LDW (Lane Departure Warning)

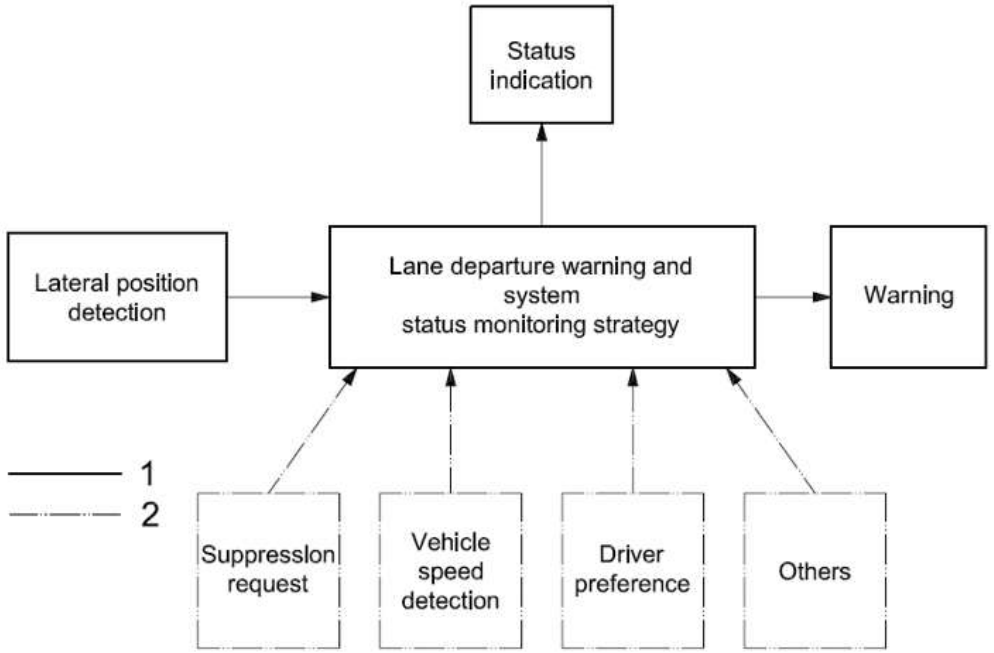
15 Jan 2021

자율주행시스템 개발팀
신 주 석

◆ LDW (Lane Departure Warning) System

- **Main Focus: Help the driver keep the vehicle in the lane on highways and highway-like roads**

- **운전자의 부주의로 인하여 차선 이탈 시 운전자에게 알려주기 위해 경고를 발생 시키는 System**
  - » 다른 차량과의 충돌에 대해 경고하거나 차량의 움직임을 제어하기 위한 System은 아님
  - » 차량의 제어는 운전자에게 책임이 있음



1 Minimum requirement.
2 Optional function.

- **Lane Detection in image (from Camera) ⇒ LDW ⇒ LKAS ⇒ HDA ⇒ 자율 차선 변경**

(ISO 17361: Intelligent transport systems — Lane departure warning systems - Performance requirements and test procedures)

◆ Finding Lane Lines on the Road

– 차선을 인식(검출)하기 위해 어떤 특징들이 좋은 걸까?
  - » Color
  - » Shape
  - » Orientation
  - » Position in the image

- **Finding Lane Lines on the Road**
  - **Identifying Lane Lines by Color**

R G B

Pure White = [?, ?, ?]

- **Install Pycharm**

pip install numpy (np : array)

" matplotlib (plot)

" OpenCV-Python (opencv)

" pillow (plt. —)

" moviepy (video)

◆ Color Selection

– Write Code

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
```

```
image = mpimg.imread('test.jpg')
color_select = np.copy(image)
```

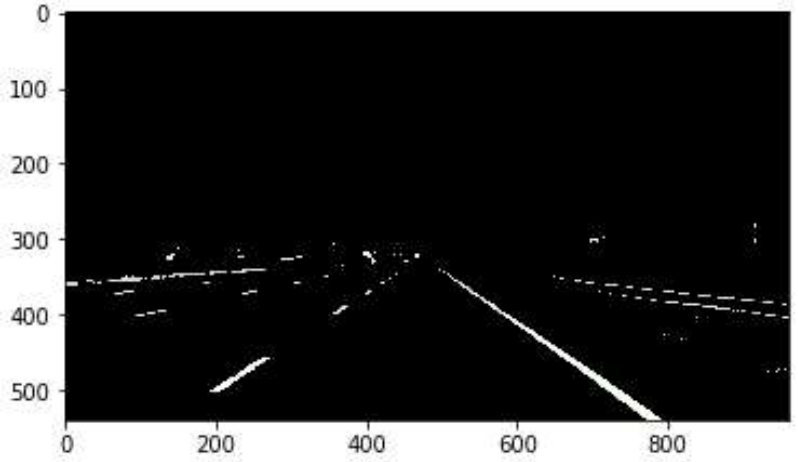
```
r_th = 0
g_th = 0
b_th = 0
rgb_th = [r_th, g_th, b_th]
```

⟸ Modify these values

