비전 시스템

HW 1

Sobel Edge Detector

과제 목표 :

- Sobel filter를 통해 edge를 검출한다.
- Sobel x축과 y축 Mask를 통해 edge가 어떻게 검출되는지 확인 후 Magitude를 구하여 출력이 어떻게되는지 확인한다.
- 각 픽셀당 얼마나 변하는 폭이 큰지 angle(direction)을 통해 확인하여본다
- Sobel filter만 사용하지말고 Gaussian filter로 smoothing 시킨 후 출력도 확인해여본다.
- Gaussian filter를 원본 image에 적용시킨 후 sobel fiter를 적용시키는 것 보다 Gaussian filter를 먼저 sobel filter로 미분시킨 후 Image를 미분하는 것이 연산량을 줄여줌을 확인한다.
-

과제 내용 :

우선 코드를 시작하기 앞서 사용할 라이브러리를 import하여줍니다.

라이브러리를 import 하고 이미지가 저장되어 있는 위의 주소로 현재 주소를 변경합니다.

```
img = cv2.imread('lane_image.jpg')
img_GRAY = np.float32(cv2.cvtColor(img,cv2.COLOR_BGR2GRAY))
```

img 라는 변수에 lane_image 사진을 불러오고 그 사진을 gray_scale 로 변경하여줍니다.

```
#make filter
Gauss1 = np.float32(cv2.getGaussianKernel(7,3))
Gauss2 = np.outer(Gauss1,Gauss1.transpose())



#sobel filter
mask_x = np.float32(0.25*np.array([[-1,0, 1],[-2, 0, 2],[-1, 0, 1]]))
mask_y = np.float32(0.25*np.array([[1,2,1],[0,0, 0],[-1,-2,-1]]))

#먼저 가우시안 filter를 sobel filter로 미분한다.
derivative_x = cv2.filter2D(Gauss2,-1,mask_x)
derivative_y = cv2.filter2D(Gauss2,-1,mask_y)
```

Gaussian filter 와 Sobel filter (x,y Mask)를 만들어줍니다.

이는 gray_scale 의 이미지를 투영시킬 것입니다.

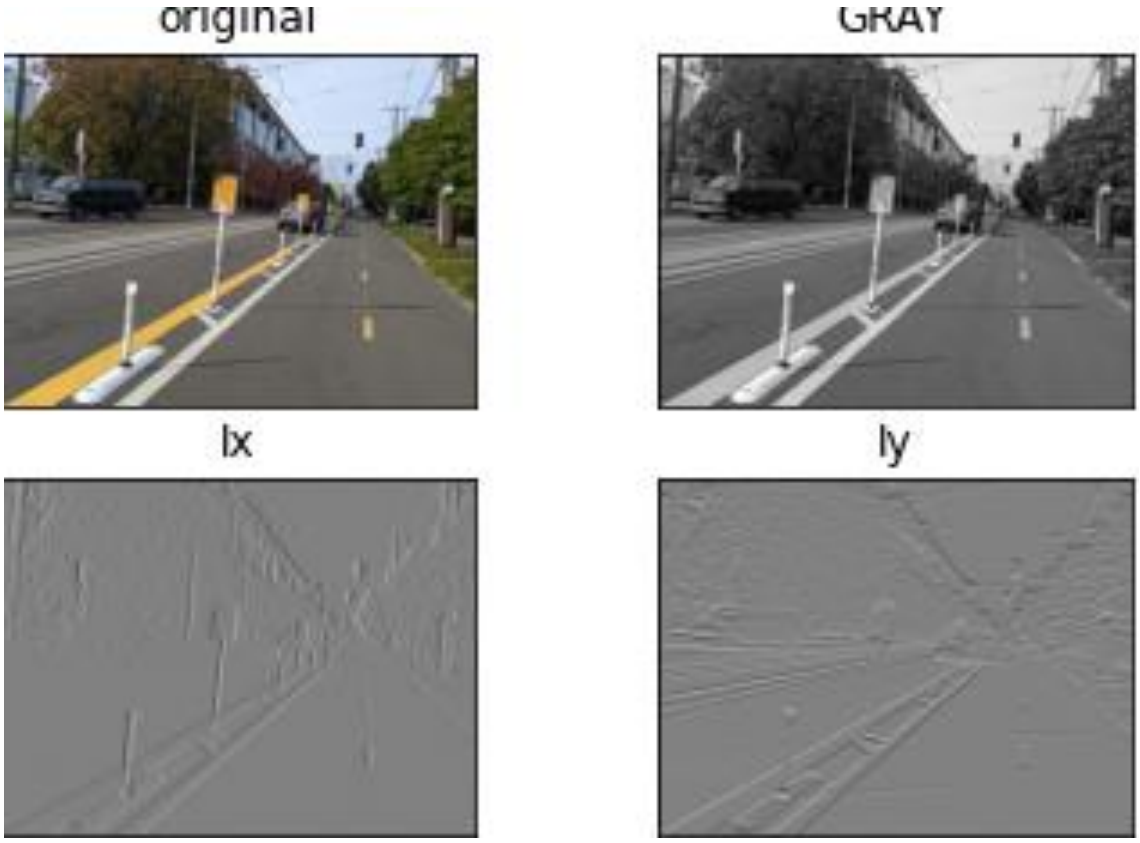
밑에보이는 derivative filter 는 Gaussian filter 를 Sobel filter 로 통과시킨 것입니다.

