HW1: Sobel Edge Detector

Report

Vision system

- Load “lane_image.jpg”
- Make Ix, ly with Sobel mask (x-direction) and display

- Make magnitude and angle (direction) image
- Make magnitude image (pow, sqrt)
- Make angle image (np.arctan, np.rad2deg)

Term purpose

Apply Sobel filter and techniques to image for detecting precise edge

step

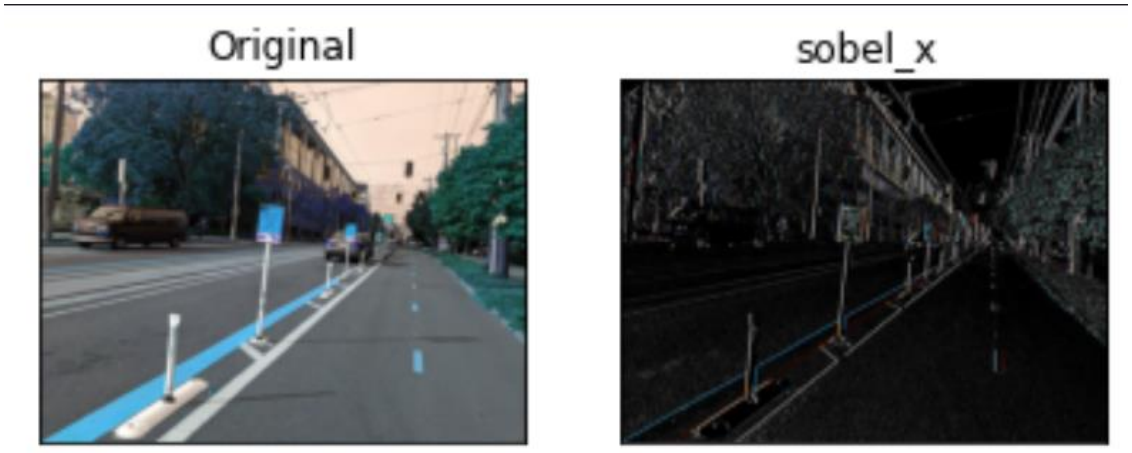
- Load “lane_image.jpg”


```
import cv2
import numpy as np


img = cv2.imread('C:/Users/roadgood/Desktop/visionSystem/lane_image.JPG')



cv2.imshow('lane_imgage',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Make Ix with Sobel mask (x-direction) and display**

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('C:/Users/roadgood/Desktop/visionSystem/lane_image.JPG',cv2.COLOR_BGR2GR

sobel_x = np.array([[-1,0,1],  #sobel horizontal
                    [-2,0,2],
                    [-1,0,1]])
sobel_y = np.array([[-1,-2,-1],  #sobel vertical
                    [0,0,0],
                    [1,2,1]])

dst = cv2.filter2D(img,-1, sobel_y)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([]) # x,y표시 값 지우기
plt.subplot(122),plt.imshow(dst),plt.title('sobel_y')
plt.xticks([]), plt.yticks([])
plt.show()
```

```python
dst = cv2.filter2D(img,-1, sobel_x)
```

