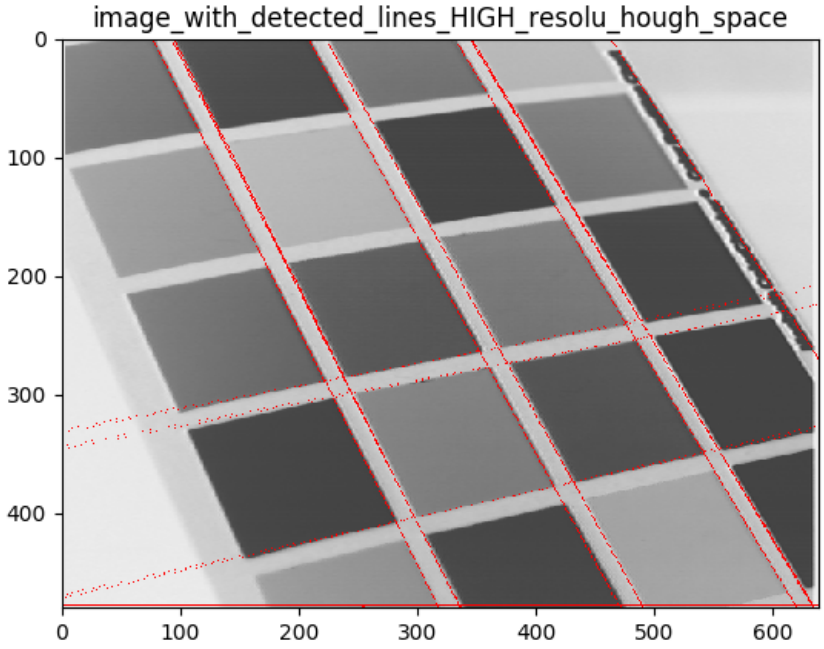

Figure 18: Combine the original image with the detected lines based on HIGH resolution hough space image

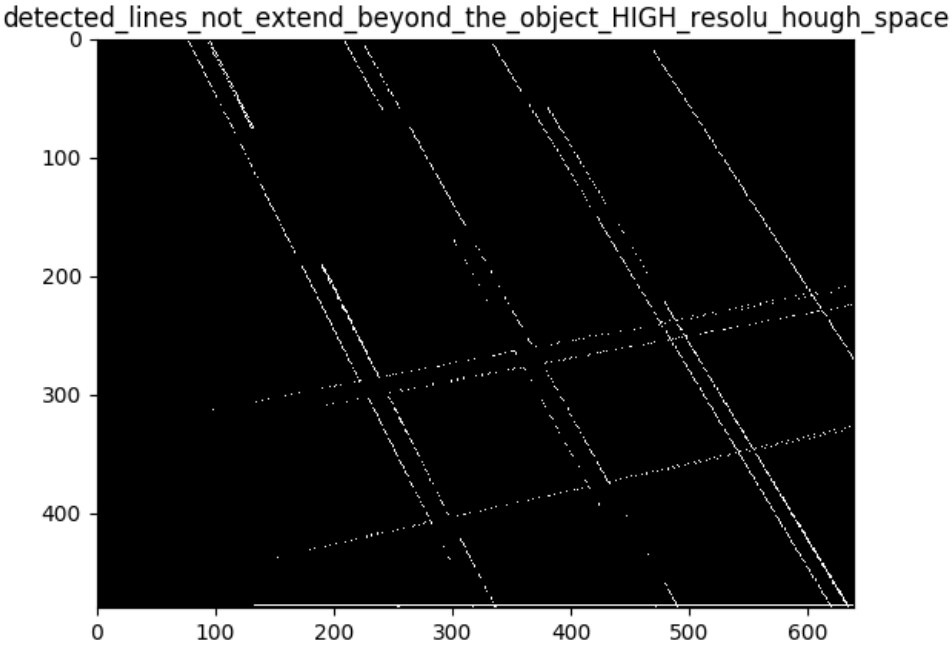

Figure 19: Detected lines not extend beyond the object based on HIGH resolution hough space image

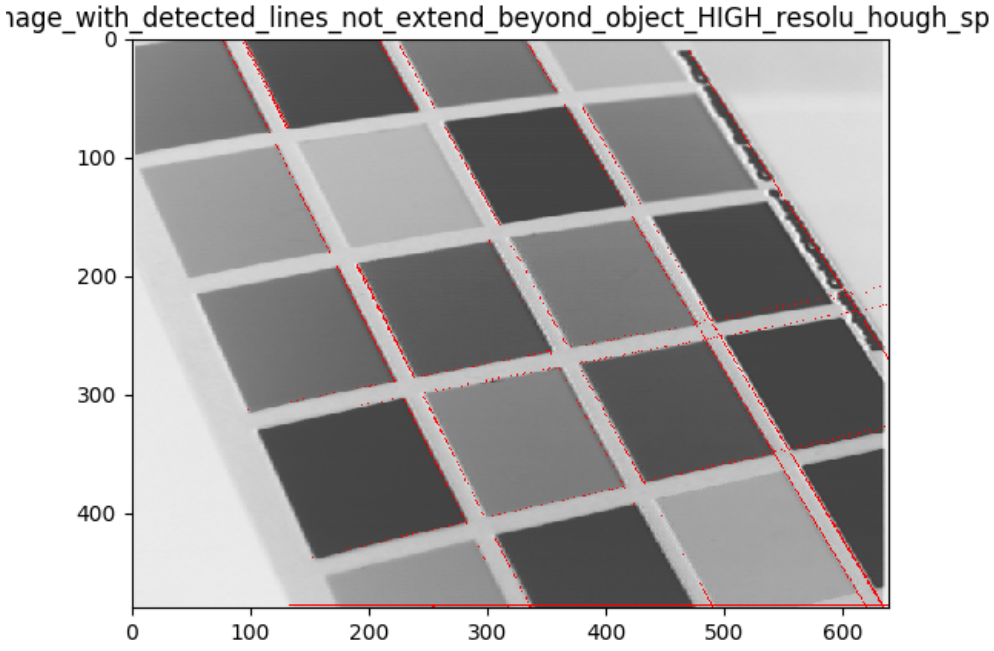

Figure 20: Combine the original image with the detected lines not extend beyond the object based on HIGH resolution hough space image

3.2.3 Submission for the hough simple 2 image

The results of the hough simple 1 are shown in this section.

