2018-17119 박수현 컴퓨터비전 hw2 writeup

1 Image Filters [5 points]

The image pixel array on the left is convolved ($*$) with what operator $\boxed{?}$ to give the result on the right. Specify the operator by numbers within an array, state its relationship to finite difference operators of specific orders, and identify what task this convolution accomplishes in computer vision.

size 10×8

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$* \boxed{?} \Rightarrow$ size 3×1

size 8×8

0	0	0	0	0	0	0	0
0	0	-1	-1	-1	-1	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	-1	-1	-1	-1	0	0
0	0	0	0	0	0	0	0

operator $\boxed{?} = \boxed{\begin{matrix} -1 \\ 2 \\ -1 \end{matrix}}$ (a 3×1 filter)

The operator is related to the second order finite difference, which is an approximation of the second order derivative:

$$f''(y) \approx \frac{f(y+h) - 2f(y) + f(y-h)}{h^2} \Rightarrow \boxed{\begin{matrix} 1 \\ -2 \\ 1 \end{matrix}}$$

The filter is in the form of the second order finite difference operator multiplied by -1.

The convolution task stated in the problem accomplishes horizontal edge detection. The detected edge region lies in between -1 and 1 in the result image pixel array.

2 Edge Detection [10 points]

The Laplacian of Gaussian (see figure 1) is often applied to an image $I(x, y)$ in computer vision algorithms, resulting in the function $h(x, y)$:

$$h(x, y) = \int_\alpha \int_\beta \nabla^2 \, e^{-((x-\alpha)^2+(y-\beta)^2)/\sigma^2} \, I(\alpha, \beta) \, d\beta \, d\alpha$$

where

$$\nabla^2 = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)$$

1. What image properties should correspond to the zero-crossings of the equation, *i.e.* those isolated points (x, y) in the image $I(x, y)$ where the above result $h(x, y) = 0$? [5 points]

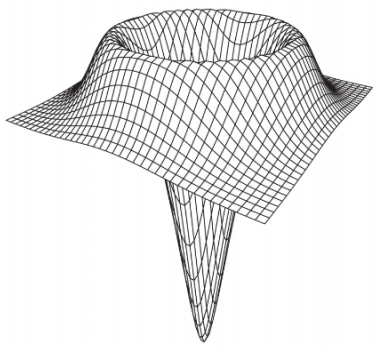


Figure 1: The Laplacian of Gaussian

The Laplacian of an image outputs regions of rapid intensity change, which are expected to be edges. A discrete convolution kernel can be constructed to approximate the second derivatives in the definition of the Laplacian. Since such second order differentiation is very sensitive to noise, smoothing the image with a Gaussian filter first and then applying the Laplacian highlights the desired edges better. $\Rightarrow \nabla^2(g * I)$

To speed up the process by performing only one convolution and to define points of rapid intensity changes into zero-crossings, a Laplacian of Gaussian operator is constructed (∵ associativity of convolution).

$\Rightarrow (\nabla^2 g) * I$

$h(x,y) = (\nabla^2 g) * I$ as stated in the problem. Therefore, through the LoG process, an edge point should correspond to isolated points (x,y) in the image $I(x,y)$ where $h(x,y) = 0$.

2. What is the significance of the parameter σ? If you increased its value, would there be more or fewer points (x, y) at which $h(x, y) = 0$? [5 points]

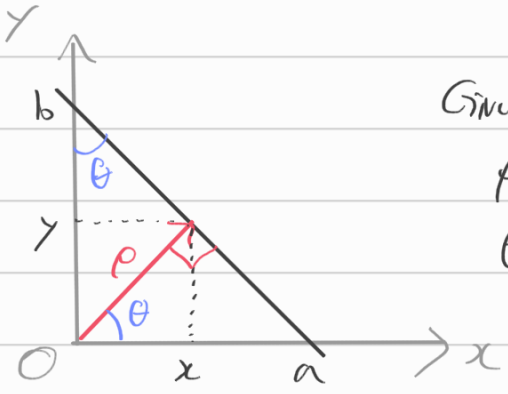
Parameter σ determines the size (spread) of the Gaussian smoothing, hence the degree of noise reduction of the input image.