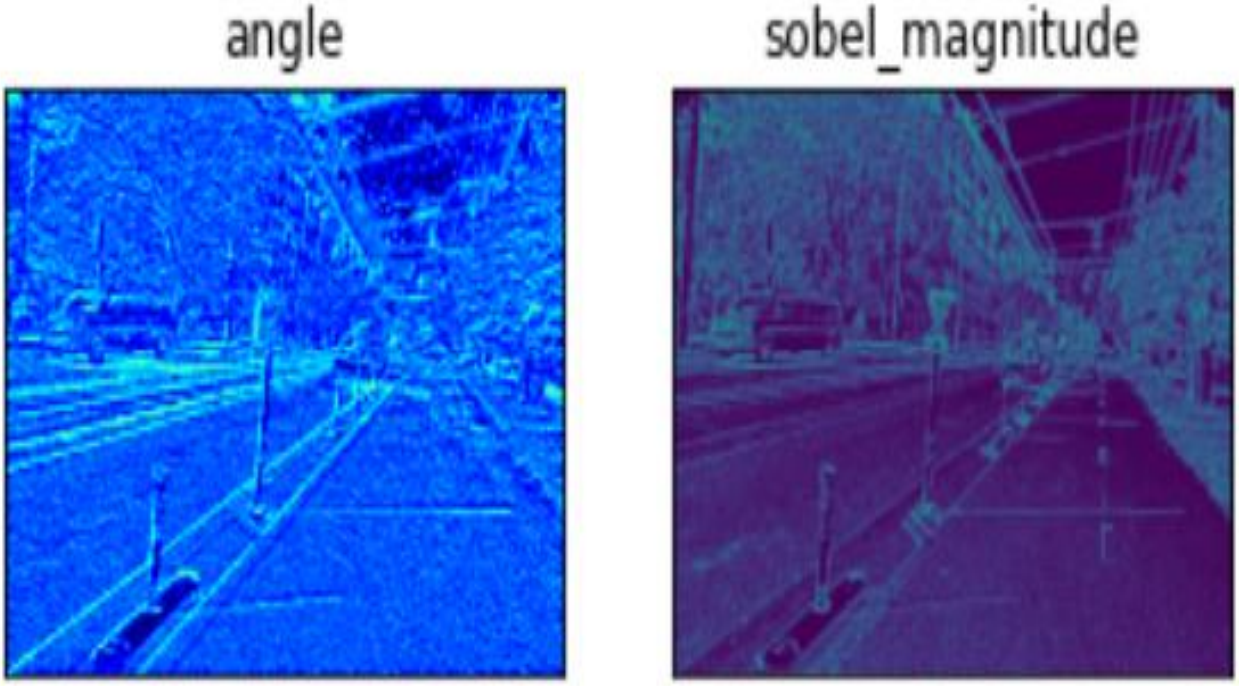


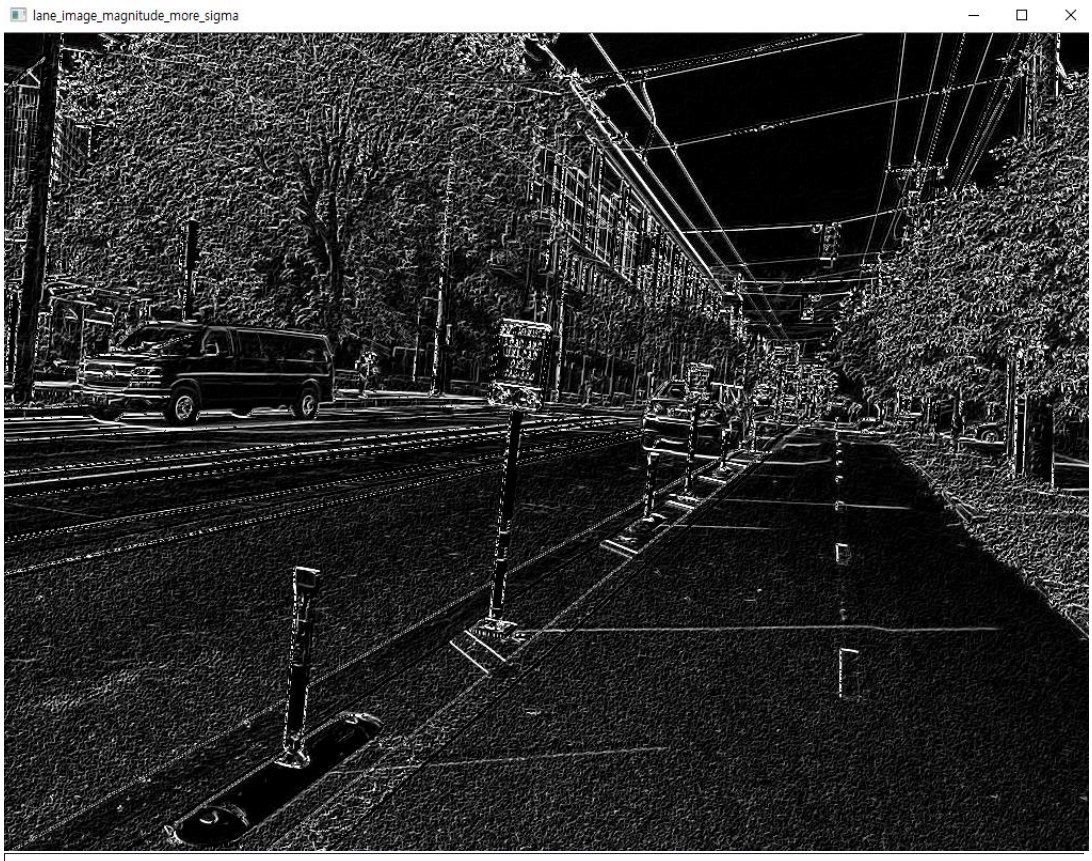
- **Make magnitude and angle (direction) image**