이미지를 필터 통과시킬 시 Gaussian filter 를 먼저 통과시킴으로써 세세한 edge 는 detect 하지못하나 노이즈를 잡아주고, 투영된 이미지에서 sobel filter 를 통과시키면서 원하는 얻고자하는 edge 를 얻을 수 있을 것입니다.

그리고 gaussina filter 를 통과시키고 sobel filter 로 1 차 미분시키는 방식은 2 번 convolution 을 해야하지만 , gaussian filter 를 sobel filter 로 미분된 필터를 이미지 Conv 를 하면 이미지에 대해서는 Conv 을 한번만 한 것이기 때문에 연산량을 줄일 수 있습니다.

위의 방법이 $\frac{d}{dx}(f * h)$ 식입니다.

```
#미분된 필터로 Image를 Conv 진행
Ix = cv2.filter2D(img_GRAY,-1,derivative_x)
Iy = cv2.filter2D(img_GRAY,-1,derivative_y)
```



위의 그림은 원본이미지를 Gray_scale 로 변경시킨 후 filter 를 통과시켜 Edge 를 얻은 모습입니다. (코드는 아래에 첨부되어있습니다.)

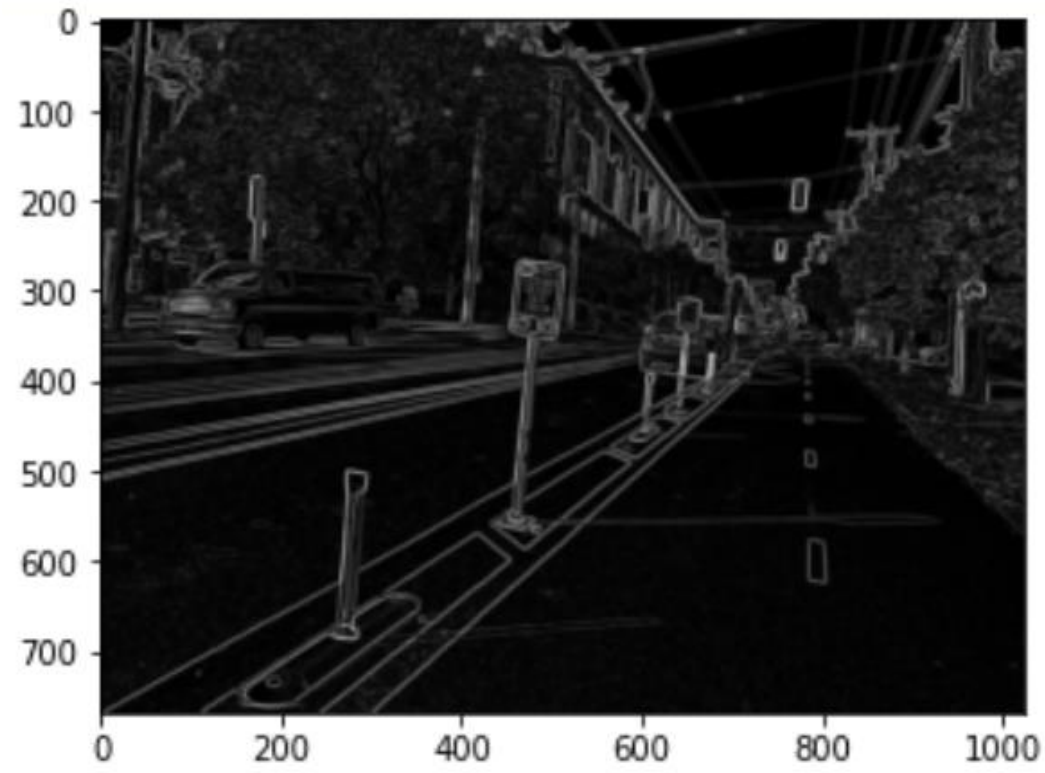
```
plt.subplot(221),plt.imshow(img_RGB),plt.title('original')
plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(img_GRAY,cmap='gray'),plt.title('GRAY')
plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.imshow(Ix,cmap='gray'),plt.title('Ix')
plt.xticks([]), plt.yticks([])
plt.subplot(224),plt.imshow(Iy,cmap='gray'),plt.title('Iy')
plt.xticks([]), plt.yticks([])
plt.savefig('homework1_comparison.png',size = (768,1024))

plt.show()
```

다음으로 Magnitide 를 구한 결과입니다.

```
#magnitude( gradient ) 값을 구함
Mask_magnitude = np.sqrt(pow(Ix,2)+pow(Iy,2))
```