```
th = (image[:,:,0] < rgb_th[0])\
   | (image[:,:,1] < rgb_th[1])\
   | (image[:,:,2] < rgb_th[2])

color_select[th] = [0, 0, 0]
```

```
plt.imshow(color_select)
plt.show()
```

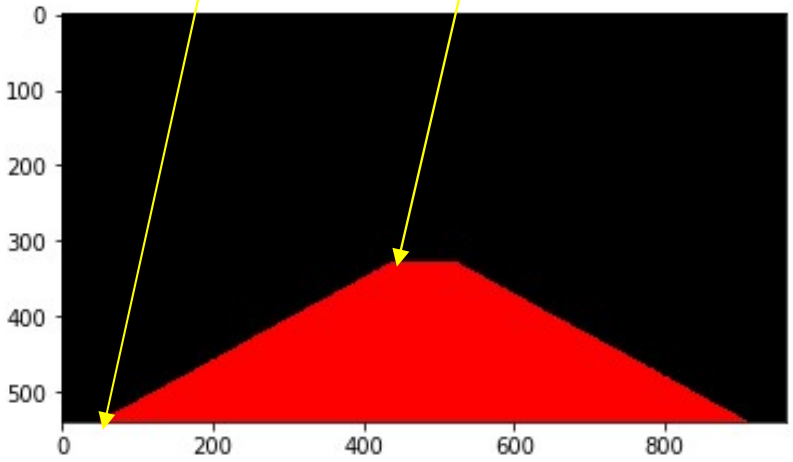


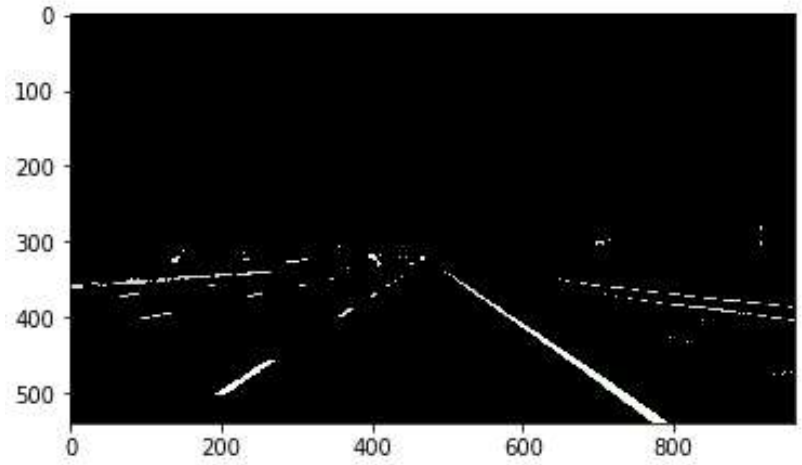
◆ ROI (Region Of Interest)

– Make Mask

```
import cv2
```
← 추가

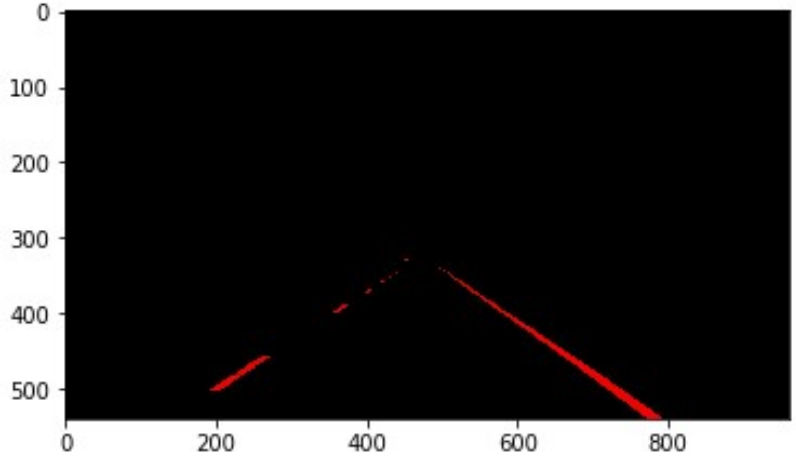
```
#th_img = cv2.cvtColor(color_select, cv2.COLOR_RGB2GRAY)
mask = np.zeros_like(color_select)
ignore_mask_color = 255
vertices = np.array([[(50, height), (width/2-45, height/2+60),
                      (width/2+45, height/2+60), (width-50, height)]], dtype=np.int32)
cv2.fillPoly(mask, vertices, ignore_mask_color)
plt.imshow(mask, 'gray')
plt.show()
```





Bitwise AND? OR?

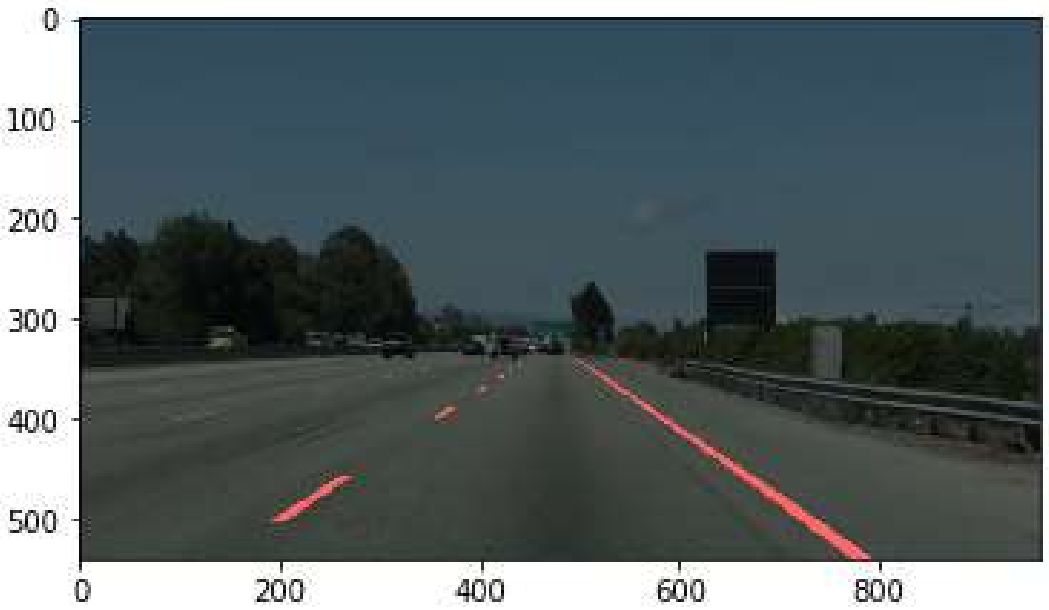
Img0 = cv2.bitwise_?(color_select, mask)



- **Color Select + ROI**