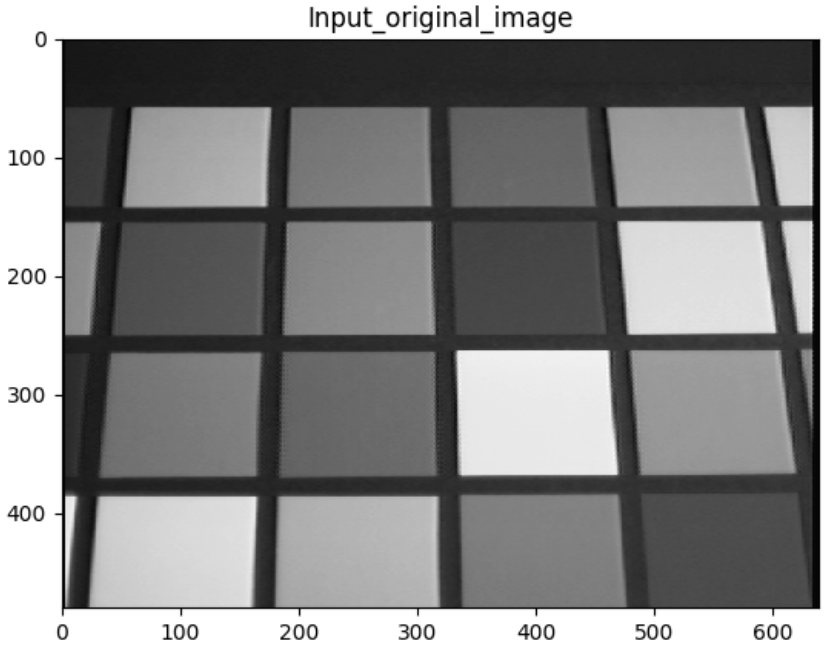

Figure 21: Input original image

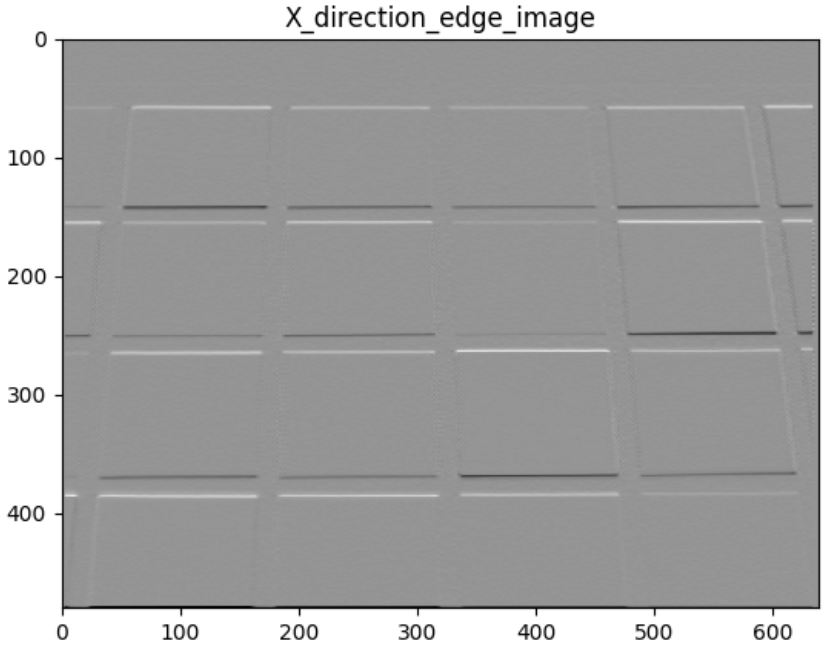

Figure 22: X direction edge image

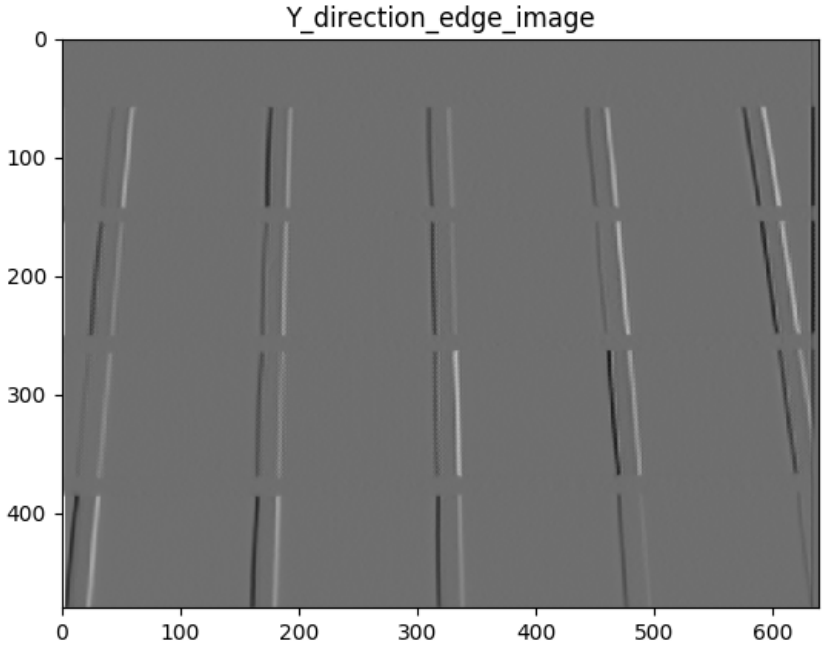

Figure 23: Y direction edge image

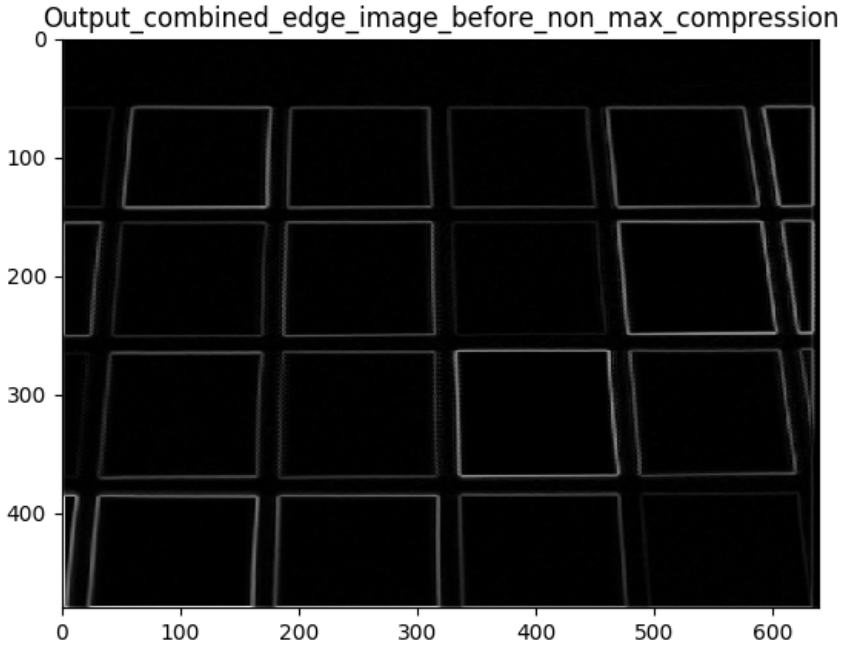

Figure 24: Output combined edge image before non max compression

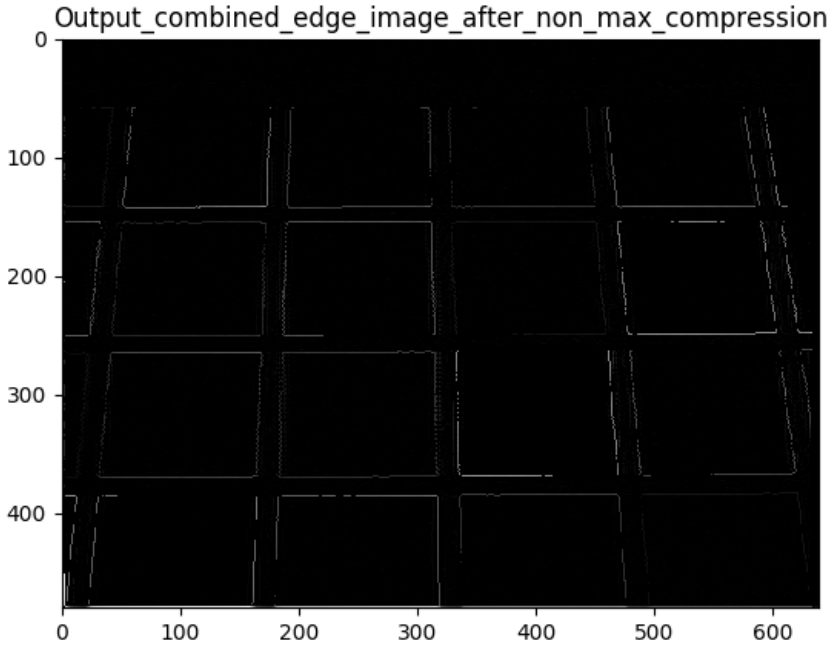