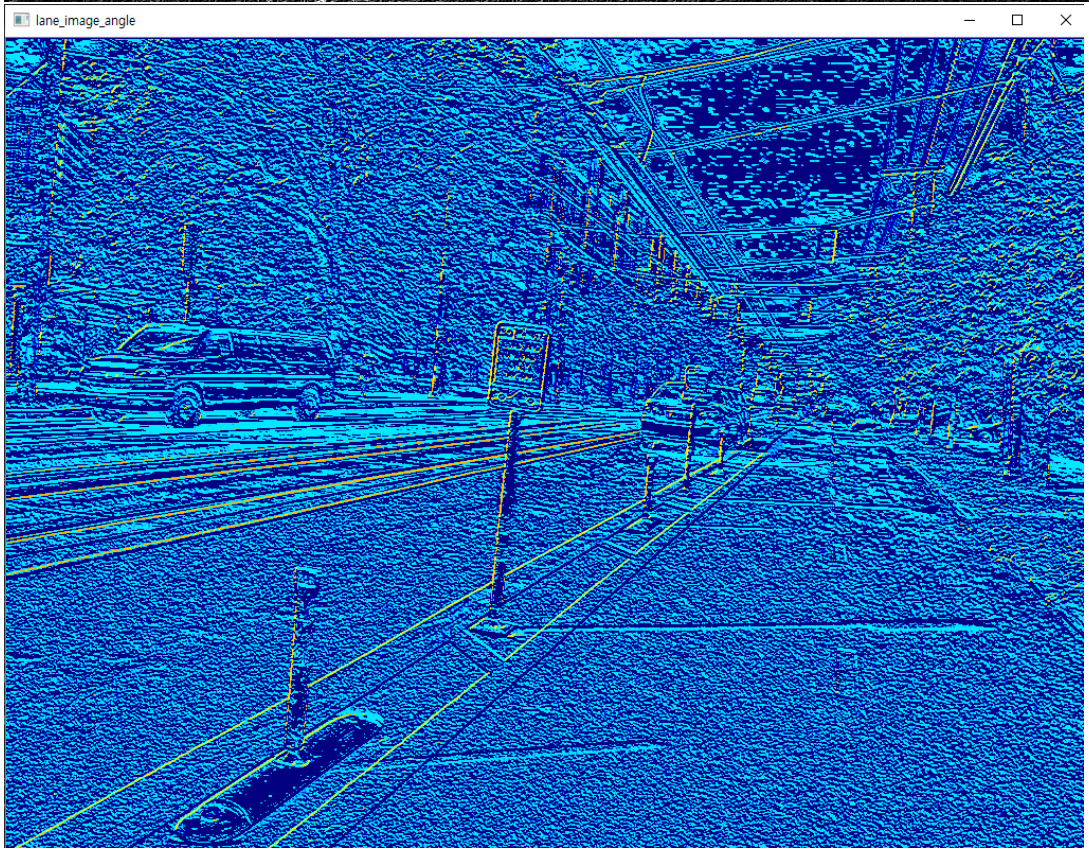
```
dst_x = cv2.filter2D(img,-1, sobel_x) #src,ouput datatype, kernel
dst_y = cv2.filter2D(img,-1, sobel_y)

width,height = img.shape
```

```
dst_jet = np.zeros((width,height), np.uint8) #for jet result

for h in range(0,height):
    for w in range(0,width):
            dst_x[w][h] = math.sqrt(dst_x[w][h]**2+dst_y[w][h]**2)
            dst_jet[w][h] = np.rad2deg(np.arctan2(dst_y[w][h],dst_x[w][h]))
```

```
imc = cv2.applyColorMap(dst_jet,cv2.COLORMAP_JET) #make colorMap with openCV
cv2.imshow('lane_image_magnitude_more_sigma', dst_x)
```