A larger σ will reduce the noise, or in other words smooth out fine features, and preserve fewer but stronger edges.

Therefore, increasing σ would produce less edges, and since edges are points at which $h(x,y)=0$, there would be fewer points (x,y) at which $h(x,y)=0$.

3 Hough Transform Line Parameterization [10 points]

1. Show that if you use the line equation $\rho = x \sin\theta + y \cos\theta$, 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). [5 points]



Given line $\frac{x}{a} + \frac{y}{b} = 1$,

ρ = distance from the origin to the closest point on the line

θ = angle between the x-axis and the line connecting the origin with that closest point

The intuition is that every image point (x, y) in the line must be orthogonal to the straight line of length ρ that comes from the origin.

Since it is easy to derive

$\cos\theta = \frac{\rho}{a}$

$\sin\theta = \frac{\rho}{b}$,

the line $\frac{x}{a} + \frac{y}{b} = 1 \iff x\frac{\cos\theta}{\rho} + y\frac{\sin\theta}{\rho} = 1 \iff \rho = x\cos\theta + y\sin\theta$

Therefore, each line $\frac{x}{a} + \frac{y}{b} = 1$ of the image is associated with a pair (ρ, θ). Given a single image point (x, y), a straight line going through that point corresponds to a sinusoid in the (ρ, θ) Hough space.

$\rho = x\cos\theta + y\sin\theta$

$\rightarrow \rho = \sqrt{x^2+y^2}\left(\frac{x\cos\theta}{\sqrt{x^2+y^2}} + \frac{y\sin\theta}{\sqrt{x^2+y^2}}\right)$ ↓ multiply & divide by $\sqrt{x^2+y^2}$ (distance of point from origin)

$= \sqrt{x^2+y^2}\,(\sin\alpha \cdot \cos\theta + \cos\alpha \cdot \sin\theta)$ ↓ $\sin\alpha = \frac{x}{\sqrt{x^2+y^2}}$ $\cos\alpha = \frac{y}{\sqrt{x^2+y^2}}$

$= \sqrt{x^2+y^2}\,(\sin(\alpha+\theta))$

∴ amplitude $= \sqrt{x^2+y^2}$

phase $= \alpha = \arctan\left(\frac{x}{y}\right)$

2. Does the period (or frequency) of the sinusoid vary with the image point (x, y)? [5 points]

No.

A sinusoid relative to an image point (x,y) is expressed in $\rho = \sqrt{x^2+y^2}\,(\sin(\theta+\alpha))$ as shown above.

The period of the sinusoid is 2π and this value is invariant with the choice of (x,y).

2018-17119 박수현

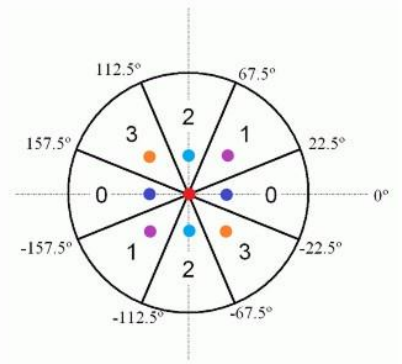
Supplementary materials for HW2_HoughTransform.py

EdgeDetection

Note: **the filter size** I choose is int(sigma * 6) + 1, because when computing a discrete approximation of the Gaussian function, pixels at a distance of more than 3σ have a small enough influence to be considered effectively zero.[1] Thus contributions from pixels outside that range can be ignored, and having a filter size of approximately sigma * 6 can derive a result sufficiently close to that obtained by the entire Gaussian distribution. I also added 1 to ensure that the filter size is an odd number.