```
result = cv2.addWeighted(image, 0.4, img0, 1, 0)
plt.imshow(result)
plt.show()
```

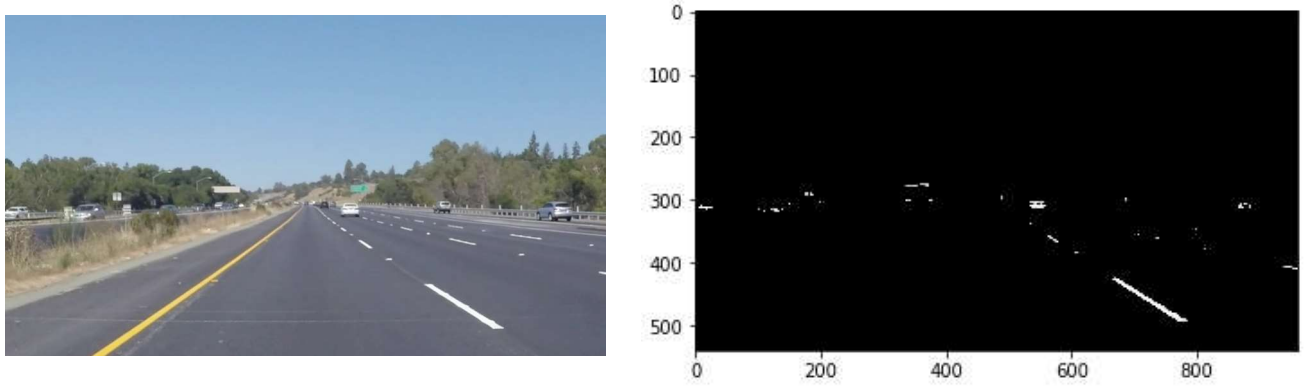
원본이미지 Color Select + ROI (mask)



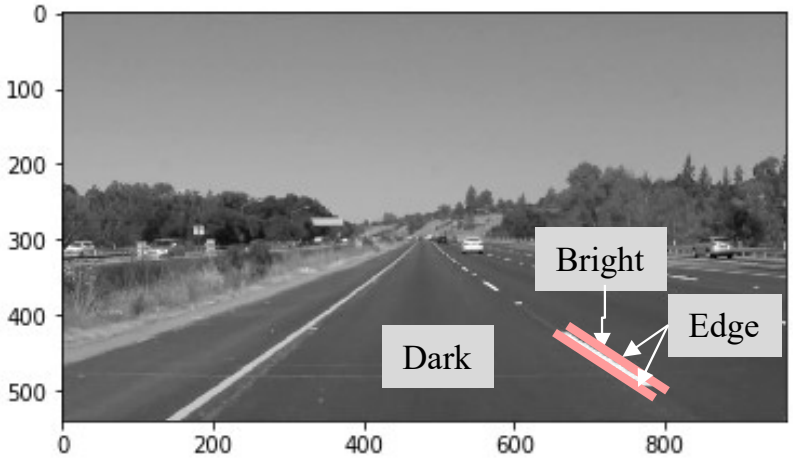
So, Can we upload
this algorithm to the Car?

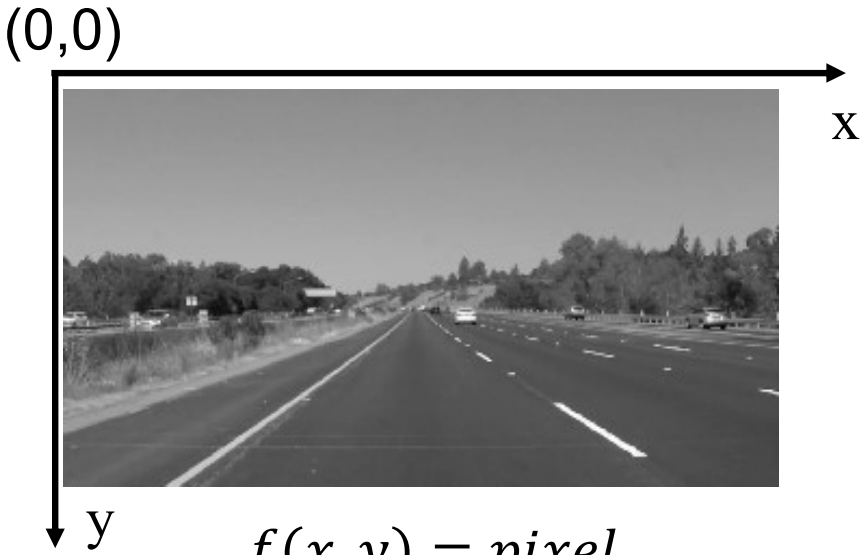
◆ Using Edge

– **Result (using color: same code)**



– **What is edge?**

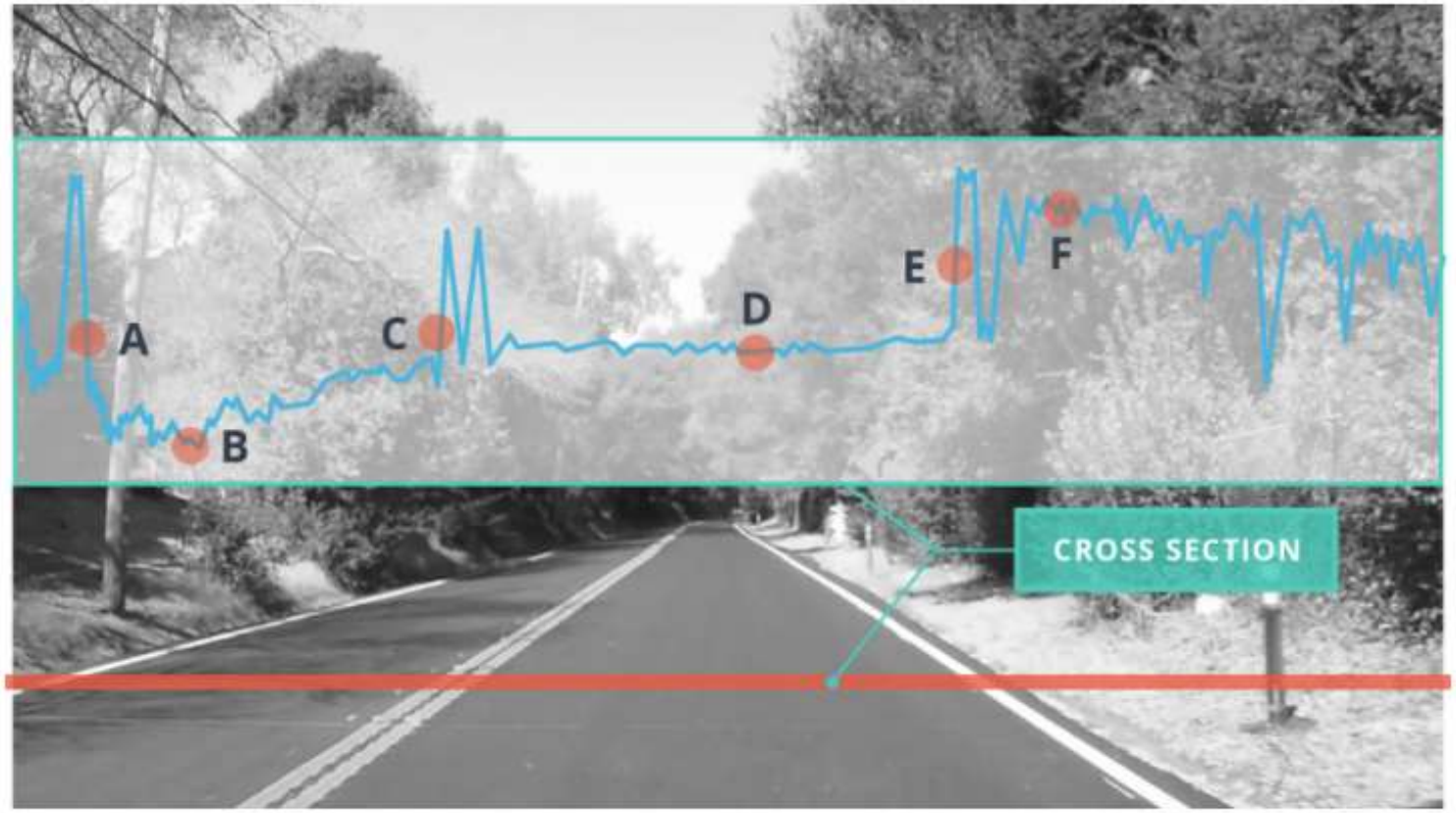




$$f(x,y) = pixel$$

$$\frac{df}{dx} = \Delta(pixel)$$

- **Using Edge**
  - **Where you expect to find strong edge?**

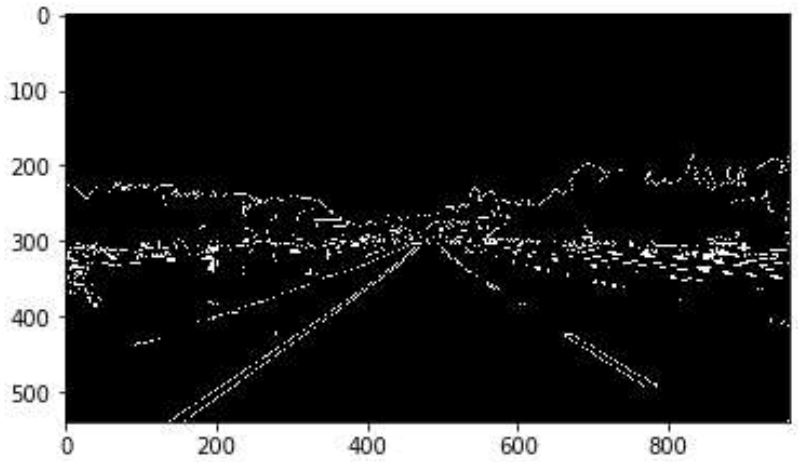


- **Canny Edge**