Figure 25: Output combined edge image after non max compression

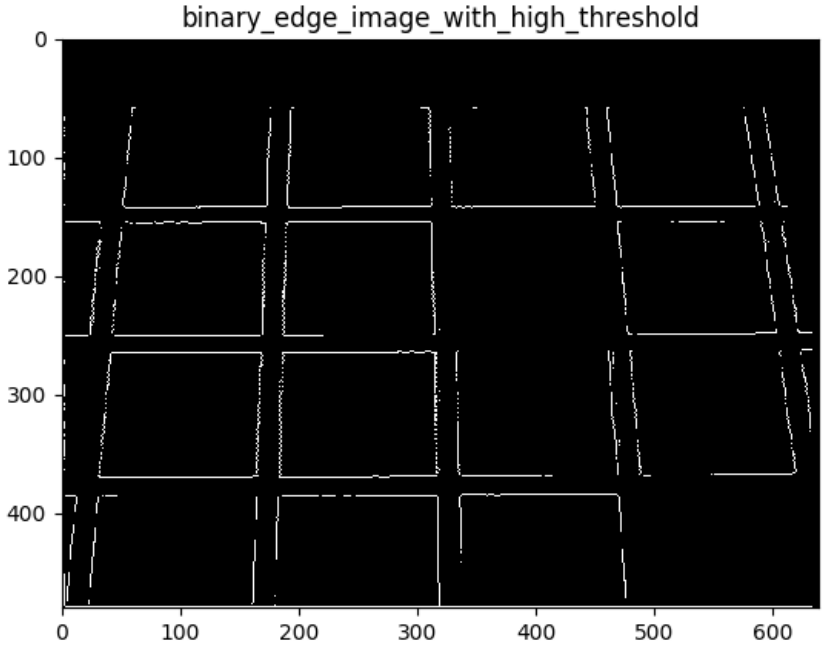

Figure 26: binary edge image with high threshold

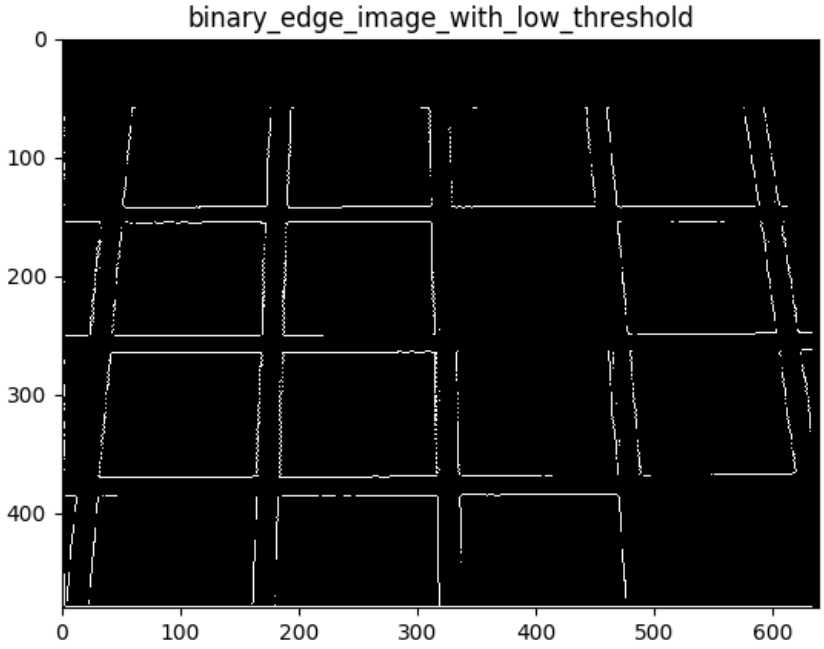

Figure 27: binary edge image with low threshold

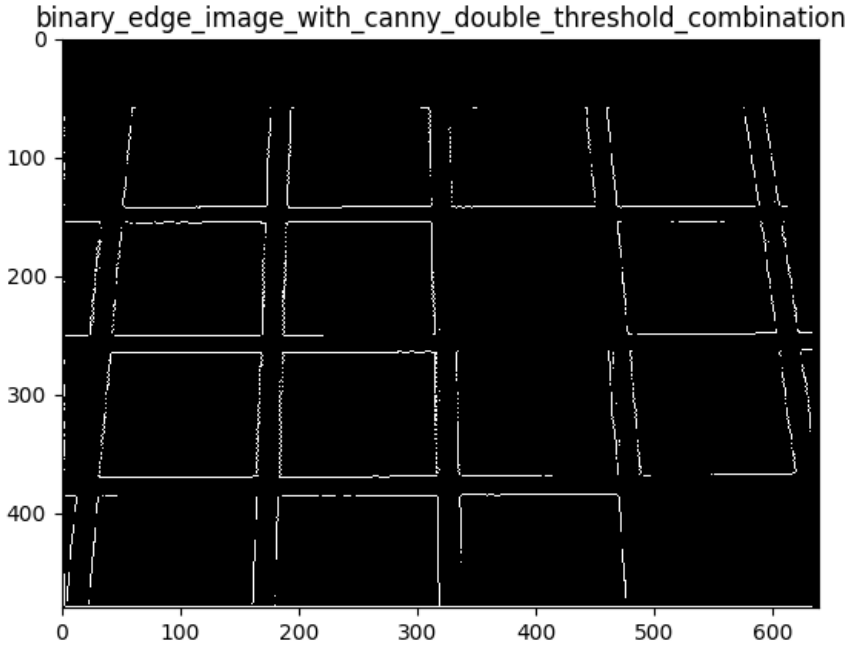

Figure 28: binary edge image by combining the low threshold and the high threshold based on the canny edge detection algorithm.