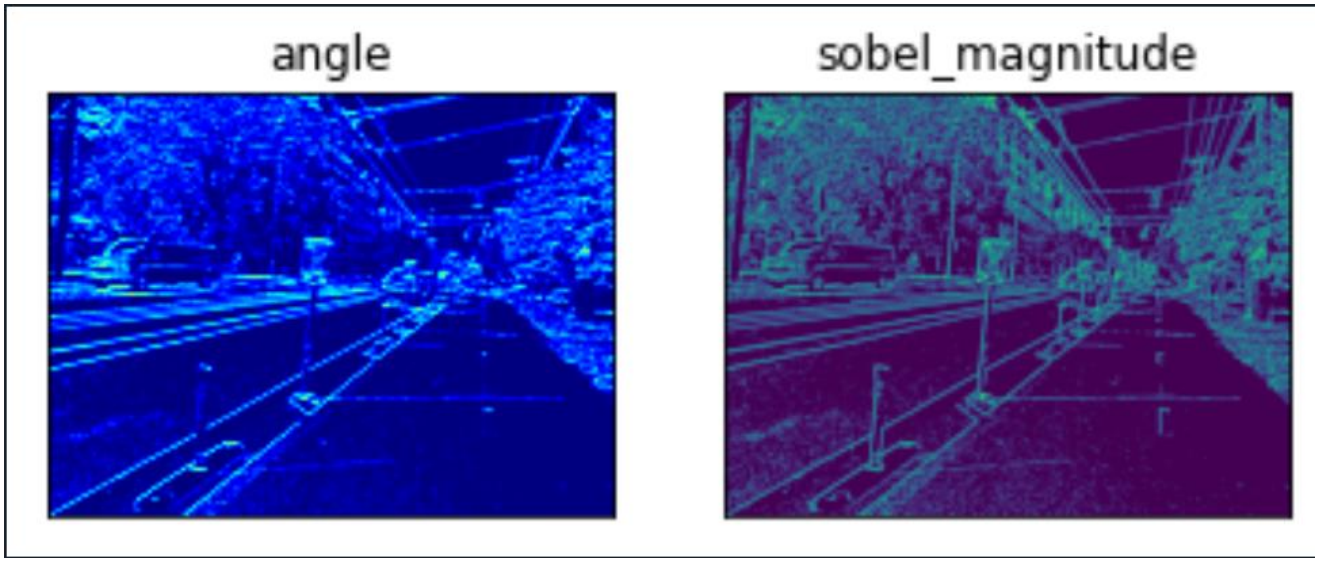


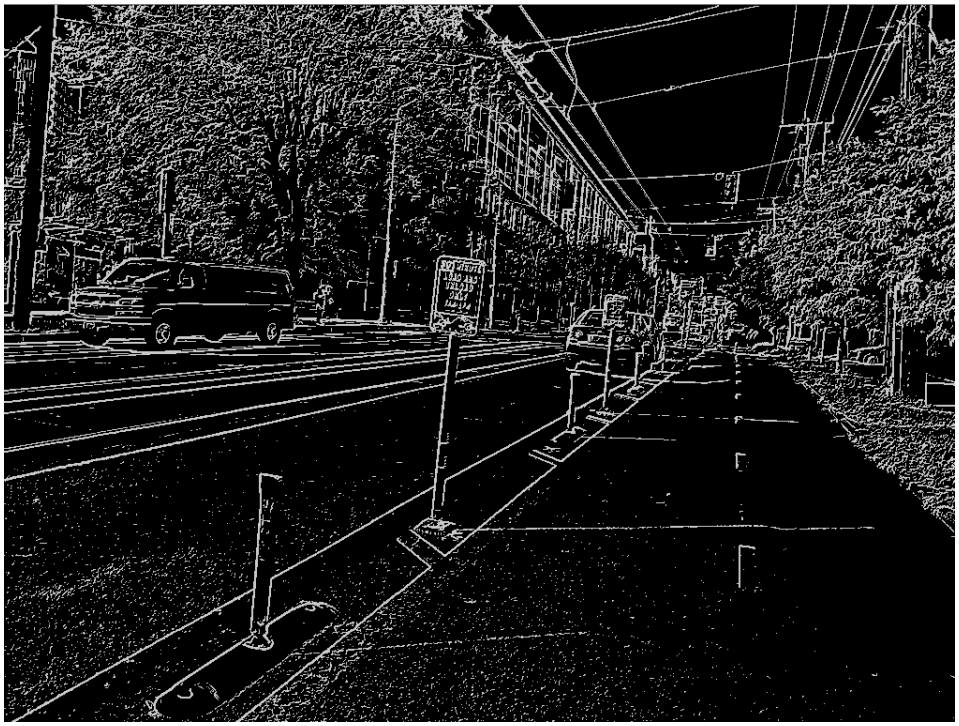

lane_image_magnitude_more_sigma


lane_image_angle

#Too complex at both images => threshold

```
dst_x = cv2.filter2D(img,-1, sobel_x) #src,ouput datatype, kernel
dst_y = cv2.filter2D(img,-1, sobel_y)

width,height = img.shape

dst_jet = np.zeros((width,height), np.uint8) #for jet result

for h in range(0,height):
    for w in range(0,width):
        if math.sqrt(dst_x[w][h]**2+dst_y[w][h]**2)>100: #binarization , threshold
            dst_jet[w][h] = np.rad2deg(np.arctan2(dst_y[w][h],dst_x[w][h]))*4