```
low_th =  0
high_th = 0
edge_img = cv2.Canny(smooth_img, low_th, high_th)
plt.imshow(edge_img, 'gray')
```

Modify these values

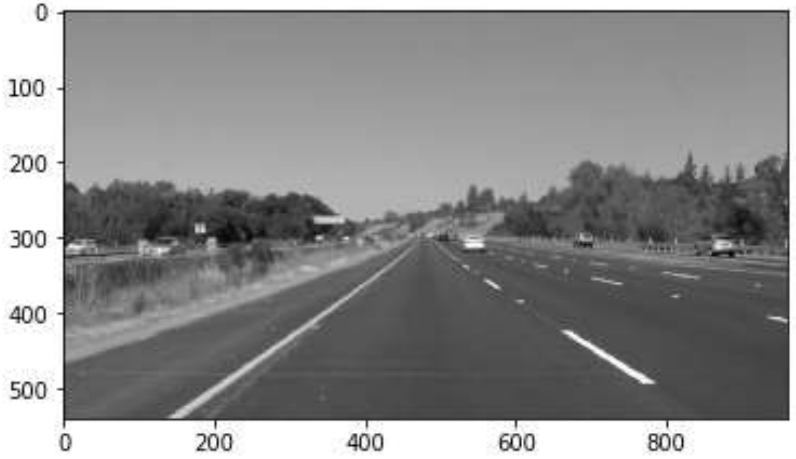
```
<matplotlib.image.AxesImage at 0x7f7e3796e6d8>
```