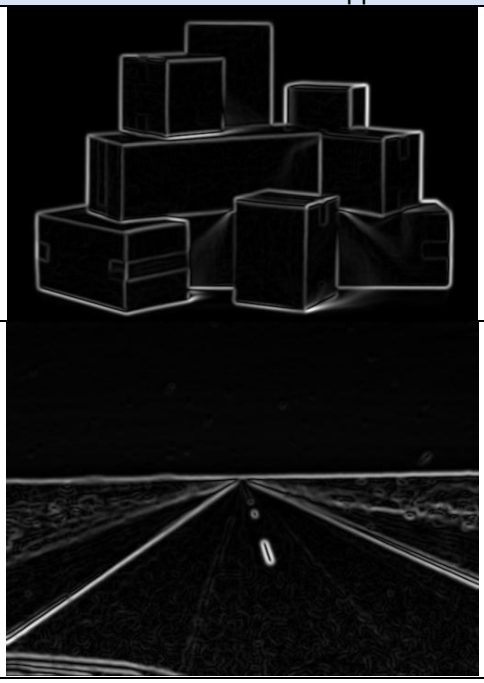
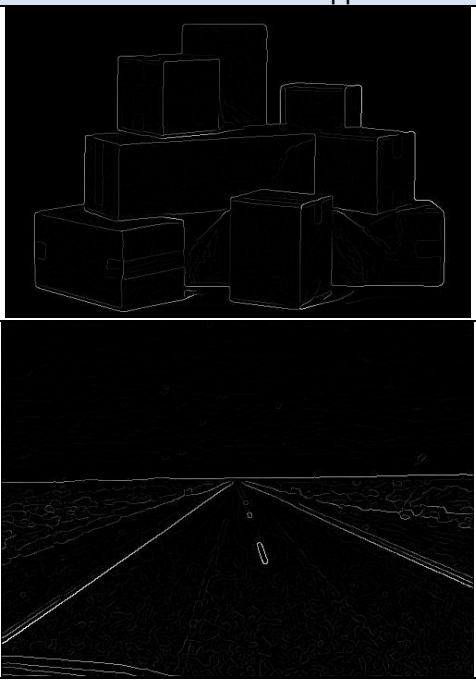
Non-maximum Suppression

A pixel is determined as an edge point if its gradient magnitude is locally maximum along the gradient direction. Non-maximum suppression is a method of comparing the gradient magnitude of the current pixel with those in positive and negative pixels to thin out the edges. With the current pixel in the center, a 3 by 3 grid of pixels can be divided into 8 sections. If the gradient direction falls in a certain range of angles, then we use two neighboring pixels that fall between the angle for comparison. Since the surrounding pixels lie in the horizontal or vertical direction, or along the positive or negative diagonal, there are 4 ranges of angles (in degrees) to determine the two pixels from: 0° ~ 22.5°, 22.5° ~ 67.5°, 67.5° ~ 112.5°, 112.5° ~ 157.5°, 157.5° ~ 180°, as illustrated in the figure below.[2]