            dst_x[w][h] = 200 #Binary x 200
        else: dst_x[w][h] = 0

plt.subplot(121),plt.imshow(dst_jet,cmap='jet'),plt.title('angle')
plt.xticks([]), plt.yticks([]) # x,y표시 값 지우기
plt.subplot(122),plt.imshow(dst_x),plt.title('sobel_magnitude')
plt.xticks([]), plt.yticks([])
plt.show()

imc = cv2.applyColorMap(dst_jet,cv2.COLORMAP_JET) #make colorMap with openCV
cv2.imshow('lane_image_angle', imc)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

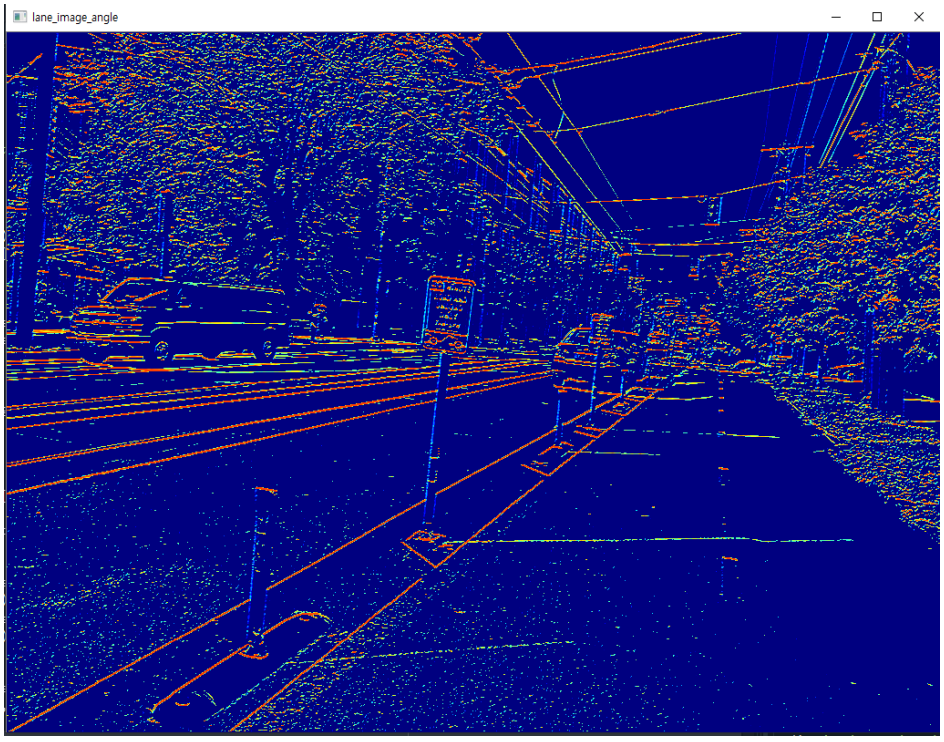




Result: sort out significant pixel from image

#Add dot product(*4) for highlight, color variation