```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
```

```
image = mpimg.imread('test_1.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

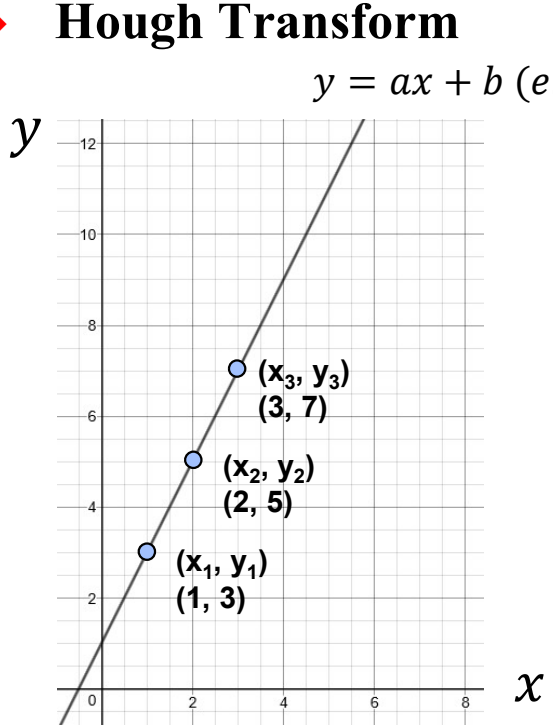
```
kernel_size = 3
smooth_img = cv2.GaussianBlur(gray, (kernel_size, kernel_size), 0)
plt.imshow(smooth_img, 'gray')
plt.show()
```

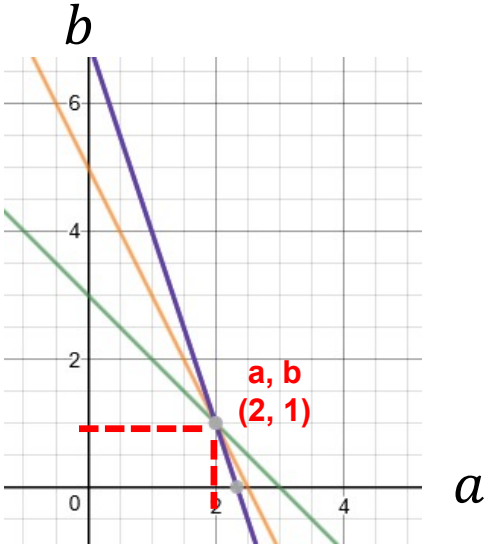


Hough Transform

$y = ax + b\ (e.g.\ y = 2x + 1)$ ⇒ $b = -xa + y$

$b = -1a + 3$
$b = -2a + 5$
$b = -3a + 7$





Hough Transform
Use the following equation to express a straight line.

$$\rho = xcos\theta + ysin\theta$$

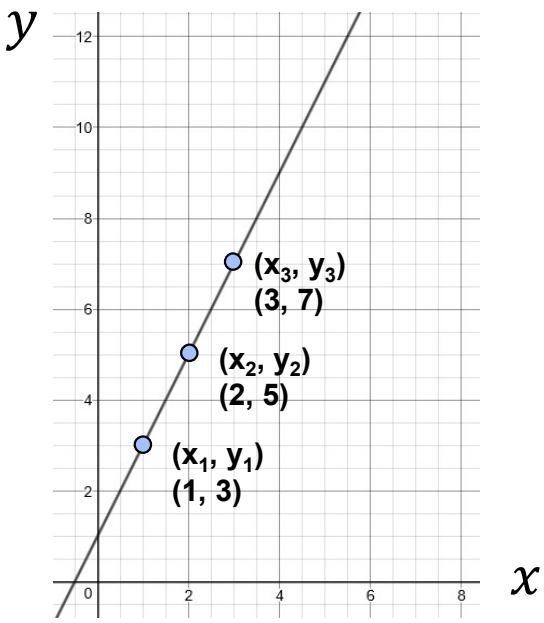
Why polar coordinate is used in Hough Transform?