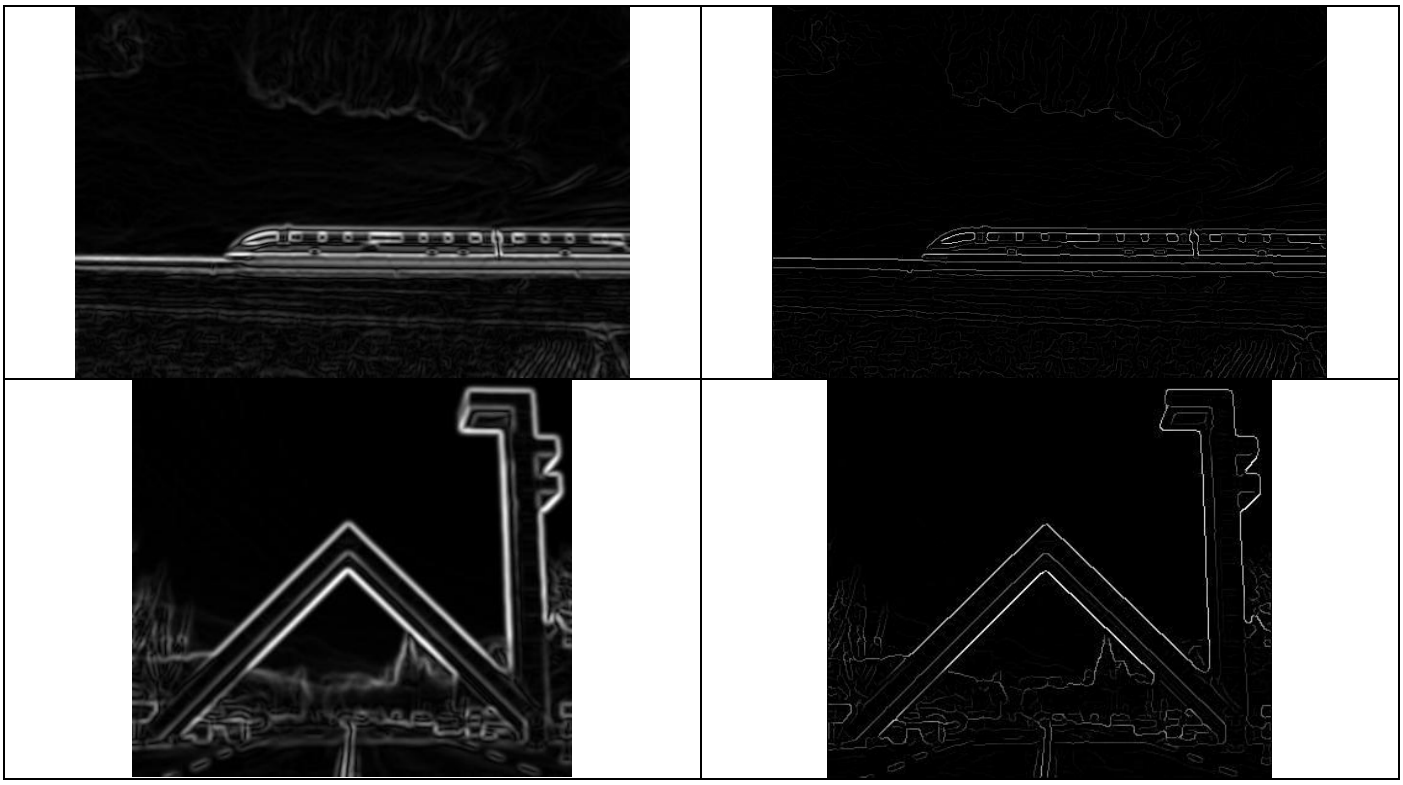
To implement the method in the code, I first compute the gradient orientation of all pixels in the image using the arctan operation, which gives an array of angles in radians that fall between the range [-pi, pi]. Next, I convert the radians into degrees with the range of [-180., 180.], and then add 180 to any negative values, ultimately deriving only positive values. This is because a negative angle has a symmetric positive angle, and the resulting neighboring pixels we want to see are same for both cases.

Below shows a comparison of the original gradient magnitude image and the result after non-maximum suppression, for each input image. Since the results look valid with the sigma set to the initial value 2, I didn't alter the sigma value during NMS.

`Im` before non-maximum suppression	`Im` after non-maximum suppression
	

[1] https://en.wikipedia.org/wiki/Gaussian_blur#:~:text=Typically%2C%20an%20image,entire%20Gaussian%20distribution

[2] Original image retrieved from https://climserv.ipsl.polytechnique.fr/documentation/idl_help/CANNY.html