```
dst_jet[w][h] = np.rad2deg(np.arctan2(dst_y[w][h],dst_x[w][h]))*4
```

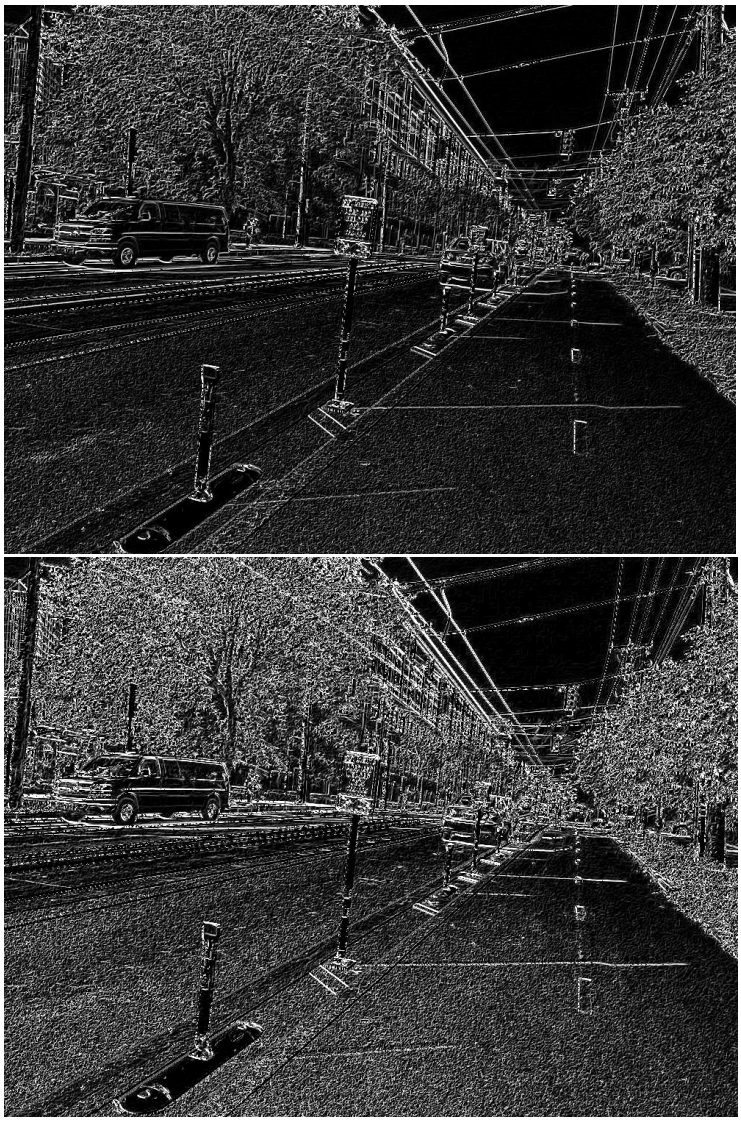


Test

Small sigma

More smoothing, no threshold