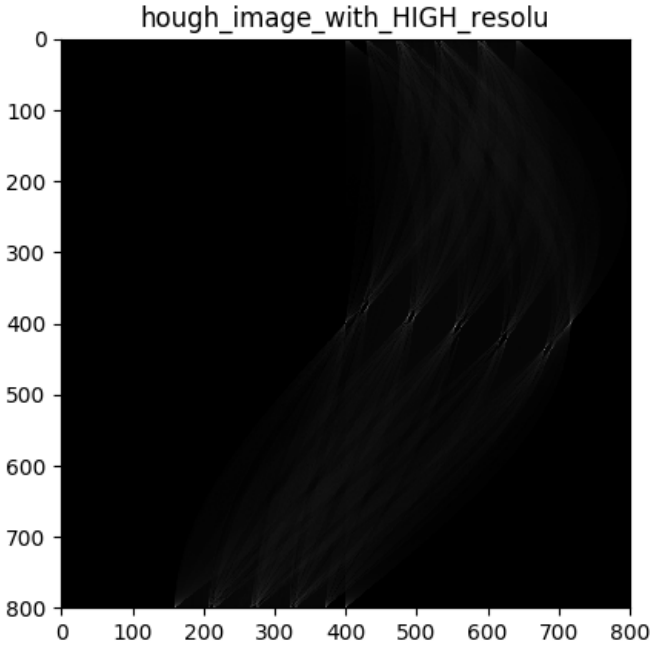

Figure 29: hough image with HIGH resolution

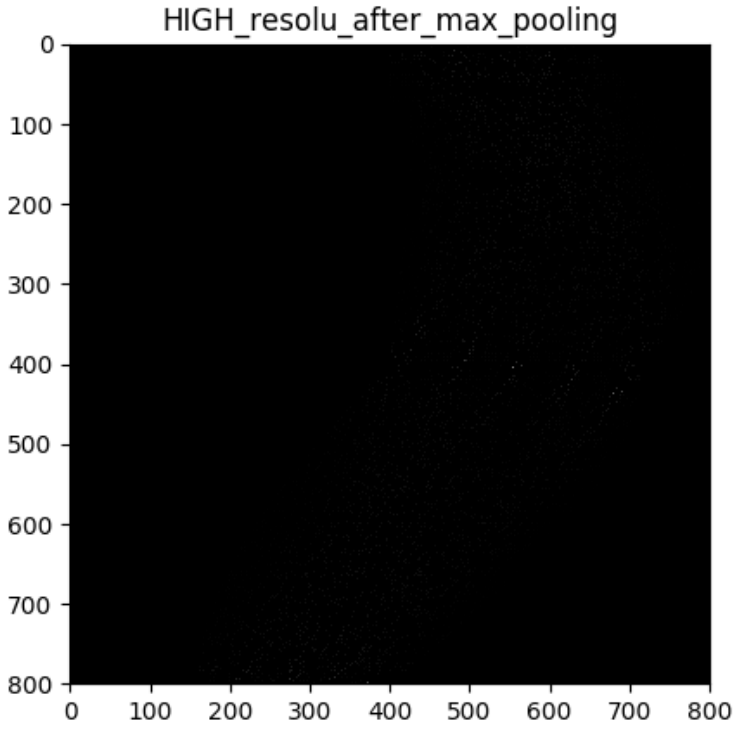

Figure 30: hough image with HIGH resolution after max pooling

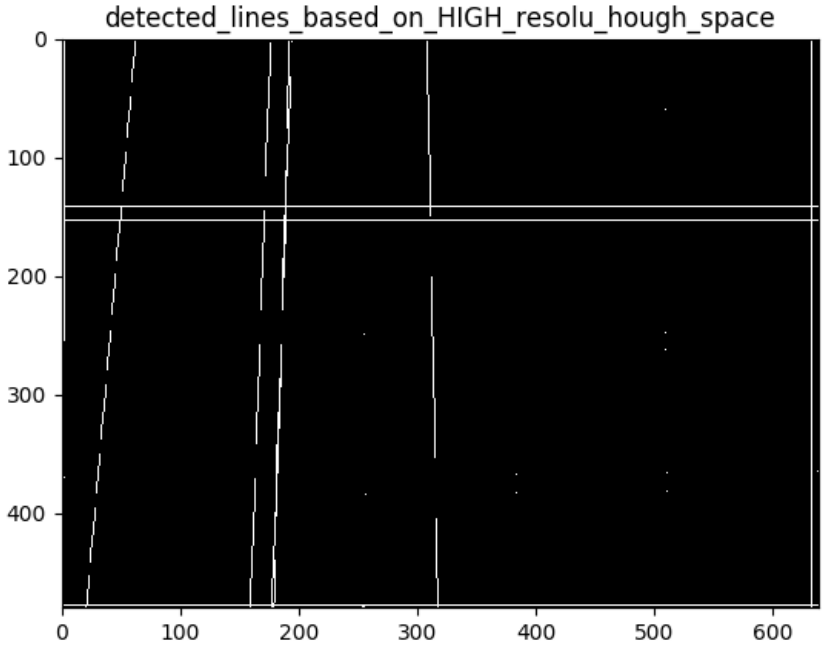

Figure 31: Detected lines based on HIGH resolution hough space image

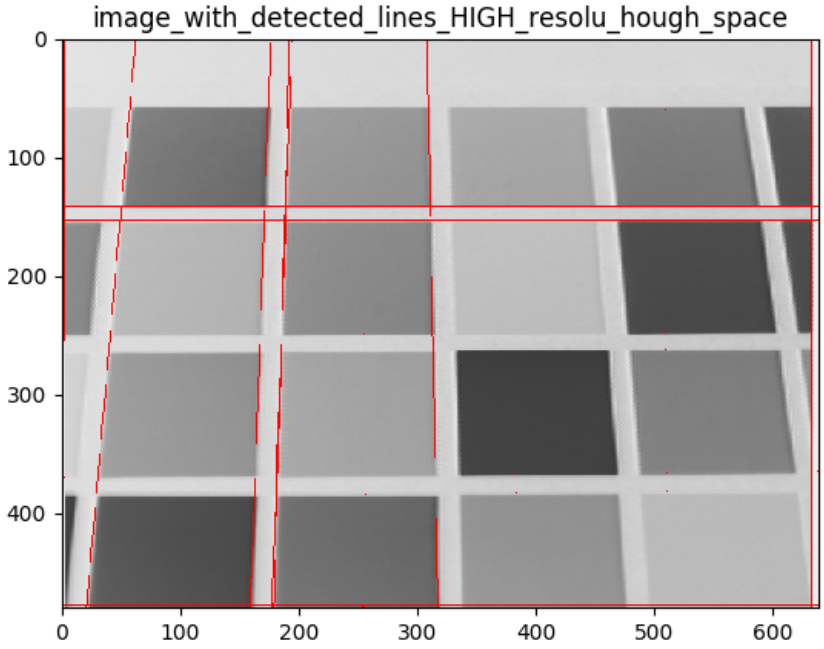

Figure 32: Combine the original image with the detected lines based on HIGH resolution hough space image

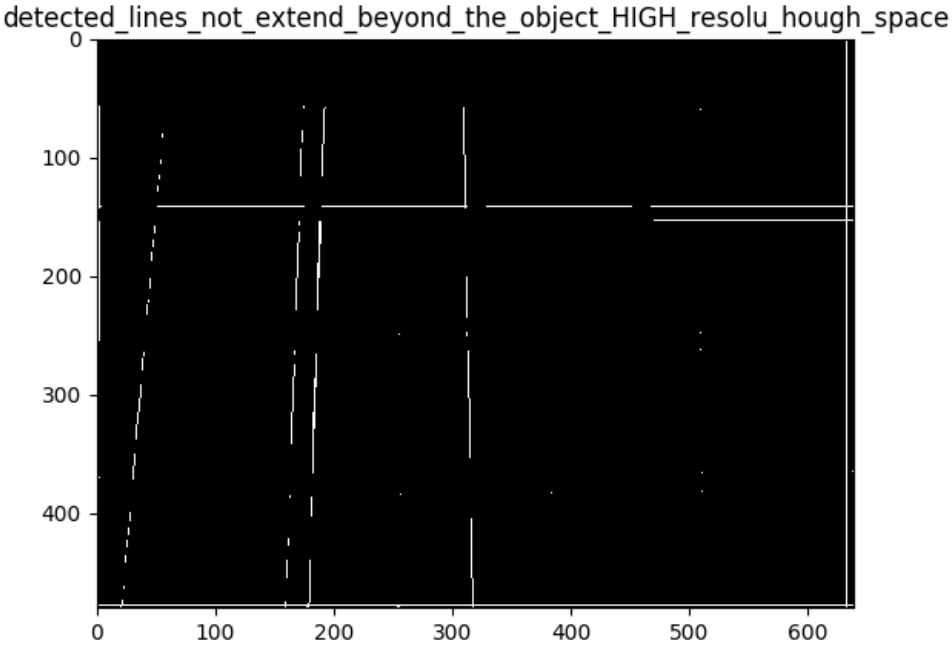

Figure 33: Detected lines not extend beyond the object based on HIGH resolution hough space image