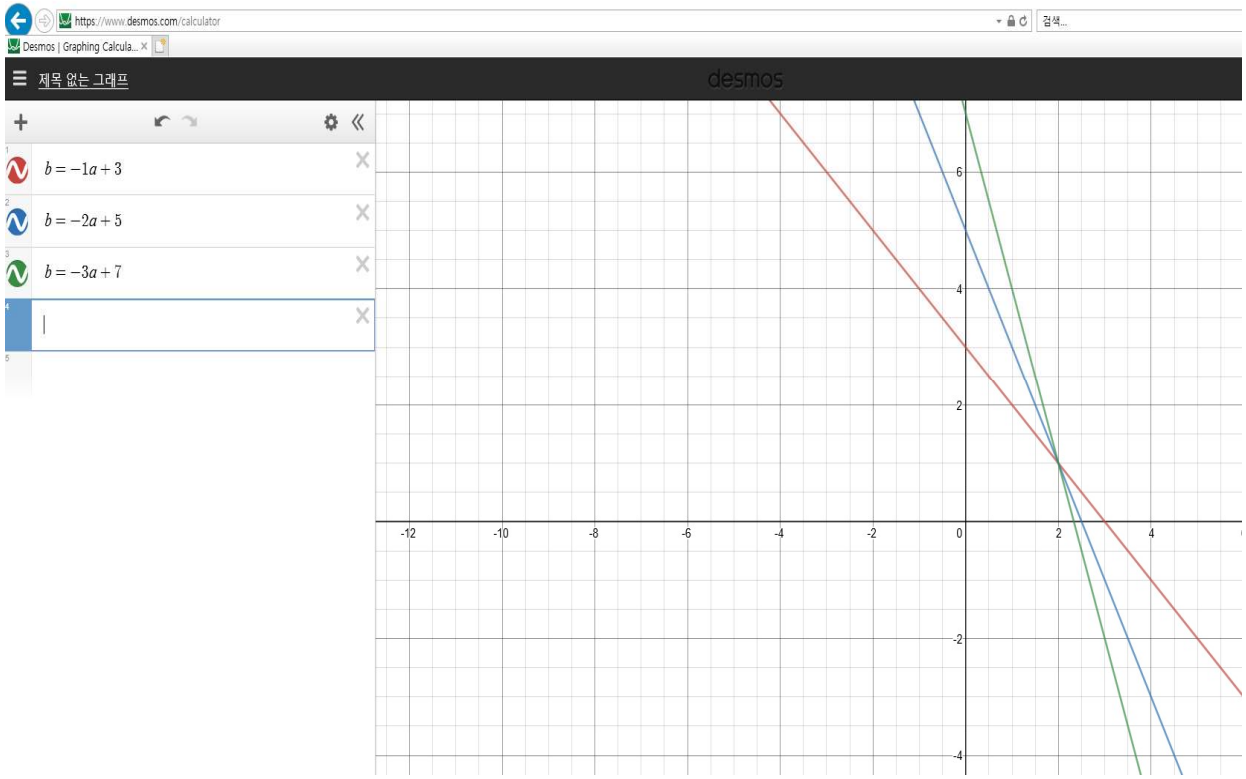
⇒ **Vertical lines have infinite slope in a-b representation**

◆ Hough Transform

$$y = ax + b \ (e.g.\ y = 2x + 1) \Rightarrow b = -xa + y$$

$$b = -1a + 3$$
$$b = -2a + 5$$
$$b = -3a + 7$$



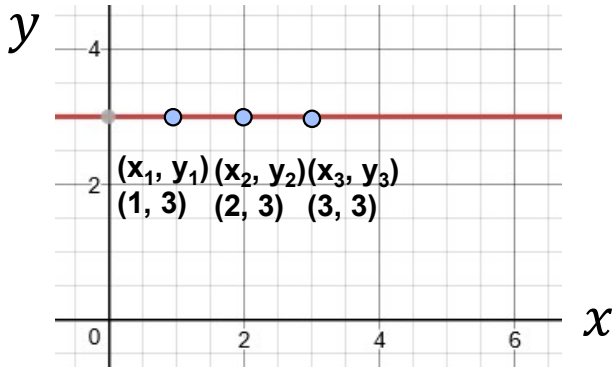


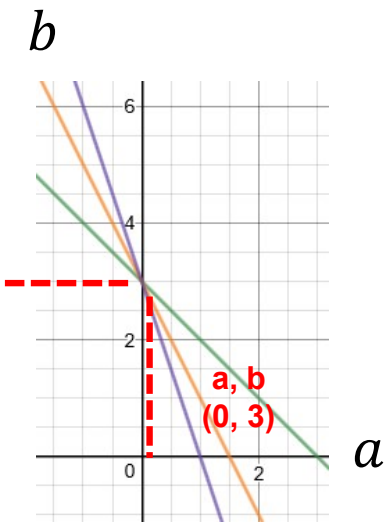
https://www.desmos.com/calculator/

◆ Hough Transform

$y = ax + b\ (e.g.\ y = 3)$ ⇒ $b = -xa + y$

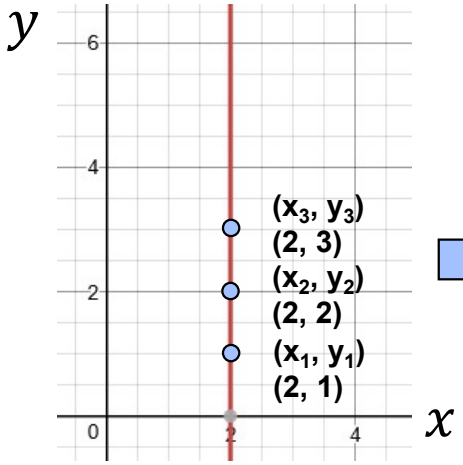
$b = -1a + 3$
$b = -2a + 3$
$b = -3a + 3$