```
sobel_x = np.array([[-1,0,1],  #sobel horizontal
                    [-3,0,3],
                    [-1,0,1]])
sobel_y = np.array([[-1,-3,-1],  #sobel vertical
                    [0,0,0],
                    [1,3,1]])
```



=> Large scale edge , more detail

Whole Code

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
import math

img = cv2.imread('C:/Users/roadgood/Desktop/visionSystem/lane_image.JPG',cv2.IMREAD_GRAYSCALE)


sobel_x = np.array([[-1,0,1],  #sobel horizontal
                    [-2,0,2],
                    [-1,0,1]])
sobel_y = np.array([[-1,-2,-1],  #sobel vertical
                    [0,0,0],
                    [1,2,1]])

dst_x = cv2.filter2D(img,-1, sobel_x) #src,ouput datatype, kernel
dst_y = cv2.filter2D(img,-1, sobel_y)

width,height = img.shape

dst_jet = np.zeros((width,height), np.uint8) #for jet result

for h in range(0,height):
    for w in range(0,width):
        if math.sqrt(dst_x[w][h]**2+dst_y[w][h]**2)>100: #binarization , threshold
            dst_jet[w][h] = np.rad2deg(np.arctan2(dst_y[w][h],dst_x[w][h]))*4
            dst_x[w][h] = 200 #Binary x 200
        else: dst_x[w][h] = 0

plt.subplot(121),plt.imshow(dst_jet,cmap='jet'),plt.title('angle')
plt.xticks([]), plt.yticks([]) # x,y표시 값 지우기
plt.subplot(122),plt.imshow(dst_x),plt.title('sobel_magnitude')
plt.xticks([]), plt.yticks([])
plt.show()

imc = cv2.applyColorMap(dst_jet,cv2.COLORMAP_JET) #make colorMap with openCV
cv2.imshow('lane_image_angle', imc)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

실행순서에 따라 구조를 총괄해서 설명하자면

img변수에 cv2.imread로 lane_image를 GrayScale(1channel)로 받는다

Sobel 필터 sobel_x, sobel_y 를 numpy array로 만들어 준 후

cv2.filter2D를 이용해 image에 convolution filter(여기서는 sobel)를 적용한 결과값은 각각 dst_x, dst_y에 넣어주고

shape으로 받아온 width, height로 0값만 들어있는 dst_jet을 생성해주고 for문을 돌려 magnitude 연산을 해준다 np.pow() 대신에 **를 사용하였다

for문 내부에서 dst_y/dst_x를 arctan 시킨값이 radian이기 때문에 rad2deg를 사용했습니

다. (Np.arctan2는 범위가 0~2pi까지인 artan입니다.) 이 값을 4배 해주어 dst_jet에 넣어 후에 plt와 cv2.imshow로 출력하게 됩니다.

dst_x를 magnitude결과용으로 사용하였습니다(지난 값 사용이 없기 때문에)

noise를 줄이기 위해서 if문을 통해 binarization을 한 후 True면 pixel값에 200을 넣었습니다.

plt.subplot(121) -> 1X2 에 1번자리 / cmap: colormap 'jet'

plt.xticks([ ]) 그래프에 값이 표시 되지 않도록 해준다

plt.subplot(122) -> 1X2 에 2번자리

pyplot으로만 보는 것이 부정확해 보여 cv로도 띄운 코드입니다. Cv2.applyColorMap 메서드가 있어 적용한후 imshow를 썼습니다.

바로 닫히지 않게 하기 위해 waitKey(0)을 써 입력대기 상태로 두었습니다.

배운 점, 어려웠던 점

image가 어떤 형태로 computer에 읽히는지 확인할 수 있었고 edge검출을 위해 Image gradient를 구하기 위해 Linear연산을 대체하여 convolution filter인 sobel filter를 사용해보았고 구해진 Image gradient를 magnitude해 image로 보았을 때 edge검출이 되었음을 눈으로 볼 수 있었습니다.

또한 arctan(dy/dx)값 orientation을 구하여 Colormap을 적용시켰습니다.

하지만 전체적으로 색차이가 거의 나지 않아 방법을 찾아야 했습니다.

Noise를 줄이기 위해 threshold를 적용하고 orientaion값을 키워주어 색차이를 확실히 보이도록 개선시켰습니다.

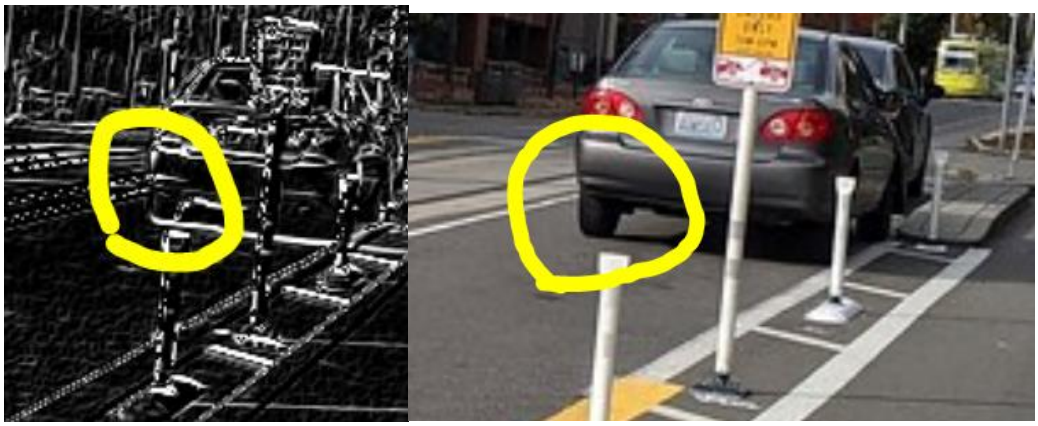
test에서 sigma값을 작게 한 경우를 sobel filter에 변형을 주어 확인했더니 선이 분명해지지만 굵어지는 것을 확인했습니다.

이전에 convolution 연산 class를 만들어 해보았는데 padding처리같은게 무척 까다로웠었습니다. cv.filter2D를 사용하니 최적화도 잘 되어있어 편했고 다음번에 Non-maximum suppression도 사용해보고 link된 edge를 확인해 볼 계획입니다.