Double Thresholding

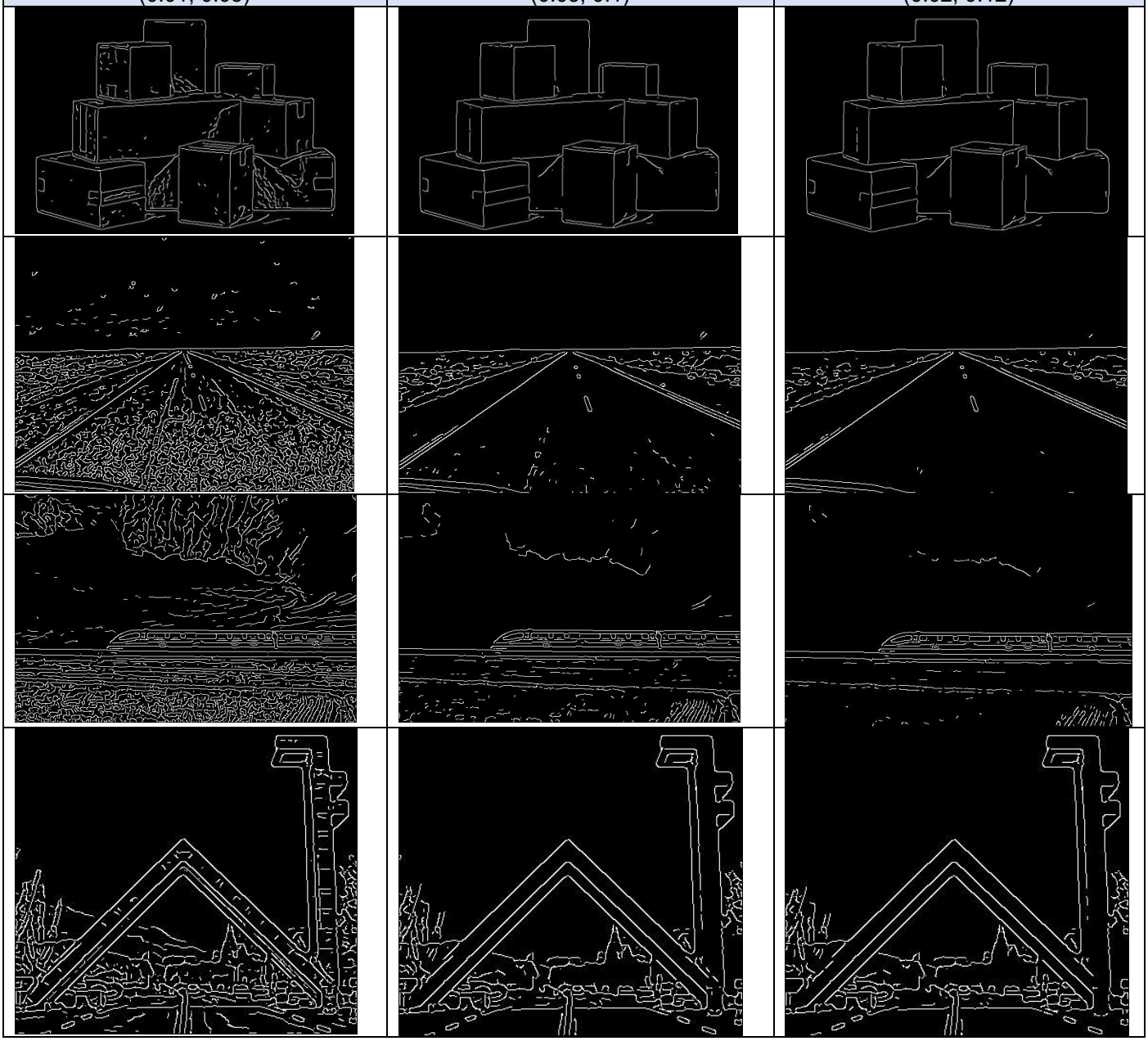
(lowThreshold, highThreshold)

Given the hyperparameters (`lowThreshold`, `highThreshold`) which are represented as the normalized pixel values in the range [0, 1], pixel values in Im equal to or bigger than the `lowThreshold` are seen as weak edges and are assigned `lowThreshold` as the value; pixel values equal to or bigger than `highThreshold` are seen as strong edges and are assigned `highThreshold` as the value; others aren't edge candidates and those values are set to zero.

For tracking edges using hysteresis, if there is a weak edge around a strong edge, I mark the weak edge as a strong edge, giving the `highThreshold` value. The strong edge is the center of attention because it is more intuitive to think that we are extending and connecting strong edges to fill in the reasonable areas. After every strong edge is inspected, I set the remaining weak edges containing `lowThreshold` values to 0, so that the resulting image is a cleaned-up version. The following are my experiments using different hyperparameter values:

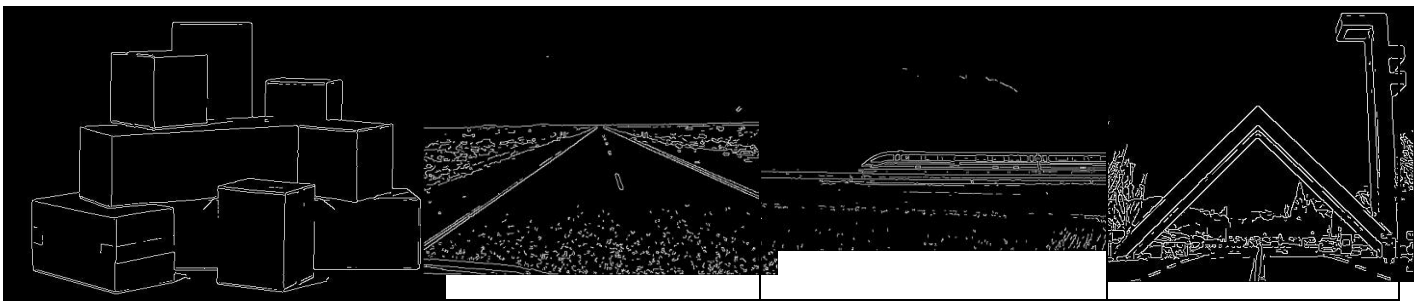
1. The initial setting given in the original python file (0.01, 0.03)
   - Too many unnecessary edges are marked, which means that the `lowThreshold` is too small to eliminate definitely-not-edges pixels and the `highThreshold` is also too generous.
2. Increasing both low and high thresholds (0.03, 0.1)
   - Edges become noticeably refined.
3. Tightening the threshold to (0.02, 0.12)
   - `highThreshold` higher than 0.12 eliminated major features of the image. After experimenting with +/- 0.01, I conclude that (0.02, 0.12) gives the clearest edges.

(0.01, 0.03)	(0.03, 0.1)	(0.02, 0.12)



sigma

I experimented more by changing the scale of Gaussian smoothing, from σ=2 to σ=1. By decreasing the sigma size, more detection of more fine features is expected. After a few tries increasing the highThreshold and tweaking the `lowThreshold`, some result images for σ=1 and (0.02, 0.13) are as follows. Although the fine edges detected for img04 are looking good, the results for img02 and img03 show too much fine-grained particles.

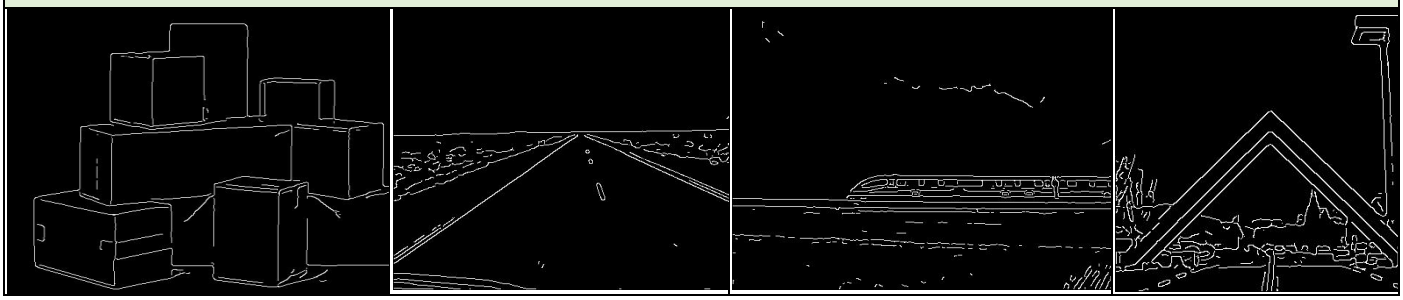


Therefore, I choose **`sigma=2, highThreshold=0.12, lowThreshold=0.02`** as hyperparameters.