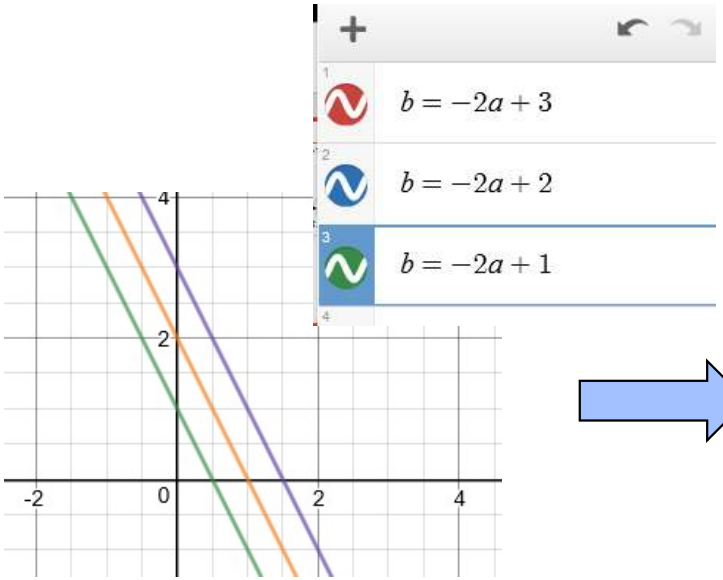




Horizontal line is OK

How about the Vertical Line?

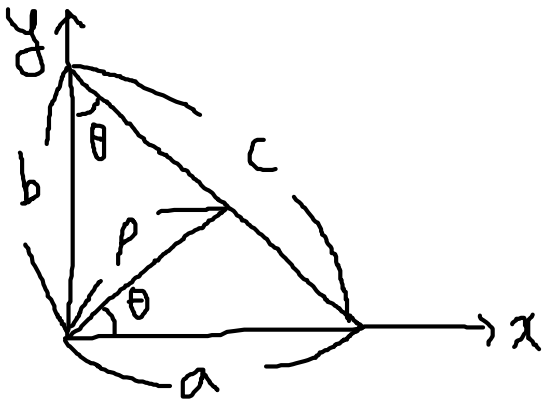




$\rho = x\cos\theta + y\sin\theta$

◆ Hough Transform

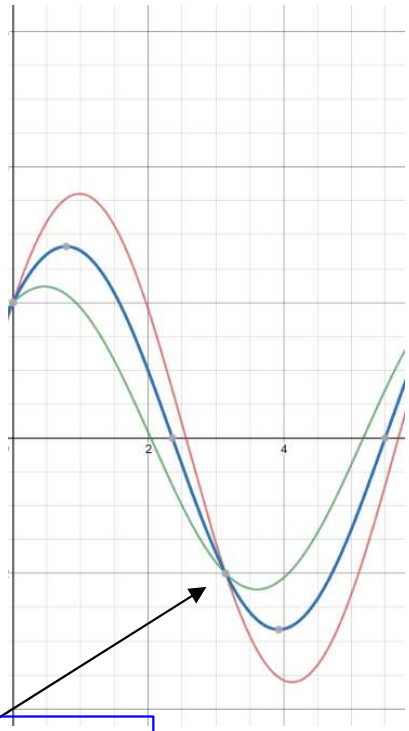
$$\rho = x\cos\theta + y\sin\theta$$



(x_3, y_3) $(2, 3)$ $\rho = 2 \cdot \cos\theta + 3 \cdot \sin\theta$

(x_2, y_2) $(2, 2)$ $\rho = 2 \cdot \cos\theta + 2 \cdot \sin\theta$

(x_1, y_1) $(2, 1)$ $\rho = 2 \cdot \cos\theta + 1 \cdot \sin\theta$



교점 발생

$$y = -\frac{b}{a}x + b$$

$$\sin\theta = \frac{a}{c}$$

$$\cos\theta = \frac{b}{c}$$

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

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

$$-\frac{b}{a} = -\frac{\cos\theta \cdot c}{\sin\theta \, c} = -\frac{\cos\theta}{\sin\theta}$$