```python
#making conv filter (filter, padding, stride(need default) )
#height:행 , width:열 chennel:color종
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math
class picture:
    def __init__(self,fileName):
        self.name = fileName
        fileAddress = 'C:/Users/dnfld/Desktop/deeplearning_learning/image/'
        self.sobel_h = np.array([[-1,0,1],  #sobel horizontal
                                 [-2,0,2],
                                 [-1,0,1]])
        self.sobel_v = np.array([[-1,-2,-1],  #sobel vertical
                                   [0,0,0],
                                   [1,2,1]])

        self.image = cv2.imread(fileAddress+fileName+'.PNG',cv2.IMREAD_GRAYSCALE)
        self.height ,self.width = self.image.shape

    def filt(self,sobel):
        operator = self.sobel_v
        if sobel is 'h':
            operator = self.sobel_h
        for h in range(0,self.height-2,1):
            for w in range(0,self.width-2,1):
                temp = np.array([[self.image[h][w],self.image[h][w+1],self.image[h][w+2]],
                                 [self.image[h+1][w],self.image[h+1][w+1],self.image[h+1][w+2]],
                                 [self.image[h+2][w],self.image[h+2][w+1],self.image[h+2][w+2]]
                                ])

                ans = np.dot(temp,operator).sum()*0.2+40
                self.image[h][w] = ans
    def norm(self,pic):
        for h in range(0,self.height):
            for w in range(0,self.width):
                self.image[h][w] = math.sqrt(pic.image[h][w]**2+self.image[h][w]**2)

pic1 = picture('lena')
pic2 = picture('lena')
pic1.filt('v')
pic2.filt('h')
pic1.norm(pic2)
cv2.imshow(pic1.name, pic1.image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Next Problem

Wrong detections



Resolution

1. Non-maximum suppression
2. Get orientation -> Linking
3. Thresholding