gradient 를 얻을 수 있음

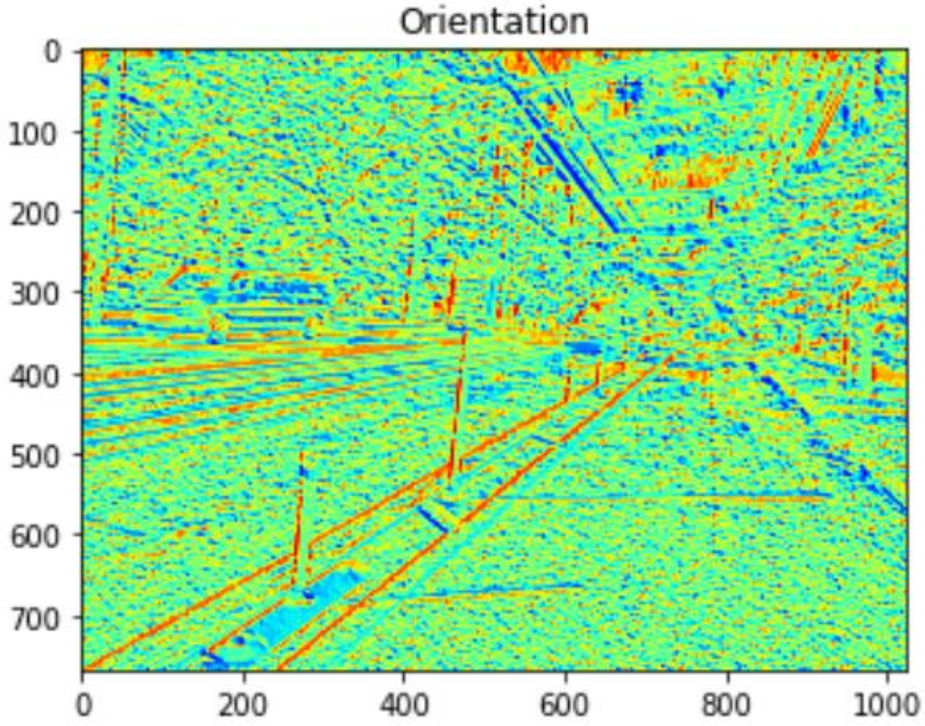


위의 코드로부터 얻은 magnitude 값입니다.

(plt.imshow(Mask_magnitude,cmap='gray') 를 하였습니다

```
#orientaion (direction) 값을 구함
theta = np.rad2deg(np.arctan2(Iy,Ix))    # -180~ 180 사이 값을 가짐.
```

다음으로 각 픽셀의 변화된 정도를 측정하기 위해 얼마나 기울기가 심한지 theta 를 구해줄 수 있습니다.



Plt.imshow(theta,cmap='jet') cmap 을 jet 형식으로 해 줌으로 해서

Theta 의 최소 값이 파란색, 최대값이 빨간색으로 표시가 되었습니다.

추가로

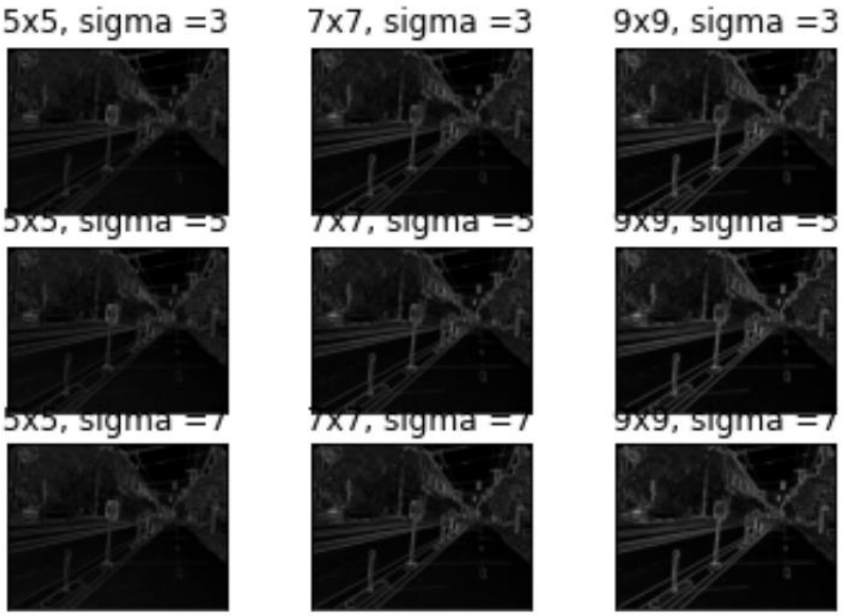
Gaussian filter 를 사용할 때 filter size 에 따라서나 Gaussian 의 sigma 값에 따라서 edge 가 어떻게 나오는지 궁금해서 그림을 출력해보았는데요

아래에 그림으로 첨부 해 놓았습니다. 흐릿해서 잘 보이지는 않으나

소스코드를 돌려보면 Gaussian filter 의 size 를 키우면 라인들이 더 굵게 탐지되지만 세세한 라인들은 사라지게 됩니다.

Simga 를 키우면 또한 더 흐릿하게 되어 이미지가 매끈해지는 것 처럼 보여 노이즈가 사라집니다 그러나 미세한 라인들은 탐지가 어려워지는 단점이있습니다.