$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$

$$\rho = \left(\frac{\cos\theta}{\sin\theta}x + y\right)\sin\theta$$

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

x절편과 y절편이 주어졌을 때 직선의 방정식

$$y = -\frac{b}{a}x + b$$

◆ Hough Transform

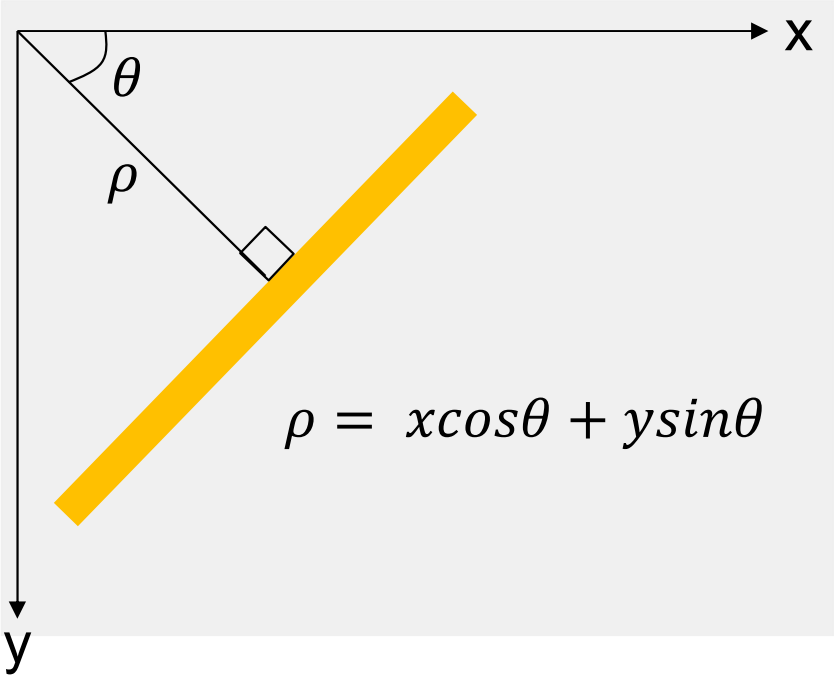
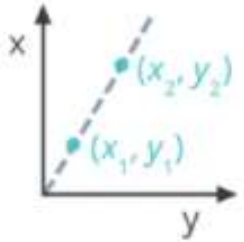
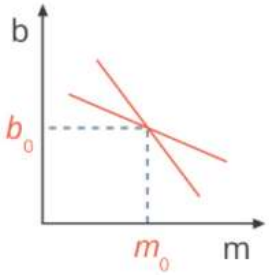
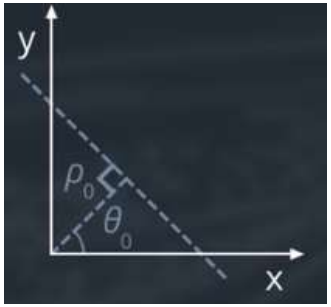


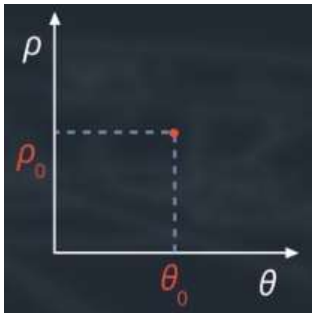
Image space



Hough space








```
cv2.HoughLinesP(image, rho, theta, threshold, minLineLength, maxLineGap)
```

image: input image (edge 영상)
rho: rho 값의 범위 (rho 값을 얼만큼 변경하면서 찾을 것인가)
theta: theta 값의 범위
threshold: 만나는 점의 개수 (많을수록 차선일 확률이 높음)
minLineLength: 직선의 최소 길이 (이 값보다 작으면, 찾고자하는 직선으로 간주 하지 않음)
maxLineGap: 찾은 직선들에서 직선끼리의 거리가 이 값 이상일 경우, 다른 직선으로 간주

◆ Hough Transform

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2

image = mpimg.imread('test_1.jpg')
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)

kernel_size = 3
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)

low_threshold = 50
high_threshold = 150
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)

mask = np.zeros_like(edges)
ignore_mask_color = 255

imshape = image.shape
vertices = np.array([[(0,imshape[0]),(450, 290), (490, 290), (imshape[1],imshape[0])]], dtype=np.int32)
cv2.fillPoly(mask, vertices, ignore_mask_color)
masked_edges = cv2.bitwise_and(edges, mask)
```

◆ Hough Transform

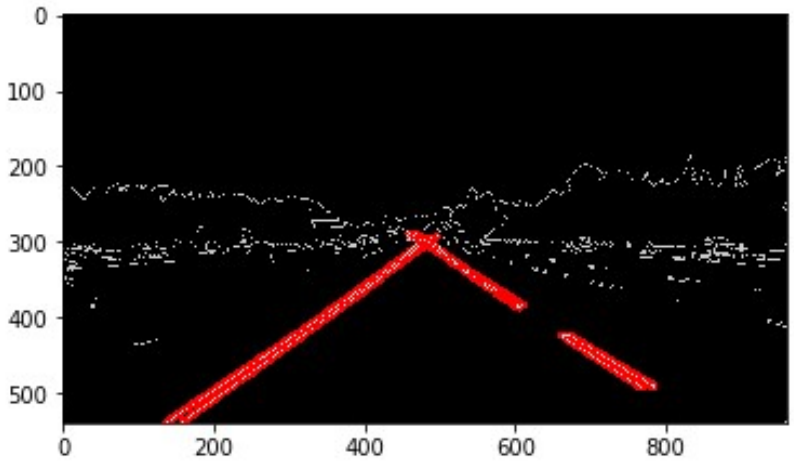
```
rho = 1
theta = np.pi/180
threshold = 1
min_line_length = 1
max_line_gap = 0
line_image = np.copy(image)*0

lines = cv2.HoughLinesP(masked_edges, rho, theta, threshold, np.array([]),
                        min_line_length, max_line_gap)

for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)

color_edges = np.dstack((edges, edges, edges))

lines_edges = cv2.addWeighted(color_edges, 0.8, line_image, 1, 0)
plt.imshow(lines_edges)
```



Thank you & Good luck !