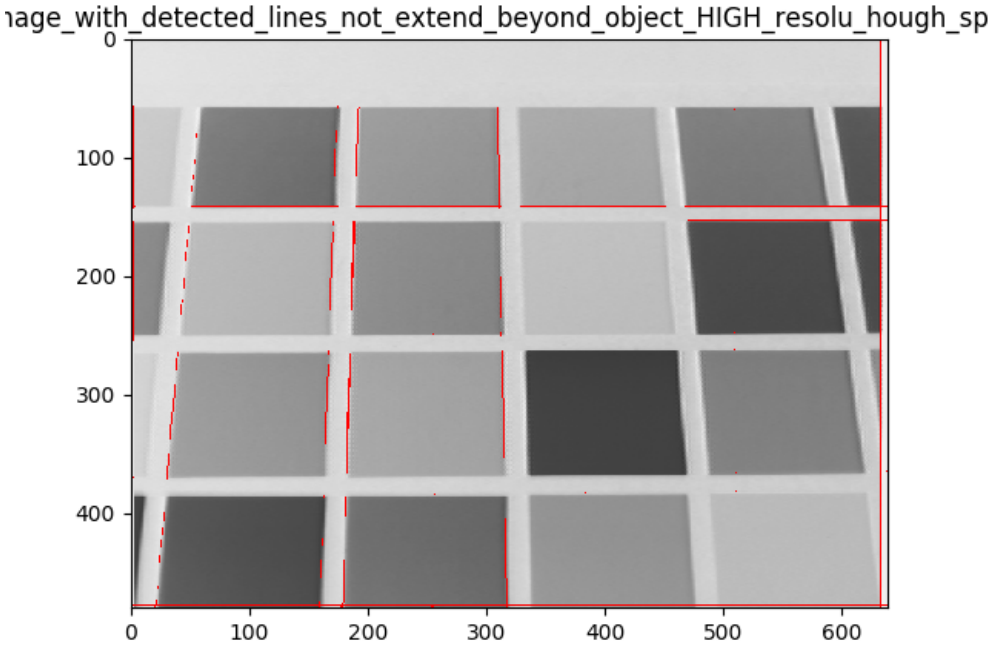

Figure 34: Combine the original image with the detected lines not extend beyond the object based on HIGH resolution hough space image

3.2.4 Submission for the hough complex 1 image

The results of the hough simple 1 are shown in this section.

