**Q2.1 Hough Transform Line Parametrization：**

1. Show that if you use the line equation $\rho = x \cos\theta + y \sin\theta$, each image point (x, y) results in a sinusoid in $(\rho, \theta)$ Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y).

**Answer:**
Define A as the amplitude, phi as the phase.
Since: A cos(theta – phi) = A cos(theta) cos(phi) + A sin(theta) sin(phi) , we can define:

ρ (rho) = A cos(theta – phi) = A cos(phi) cos(theta) + A sin(phi) sin(theta), and substitute ***x = A cos(phi), y = A sin(phi)***, and we will have the expression in form:
$\rho = x \cos\theta + y \sin\theta$
Since y/x = A sin(phi)/ A cos(phi) = tan(phi), then phi = arctan (y/x)

Since we have sin^2(phi) + cos^2(phi) = 1
A^2 = A^2*(sin^2(phi) + cos^2(phi)) = A^2* sin^2(phi) + A^2* cos^2(phi) (by the ***substitute*** step)
= x^2 + y^2

Hence, A = sqrt (x^2 + y^2). Therefore:
Amplitude = A = sqrt (x^2 + y^2)
Phase = phi = = arctan (y/x)

2. Why do we parametrize the line in terms $(\rho, \theta)$ instead of the slope and intercept (m, c)? Express the slope and intercept in terms of $(\rho, \theta)$.

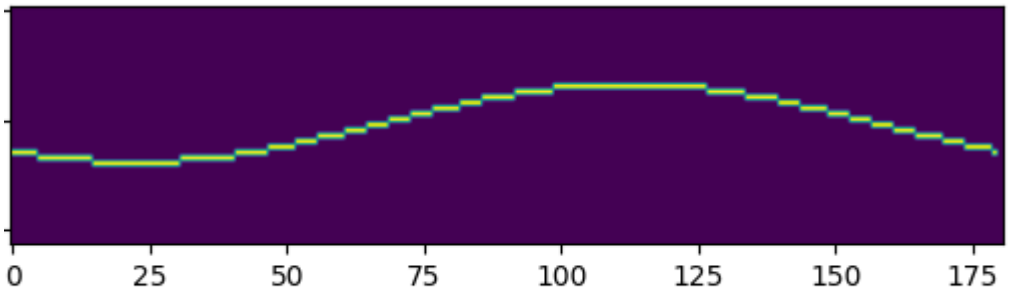
**Answer:** If we parametrize the in terms of slope and intercept, the space of the slope and the space of intercept is infinite. Basically, any number from $-\infty$ to $\infty$ can be possible for slope or intercept, so we have to maintain a huge ($\infty$ by $\infty$) Hough transform matrix to keep records of the "votes". However, the space of ρ, θ are both finite. ρ can only be a number in range (0, M), where M is the maximum possible length (usually the diagonal length of the input image). Θ can only be a number within 0 to 2pi.
Let y = mx + b
Intercept: $\rho/b = \sin\Theta$, $b = \rho/\sin\Theta$ (c)
Slope: $(-b/a) = (-\rho/\sin\Theta)/(\rho/\cos\Theta) = -(\cos\Theta/\sin\Theta)$ (m)

3. Assuming that the image points (x, y) are in an image of width W and height H, that is, $x \in [1, W]$, $y \in [1, H]$, what is the maximum absolute value of ρ, and what is the range for θ?

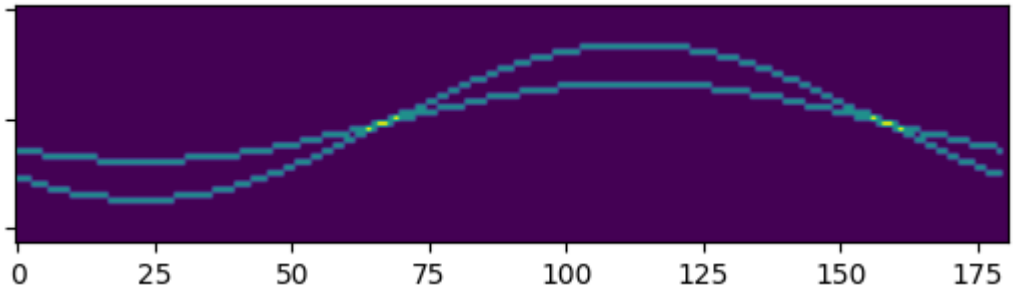
**Answer:** The maximum absolute value of ρ is: sqrt(W^2 + H^2), which is the diagonal length of the input image. The range for θ is 0 to 2pi.

