2016-17912 최형욱 HW 2

1.

8 (0-1)

8 0

10

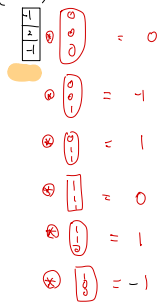
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

* ? ⇒

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

8 (-1)

operator ⇒ (3×1)



이 operator는 Second order partial derivate의 discrete 형태이다.

즉 second order finite difference의 근사값이다.

이 operator를 convolution 해서 수평축에 존재하는 edge를 탐지할수 있다.

2. Edge detection

a) 이미지의 Edge 가 $h(x,y)=0$ 이면서 부호가 바뀌는 점으로 표현된다. 즉 $h(x,y)=0$ 은 $I(x,y)$ 에서 Edge인 점들의 집합이다.

b) σ 값을 증가시키면 더욱 명확하게 보는 edge만을 탐지할 수 있다. 즉, $h(x,y)=0$ 인 (x,y)는 작게 나타날 것임.

3. Hough Transform Line Parameterization

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

임의의 점 (x_i, y_i)에서

$\rho = x_i\cos\theta + y_i\sin\theta$ 를 만족.

$$\rho = \sqrt{x_i^2+y_i^2}\left(\frac{x_i}{\sqrt{x_i^2+y_i^2}}\cos\theta + \frac{y_i}{\sqrt{x_i^2+y_i^2}}\sin\theta\right)$$

$$= \sqrt{x_i^2+y_i^2}\left(\sin\alpha\cos\theta + \cos\alpha\sin\theta\right) \quad \left(\because \sin\alpha = \frac{x_i}{\sqrt{x_i^2+y_i^2}},\ \cos\alpha = \frac{y_i}{\sqrt{x_i^2+y_i^2}}\right)$$

x_i, y_i를 각도 α로 표현. (r=1인 단위원 베르겔)

$$= \sqrt{x_i^2+y_i^2}\left(\sin(\alpha+\theta)\right)$$

$$\therefore \text{amplitude} = \sqrt{x^2+y^2}$$

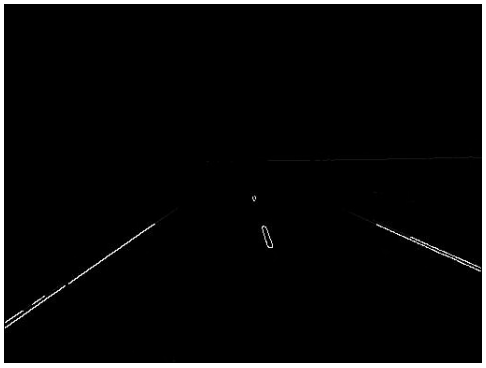
$$\text{Phase} = \alpha = \sin^{-1}\left(\frac{x_i}{\sqrt{x_i^2+y_i^2}}\right)$$

2) 아니오. a)의 식에 의하면 임의의 (x_i, y_i)와 관련 있는 값은 $\hat{I}(x,y)$의 amplitude와 phase 이다. 이 값들은 진폭, 평행이동(위상) 과 관련이 있지만, 주기 및 진동수와는 관련이 없다.

4-2. Parameter의 변화에 따른 결과의 변화 서술하기

- highthreshold를 높이는 경우 double threshold 이후 나타나는 edge가 감소하였다. 즉, 더욱 강한 intensity를 갖는 edge만 남는 것으로 확인하였다. (그림 1, 그림 1-1 참조)
- low threshold를 코드에서는 high threshold에 대한 비율로 정의하였다. 이 값을 높일 수록 weak edge에 대한 기준이 엄격해지는 것으로, 남게되는 edge가 줄었다.
- sigma의 경우 guassian smoothing에 사용되는 guassian kernel을 바꾸고, 그 결과 noise를 제거할 수 있다. 그림 3과 4에서 알 수 있듯이 그림 3은 noise가 제거되어있음을 확인할수있을
- gaussian kernel의 size도 조절해보았는데 size가 클수록 blur 효과 덕에 noise가 잘 제거되었다. (그림1과 그림5 비교)