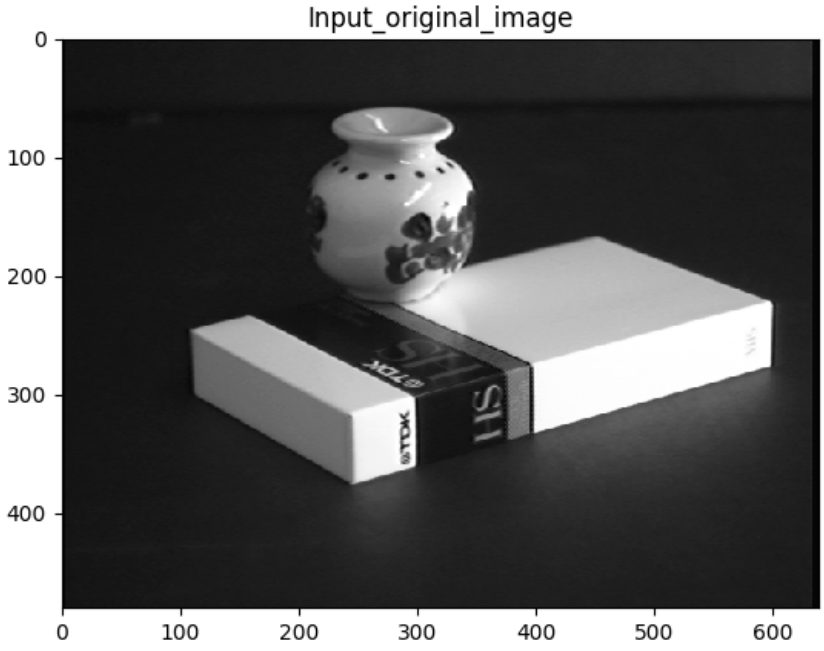

Figure 35: Input original image

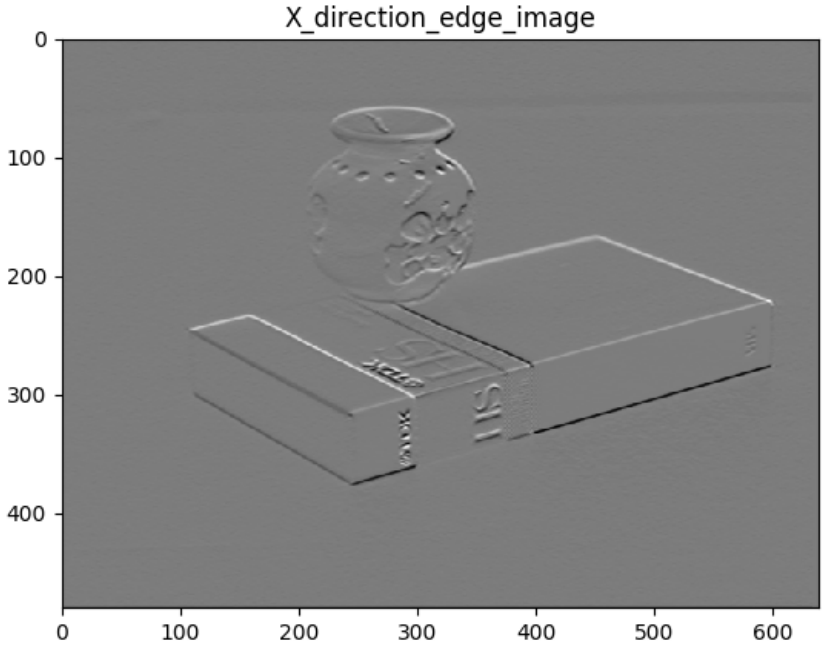

Figure 36: X direction edge image

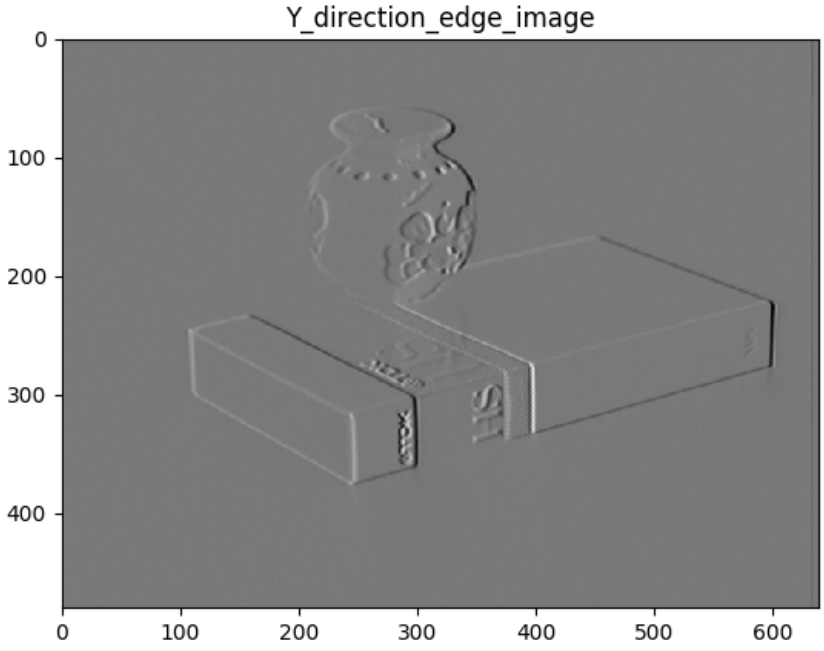

Figure 37: Y direction edge image

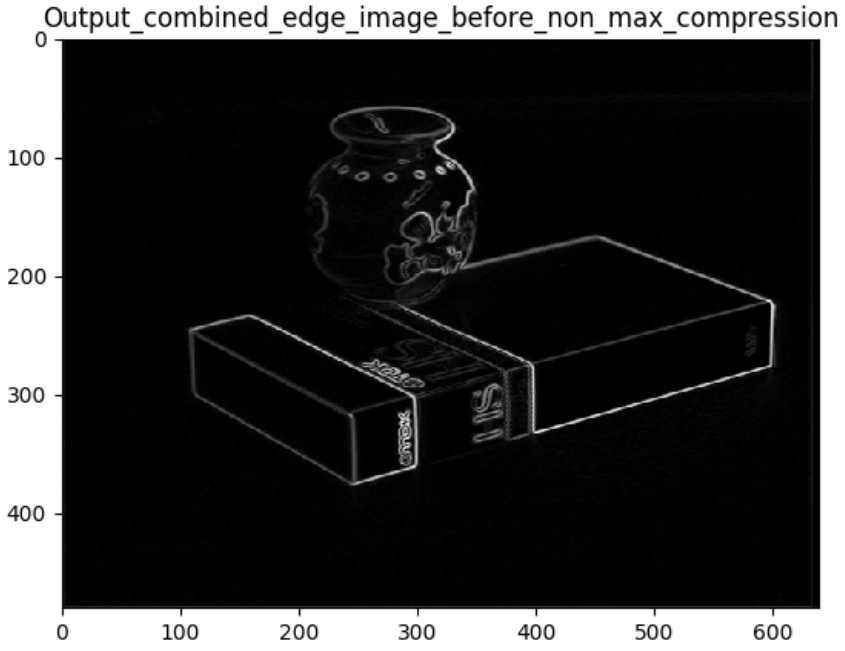

Figure 38: Output combined edge image before non max compression

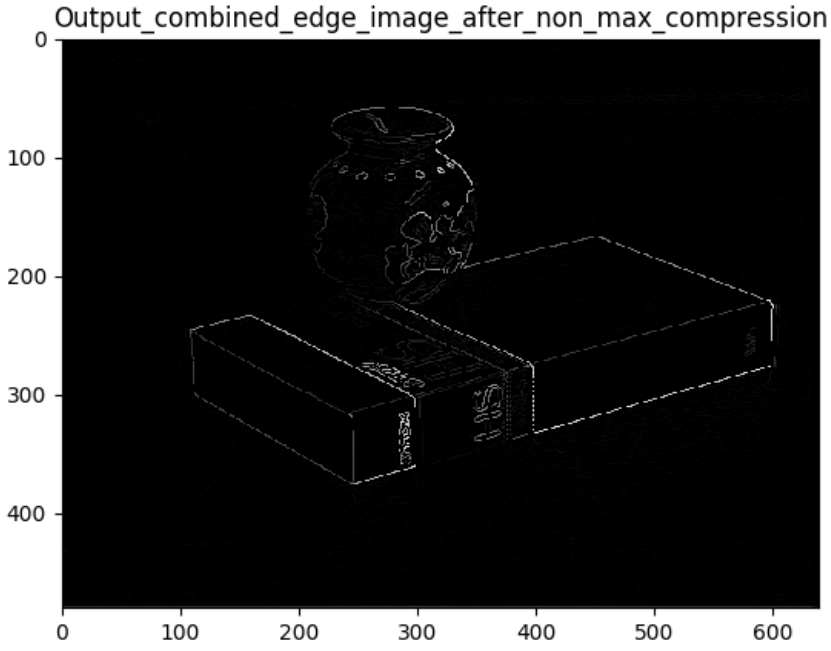