4. For point (10, 10) and points (20, 20) and (30, 30) in the image, plot the corresponding sinusoid waves in Hough space, and visualize how their intersection point defines the line. What is (m, c) for this line? Please use Python to plot the curves and report the result in your write-up.

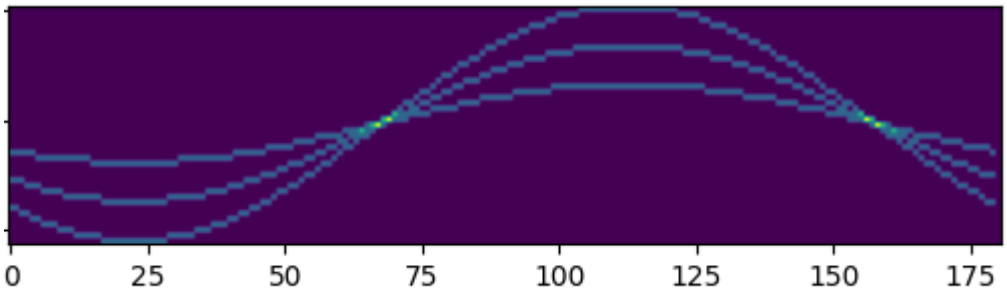
**Answer**: We have mapped the range (0, 2pi) to 0,180 on the x-axis (thetares = pi/90). For point (10,10), we have:




Then for point (20,20), we add a new curve:



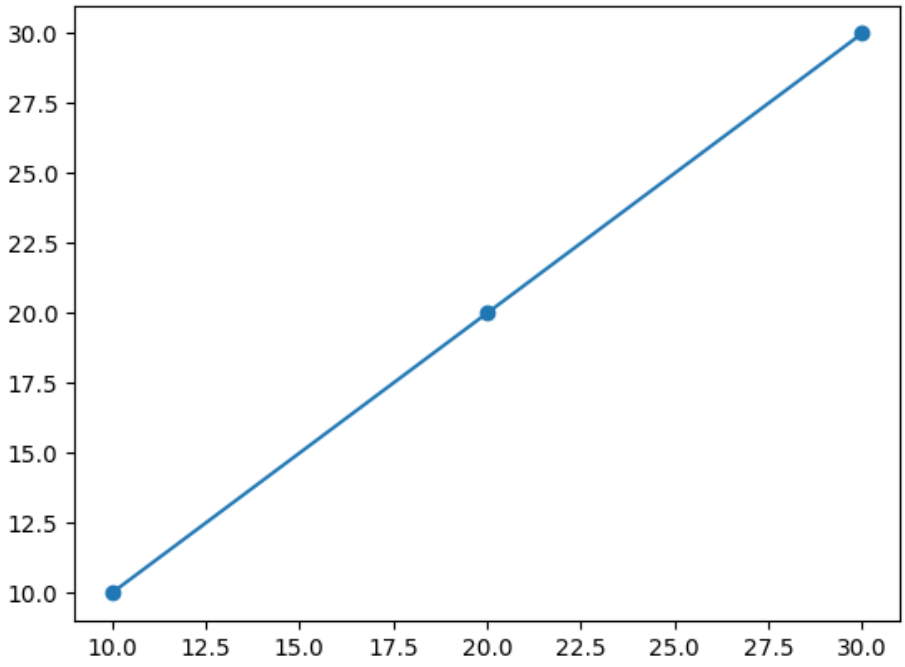

Then we add point (30,30):




The x axis is range (0,2pi); The y axis is (-m,m), where m is the maximum rho length (sqrt(30^2+30^2)). We observe the two interceptions are:

(0,67) and (0,157), which corresponds to approximately (0, 0.75pi) and (0, 1.75pi). With our formula in part2, we know that:

Slope: = -(cosΘ/ sinΘ), and if we plug in 0.75pi and 1.75pi, we know that the slopes are both one. Meanwhile, the rhos of intersections are both 0, indicating this is a line passing the origin. Hence, (m,c) = (1,0):

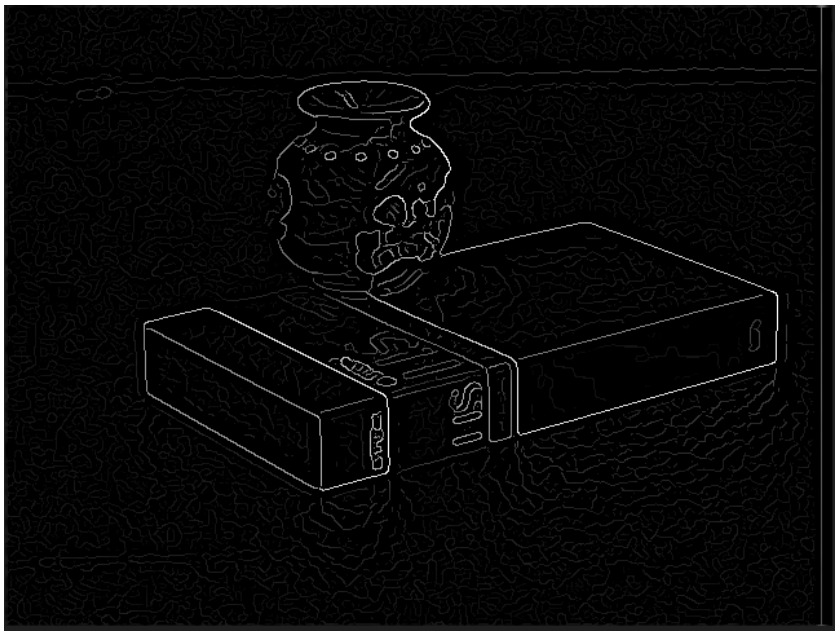



**Q4.1 Write-up**

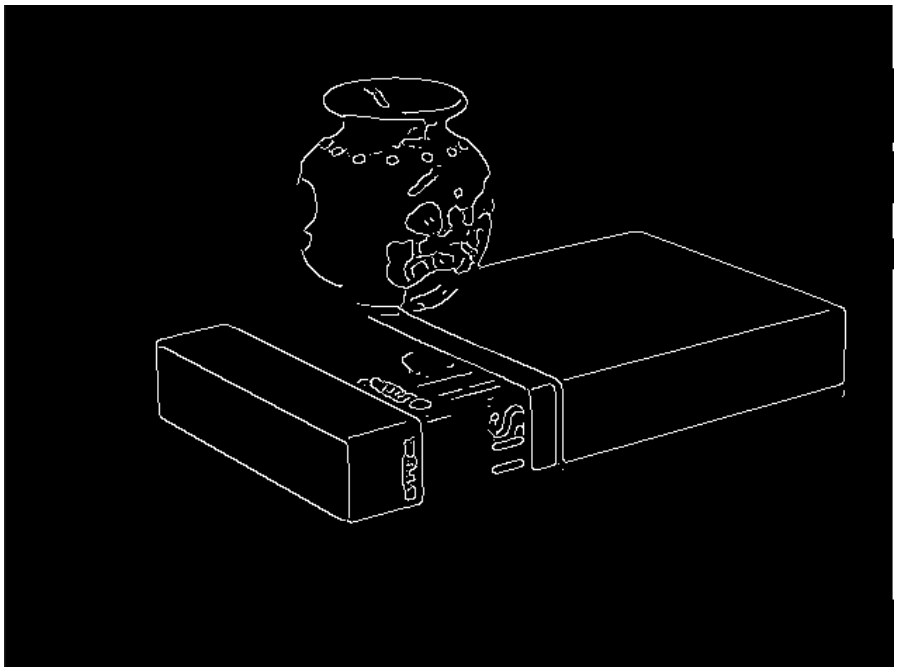
I select the second image, img2:

***ALL OF MY OUTPUT IMAGES (1-10) CAN BE FOUND IN THE RESULTS FOLDER***

*img02_01edge*: Basically, I first apply a gaussian filter to blur the input image and then find the derivative wrt x and y. After I figure out the magnitude and the angle, I to the angle mapping (NMS) and return the output:

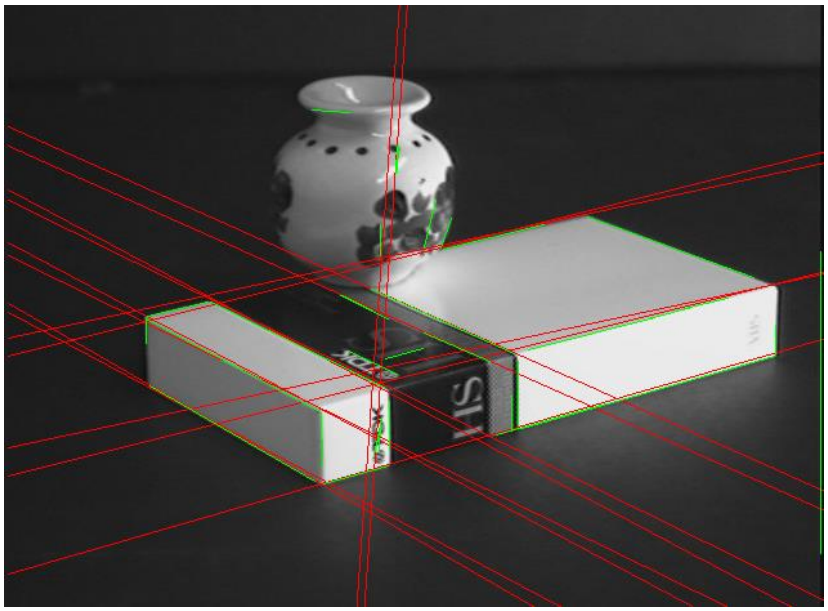


*img02_02threshold*: After I set a threshold at 0.03, I actually generate a ‘sharper’ image. Basically, it removes some vagueness and blurry part. The edge (white lines) are of significance and will be viewed as candidates for the Hough functions:

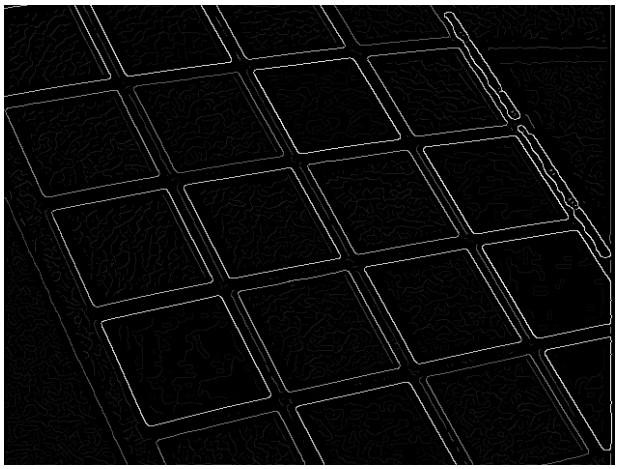


*img02_03hough:* It basically visualize the Hough matrix, which is a matrix that record the ‘votes’ in the parameter space. Its shape is like a sinusoid:

*img02_04lines*: Based on the output of the Houghtransform function, we apply NMS and select the top 15 (nlines) pair of parameters. We then convert these parameters into lines and put them on the original image:

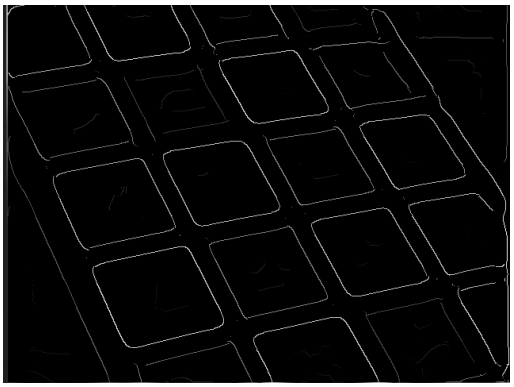


1. Did your code work well on all the image with a single set of parameters?
   Answer: Absolutely no. Based on the shape and geometry of our input matrix, we will have different parameters. For example, let's change the sigma from 2 to a larger number, like 10:
   Img1, sigma = 2:

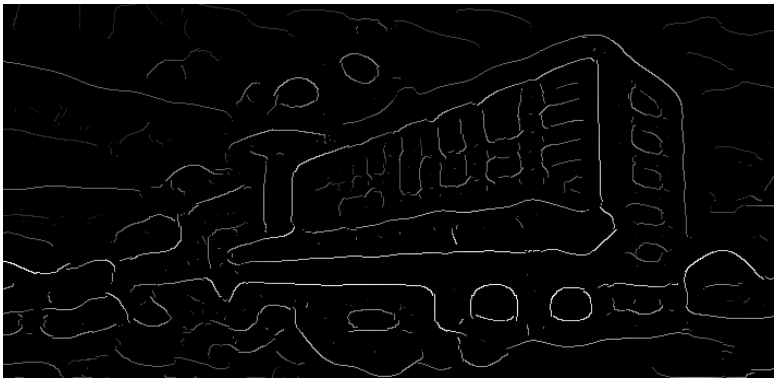


   Img9, sigma = 2:

Img1, sigma = 10:



Img9, sigma = 10:



We can see that by changing sigma from 2 to 10, the code still works pretty well in img1 but doesn't work on img9. Hence, a single set of parameters can not work well on all images

2. How did the optimal set of parameters vary with images?
   Answer: If the input image is relatively complicated, we cannot blur too much. We also need to make thetares and rhores smaller to adapt to the complexity. Hence, if:
   a. Delicate input like img9, small sigma (don't blur too much), smaller thetares and rhores (higher resolution), larger nlines (catch all the edges in hough lines).
   b. Concise image (straight lines) with some noise: higher sigma (blur a bit more), smaller nlines.

3. Which step of the algorithm causes the most problems?

For me, it should be the edge filter part. Firstly, I failed to normalize the gaussian filter at the first time, which leads to the failure of the threshold output. Meanwhile, the angle mapping could also cause troubles. Function like math.atan will generate unexpected values and impact the output. Numpy.arctan2 or other functions are safer choices. Meanwhile, the edge filter is the slowest part in the entire code because it has to iterate the entire input image and apply convolution. This is extremely inefficient.

4. Did you find any changes you could make to your code or algorithm that improved performance

   Potential improvements:
   a. Try padding more accurately. Currently I'm padding the input with edge values but this might not be a good choice if we actually have an edge on the boundary of input image.
   b. Faster soring: If we can record the positions of high magnitude pixels when we generate them, it will be great because it can prevent us from sorting the entire matrix in the hough line function.
   c. More accurate red lines with length limit: Currently, the red lines usually go through the entire image even though the edges don't. For example, we might have an edge in the middle of the image but it is relatively short. In that case, we will still generate a long red line to mark this edge, and this is ugly.

5. How well your code worked on different images:
   It works pretty well images that are simple and composed of straight lines. It does not work really well for images that are detailed and complicated like image 7 (fail to detect vertical edges) and image 9 (too detailed). The hough lines does not work really well on circle (image 10).

6. What effect do the parameters have?

   Sigma: The intensity of blurring. More sigma, more blurring.
   Threshold: The bar for selecting potential 'edge candidates'. Higher threshold, less edges selected (only edges with largest magnitude will be selected).
   rhoRes/thetaRes: Accuracy. If we make rhoRes/thetaRes smaller, the edge detected will be more accurate in terms of angles but it also takes longer to run the program.
   nLines: number of lines. If the image is very complicated, then we need to increase nLines in order to catch all edges in hough lines. If the image is simple, high nLine value will be redundant since there is not that much edges.