2018-17119 박수현

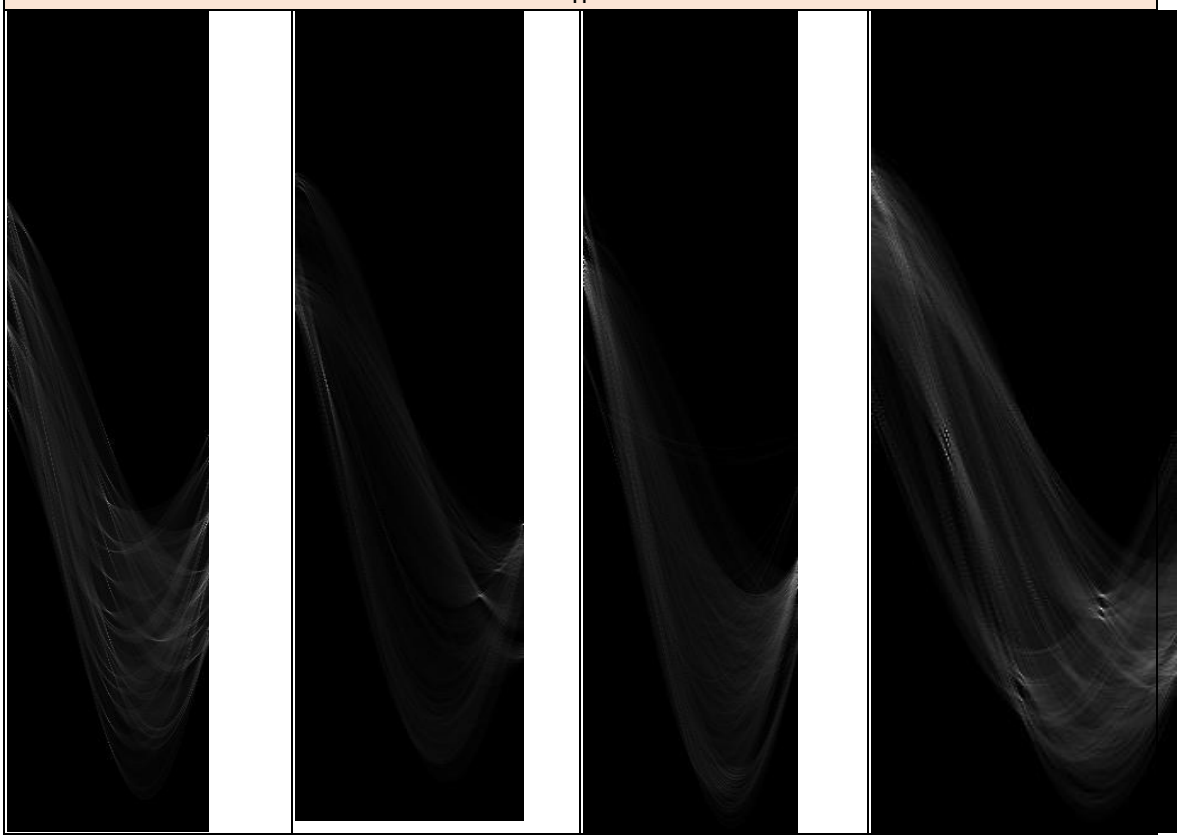
HoughTransform

Im



rhoRes=1, thetaRes=math.pi/180

H



HoughLines

Non-maximum Suppression

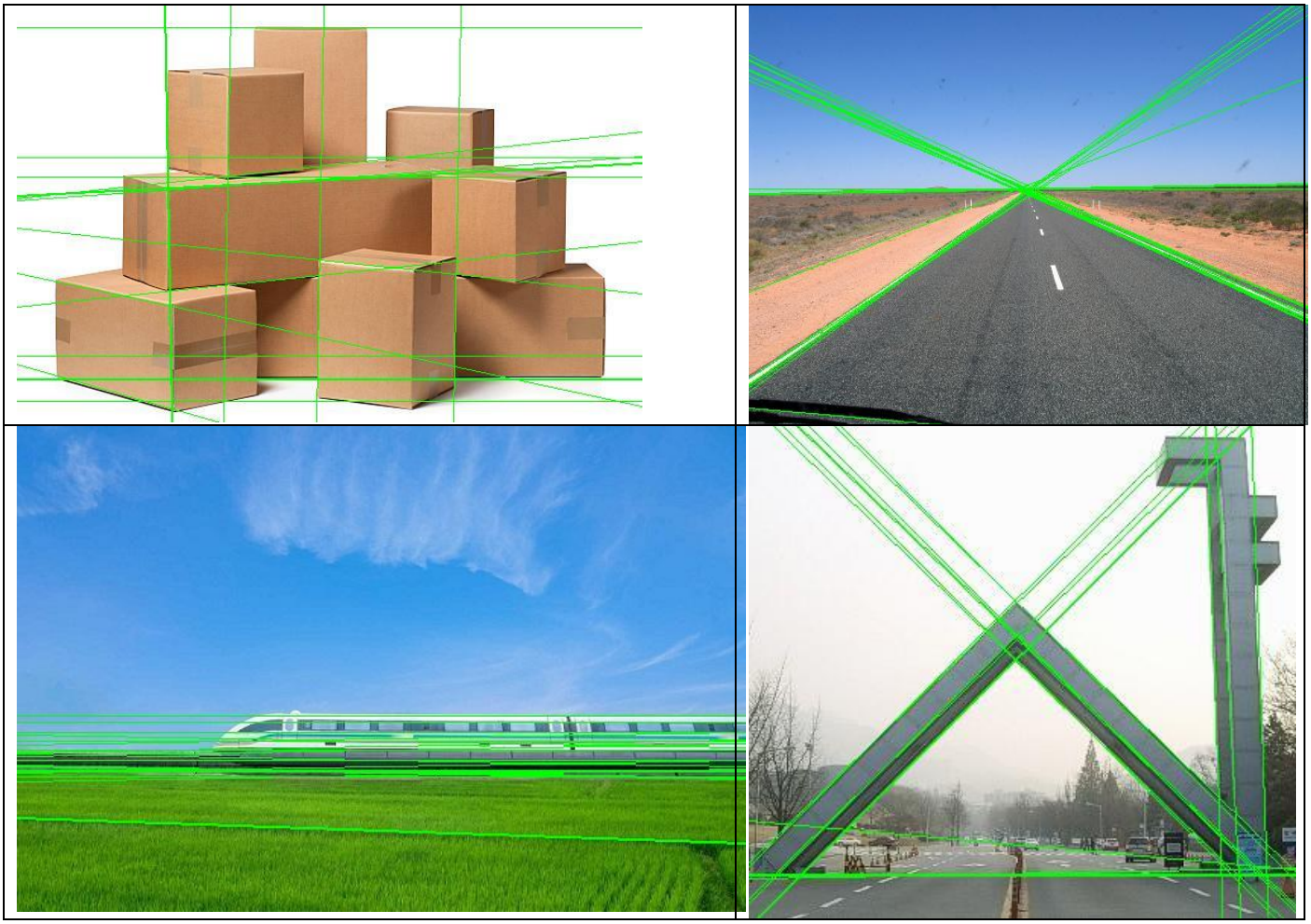
I implemented a simple non-maximum suppression. I first choose a point in the Hough space that has the highest vote. Then I investigate 8 neighbors of the point, leaving the center point as a valid Hough line only if it is a local maximum. This all-way comparison holds because unlike the edges in EdgeDetection where we need to consider the gradient orientation, we are finding the specific point that the sinusoids intersect the most. Also, suppressing neighbors of the local maxima to zero is also plausible, since each point in the Hough space refers to a line in the image space and some more lines too much in proximity with the chosen Hough line become unnecessary.

Role of `rhoRes` and `thetaRes` when plotting Hough lines

When plotting Hough lines, we need to retrieve the (x, y) image pixel coordinates corresponding to the (`rho`, `theta`) coordinates in the accumulator array. This means we use the line equation used in Hough transform, plugging in (`rho`, `theta`) values to obtain (x, y) in reverse. The original equation I use is `rho = int((x * cos_theta[t] + y * sin_theta[t]) + Im_diagonal) // rhoRes`, and I compute the y-value for a given rho and theta by `y1 = int((rho * rhoRes - Im_diagonal - x1 * cos_theta) / sin_theta)`. In addition to (`rho`, `theta`) values, `rhoRes` and `thetaRes` are required as well.

Results

`rhoRes=1, thetaRes=math.pi/180, nLines=20`



2018-17119 박수현

HoughLineSegments