Figure 39: Output combined edge image after non max compression

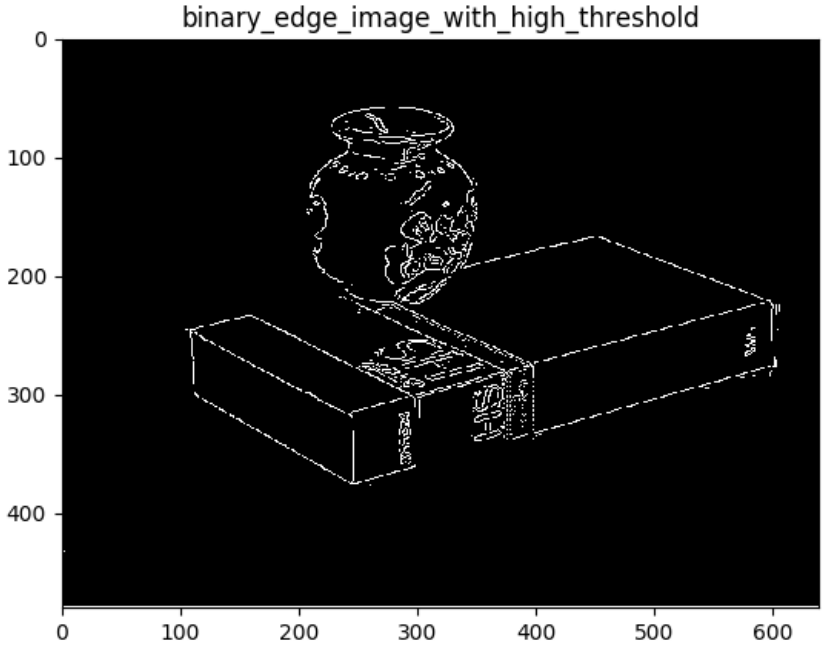

Figure 40: binary edge image with high threshold

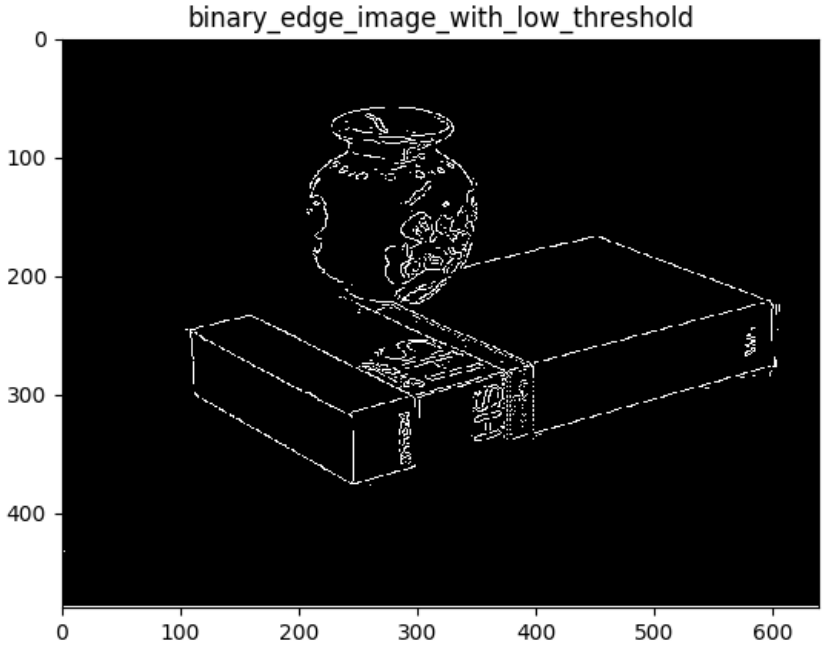

Figure 41: binary edge image with low threshold

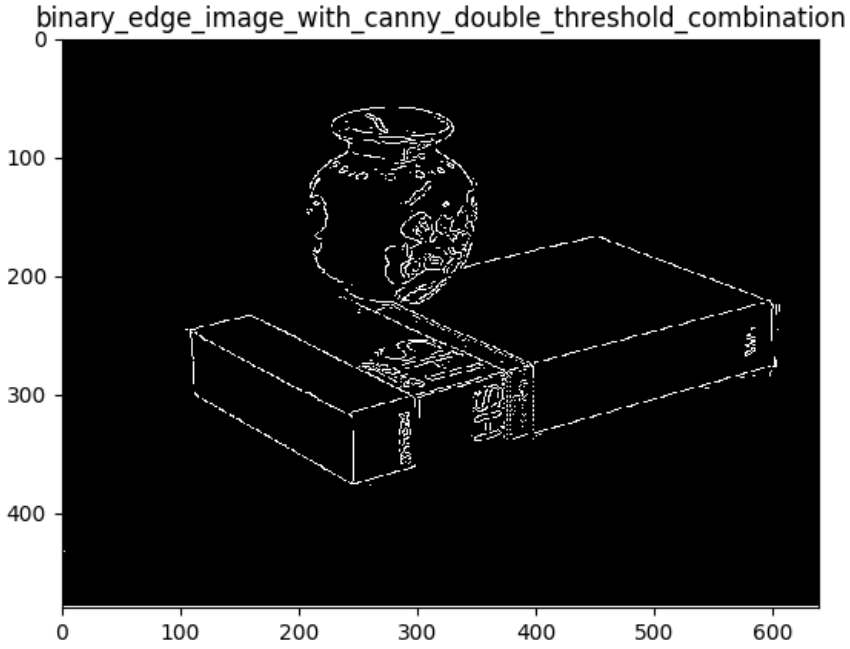

Figure 42: binary edge image by combining the low threshold and the high threshold based on the canny edge detection algorithm.