여러 시행착오를 겪어봤을 때 이 이미지에서는 size 7x7,sigma 3 필터가 눈으로봤을 때 성능이 좋았습니다.




```python
def gradient(img,size,sigma,count):
    Gauss = np.float32(cv2.getGaussianKernel(size,sigma))
    Gauss2 = np.outer(Gauss,Gauss.transpose())
    #sobel filter
    mask_x = np.float32(0.25*np.array([[-1,0, 1],[-2, 0, 2],[-1, 0, 1]]))
    mask_y = np.float32(0.25*np.array([[1,2,1],[0,0, 0],[-1,-2,-1]]))

    derivative_x = cv2.filter2D(Gauss2,-1,mask_x)
    derivative_y = cv2.filter2D(Gauss2,-1,mask_y)

    Ix = cv2.filter2D(img,-1,derivative_x)
    Iy = cv2.filter2D(img,-1,derivative_y)

    gradient_xy = (np.sqrt(pow(Ix,2)+pow(Iy,2)))

    return gradient_xy

gradient_img = gradient(img_GRAY,5,3)
plt.subplot(331),plt.imshow(gradient_img,cmap='gray'),plt.title('5x5, sigma =3')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,7,3)
plt.subplot(332),plt.imshow(gradient_img,cmap='gray'),plt.title('7x7, sigma =3')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,9,3)
plt.subplot(333),plt.imshow(gradient_img,cmap='gray'),plt.title('9x9, sigma =3')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,5,5)
plt.subplot(334),plt.imshow(gradient_img,cmap='gray'),plt.title('5x5, sigma =5')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,7,5)
plt.subplot(335),plt.imshow(gradient_img,cmap='gray'),plt.title('7x7, sigma =5')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,9,5)
plt.subplot(336),plt.imshow(gradient_img,cmap='gray'),plt.title('9x9, sigma =5')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,5,7)
plt.subplot(337),plt.imshow(gradient_img,cmap='gray'),plt.title('5x5, sigma =7')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,7,7)
plt.subplot(338),plt.imshow(gradient_img,cmap='gray'),plt.title('7x7, sigma =7')
plt.xticks([]), plt.yticks([])
gradient_img = gradient(img_GRAY,9,7)
plt.subplot(339),plt.imshow(gradient_img,cmap='gray'),plt.title('9x9, sigma =7')
plt.xticks([]), plt.yticks([])
```

과제에서 배운 것/ 어려웠던 점

배운점 : sobel필터 x,y Mask앞에 1/8 혹은 1/4를 안해주니 라인을 노이즈가 많이 타는 것을 확인함.(마스크 앞에 상수를 나눠주니 노이즈가 덜 탐.)

해결 :: 이는 Ix,Iy를 구할 때 이미지 형을 float32, float64가 아닌 uint8형식으로 되어있어서 이런 문제점이 발생했었음.

Sobel필터로만 gradient 구했는데 노이즈가 너무 많이 낌. 그래서 Gaussian filter를 거치고 line을 뽑아내니 훨씬 안정화 되었음.

Conv 2번 하지말고 필터를 conv 먼저 시키고 image를 통과시키니 출력속도가 빨라졌습니다.

어려운 점 : magnitude는 아주 예상되로출력이 되나 angle을 뽑아 보려고하니 이게 굉장히 노이즈가 많은 것 처럼 나옴. + ppt처럼 magnitude한 라인에서 그 라인에 대해 색갈이 입혀지는 것 처럼 보였는데 직접해보니 컬러가 대부분 초록색으로 나옴을 보았습니다.

Orientation 시 이미지가 유효한 edge에 대해서 theta를 구해지지 않았고 해석하기 어려울정도로 무아지경으로 그림이 도출되었습니다..

이 문제를 어떻게 해결할 수 있을까요?

이미지를 필터 씌우기 전에

Np.uint8 형식에서 np.float32형식으로 바꾸니 라인 디택트가 훨씬 잘 되고 원하는 형식으로 뽑아낼 수 있게 됬습니다.

필터 Type을 Float64형식으로하니 더 자세한 값을 받을 수 있지만 메모리를 더 잡아먹는 단점이 있습니다.