<그림 1>



<그림 1-1, high : 최댓값x0.175>
이외 그림1과 같음

<그림 2>

① kernel size : (5,5)
② sigma : 2
③ high Threshold : image Intensity의 최댓값 x 0.15
④ low Threshold : high Threshold x 0.8

① kernel size : (5,5)
② sigma : 2
③ high Threshold : image Intensity의 최댓값 x 0.15
④ low Threshold : high Threshold x 0.2

② sigma = 5 ②: sigma = 1

<그림 3, 4>

① :(5,5) ③ image Intensity의 max x 0.15

④ : ③ x 0.8

<그림 5>

① kernel size : (3,3)
이외는 그림 1과 동일

* Non - maximal - suppressen

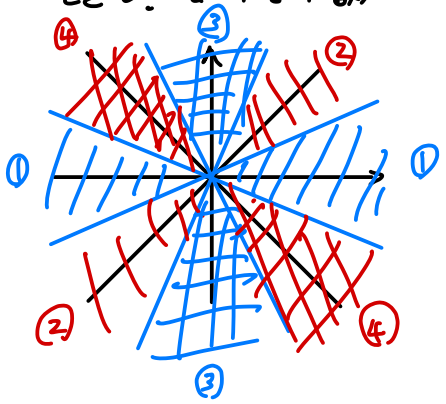
```python
def non_maximal_suppression(magnitude, angle):
    R, C = magnitude.shape

    suppressed_img = np.zeros((R, C))

    for r in range(1, R - 1):
        for c in range(1, C - 1):
            # 각도를 0~180도로 만들어줌. 직선의 성질 이용.
            pix_angle = angle[r, c] if angle[r, c] > 0 else angle[r, c] + 1
            # 비교에 사용될 변수 선언
            i = 1.0
            j = 1.0
            # x 축 기준      You, 2 minutes ago • Uncommitted changes
            if (0 <= pix_angle < 1 / 8) or (7 / 8 <= pix_angle <= 1):
                i = magnitude[r, c - 1]
                j = magnitude[r, c + 1]

            elif 1 / 8 <= pix_angle < 3 / 8:
                i = magnitude[r + 1, c - 1]
                j = magnitude[r - 1, c + 1]
            elif 3 / 8 <= pix_angle < 5 / 8:
                i = magnitude[r - 1, c]
                j = magnitude[r + 1, c]
            elif 5 / 8 <= pix_angle < 7 / 8:
                i = magnitude[r - 1, c - 1]
                j = magnitude[r + 1, c + 1]

            if magnitude[r, c] > i and magnitude[r, c] > j:
                suppressed_img[r, c] = magnitude[r, c]
            else:
                suppressed_img[r, c] = 0

    return suppressed_img
```

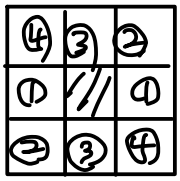
suppress

parameter magnitude는 gradient magnitude $\sqrt{S_x^2 + S_y^2}$, 즉 np.hypot(S_x, S_y)의 결과 값이고,

angle은 np.arctan2(S_y, S_x) / np.pi을 받는다.

즉, $\arctan\left(\frac{S_y}{S_x}\right)$의 값은 $[-\pi, \pi]$ 구간으로 나타내는 값이다. 이 값을 π로 나누어서 코드 작성에 용이하도록 한 값을 angle matrix로 받는다. (범위 [-1, 1]의 값)





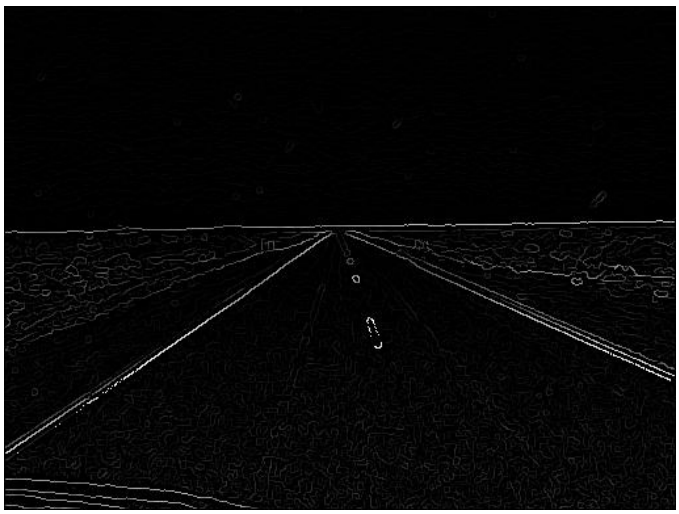
이 함수에서 angle의 범위는 [-1, 1] 이므로, (전처리에 의한) 방향에 따른 크기를 비교해야 하는데 $\frac{1}{2}$과 $-\frac{1}{2}$은 ③의 구간에 해당하고 같은 방향이다. (부호만 다른) 따라서 직선의 성질을 일부 활용하기 위해 angle [r, c] 값이 음수인 경우 1을 더해주었다.