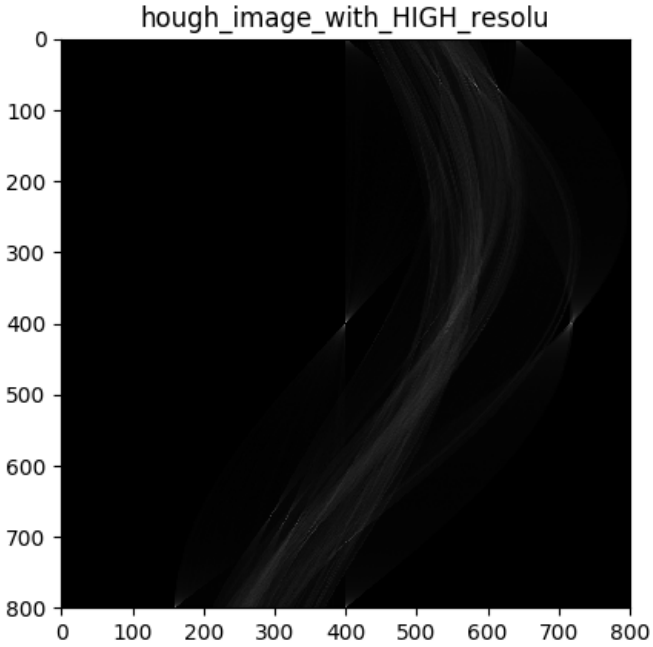

Figure 43: hough image with HIGH resolution

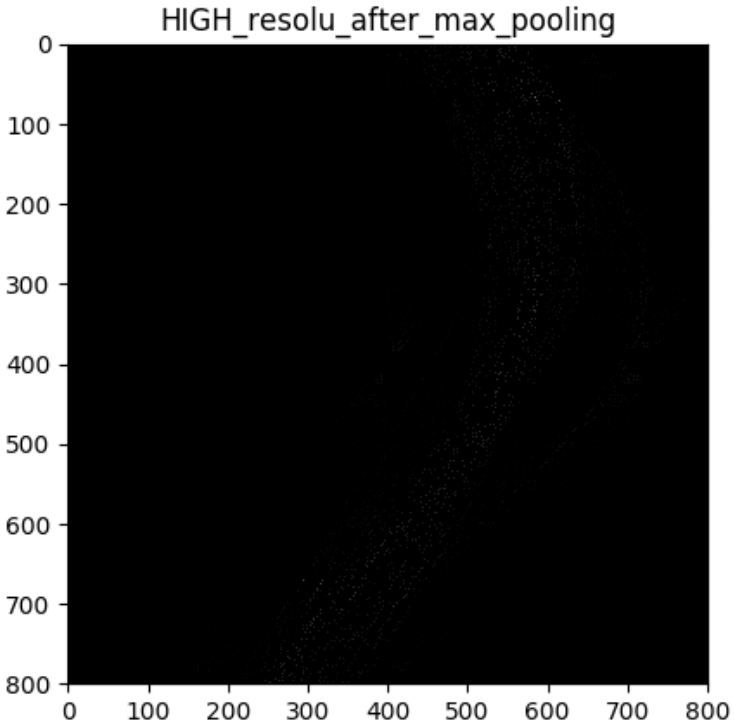

Figure 44: hough image with HIGH resolution after max pooling

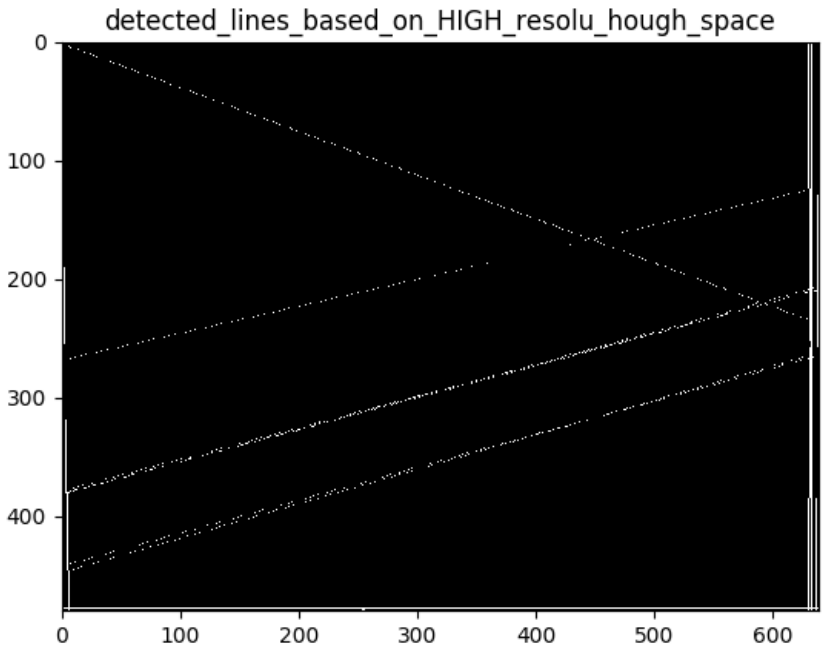

Figure 45: Detected lines based on HIGH resolution hough space image

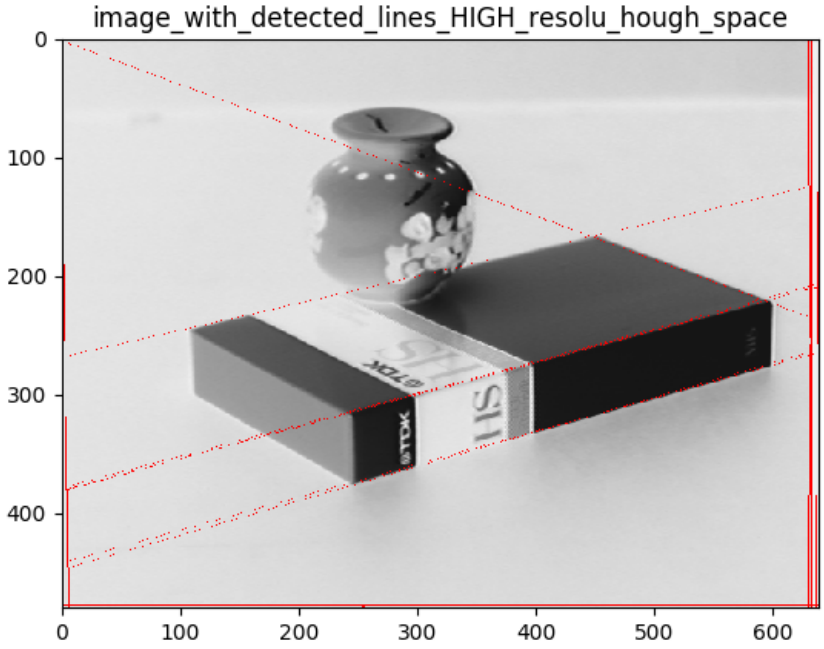

Figure 46: Combine the original image with the detected lines based on HIGH resolution hough space image

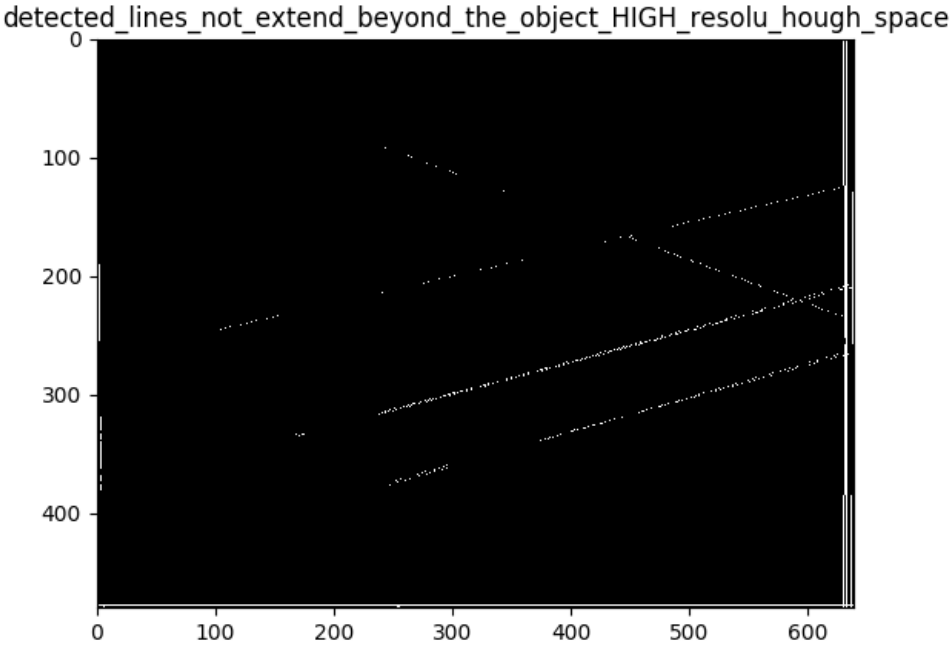

Figure 47: Detected lines not extend beyond the object based on HIGH resolution hough space image

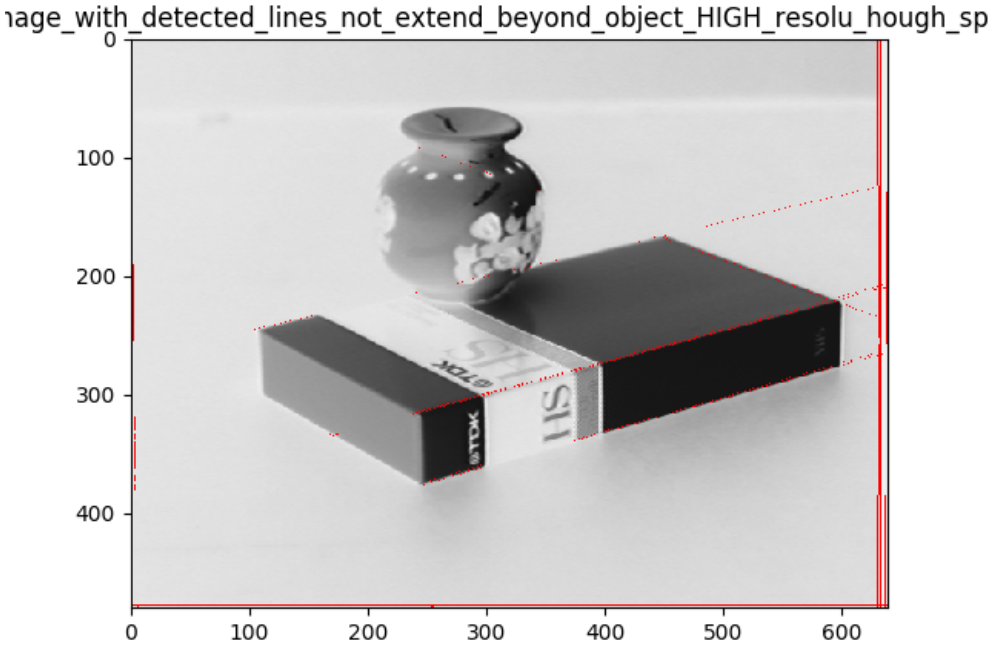

Figure 48: Combine the original image with the detected lines not extend beyond the object based on HIGH resolution hough space image