그 후 ①②③④ 구간마다 같은 방향에서 주변의 2개의 픽셀의 magnitude를 i와 j에 저장 후, 해당 픽셀의 magnitude와 비교해서 가장 큰 값이면 남겨두고 아니라면 픽셀의 magnitude를 0으로 만들어.

(example)

magnifinde

NMS →



suppressed, before edge tracking

4.3 Hough Transform

Im image & H

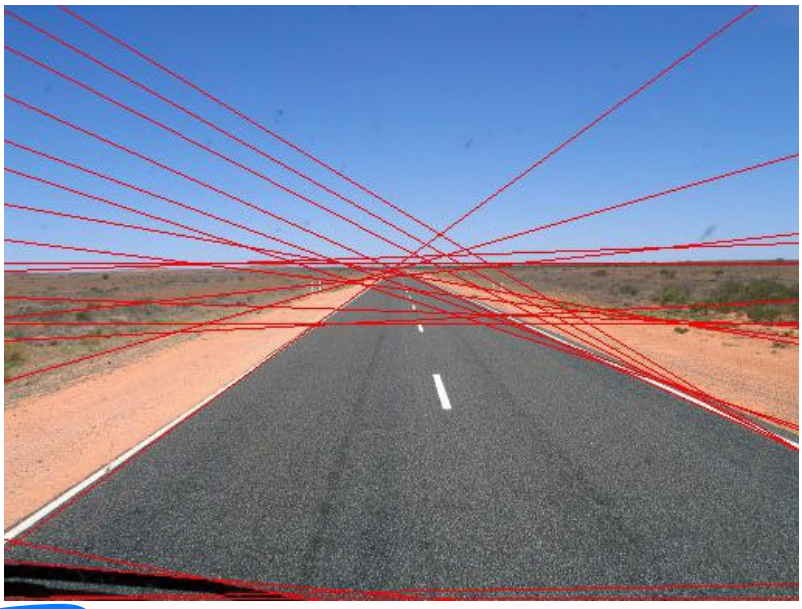
img02_Im.

img02_H

-사용한 parameter

- sigma: 9 , kernel size: (5,5)
- hightreshold : Intensity의 mask * 0.33
- low threshold : low Threshold
- rhoRes : 1
- thetaRes: π / 180
- nLines: 20

4.4 https://gist.github.com/ri-sh/45cb32dd5c1485e273ab81468e531f09 → NMS ref



$\sqrt{Im\_width^2 + Im\_height^2}$

인덱스를 구하기위해 ✓ 전체 이미지의 대각선

img_size : 원래이미지의 (height, width)를 추출, H에 저장할 때 길이만큼 대해주었던 것을 (rho index) * r_ρ 에서 빼주어야 원래 rho가 나오는데 이미지의 크기를 houghLines함수에서는 알 수 없기 때문이다.

전 HoughLines (H, rρ, rθ, n) -> HoughLines(H, rhoRes, thetaRes, nLines, img_size) 후

추가

따라서 여기서는 r_θ를 이용하여 θ의 인덱스에서 θ의 원래 값을 (θ_index * thetaRes / (π/180)) - 90° 와 같은 방법으로 구해냈다.

현재 라디안 → 원래라디안, rad → deg, ↳ offset

r_ρ을 이용하여 (rho index) * r_ρ - $\sqrt{img\_size[0]^2 + img\_size[1]^2}$ 의 식을 통해 원래의 rho값을 구했다.

∴ r_ρ, r_θ 는 H의 index에서 원래 ρ, θ를 찾기 위해 필요함.