

OpenCV-Python

03. Line Detection



2021-W
MECHA Seminar

Contents :

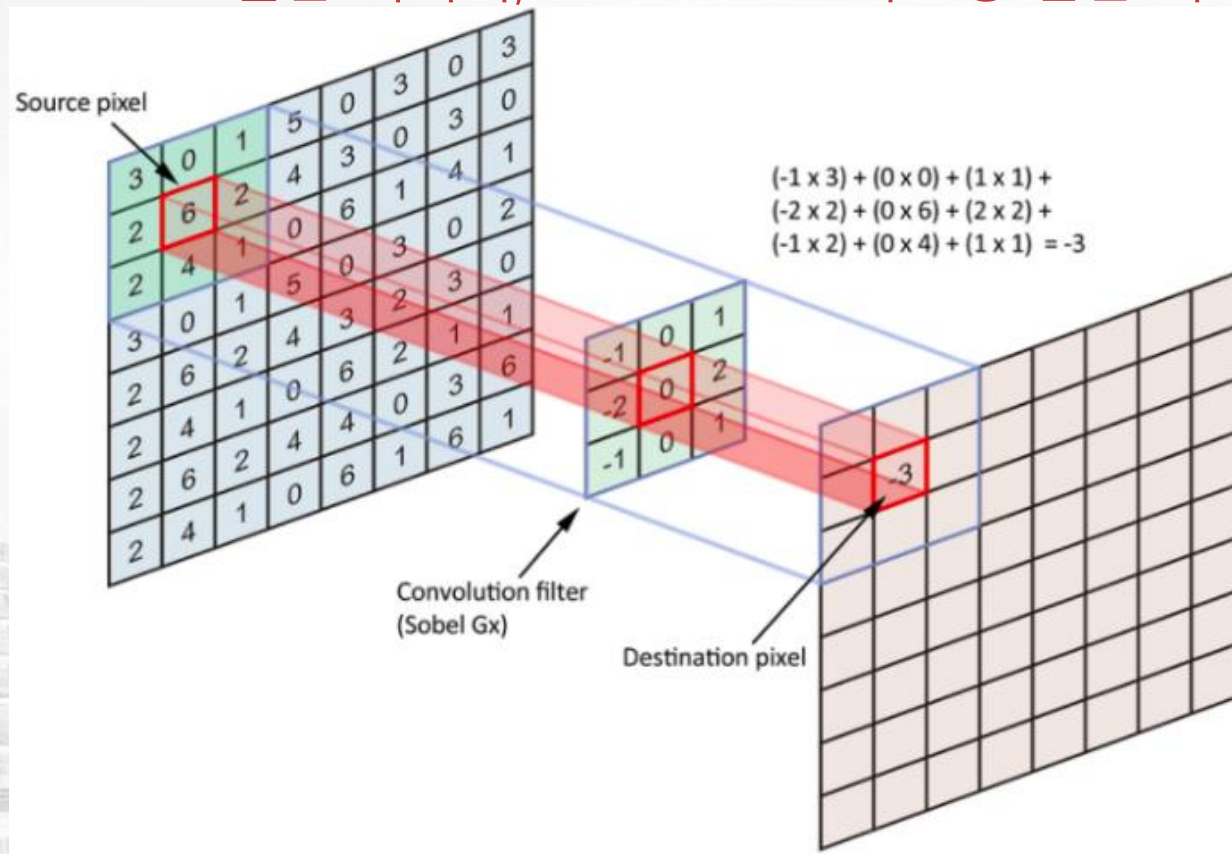
- Gaussian Blur
- Canny Edge Detection
- Hough Transform
- 과제: example2



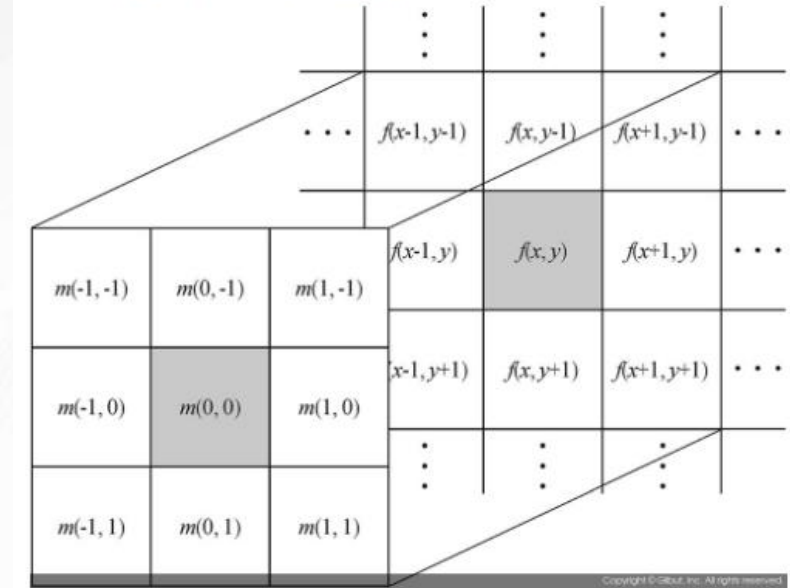
Gaussian Blur :

필터 & 컨볼루션(Convolution) 연산

- 계산 방법 ex 2행 2열
- Source: 원본 이미지, Destination: 마스크 연산 이미지



▼ 그림 8-1 마스크를 이용한 필터링 방법



마스크 연산에 의해 새로 결정되는 영상의 픽셀 값은 다음과 같다.

$$g(x, y) = \sum_{i=-1}^1 \sum_{j=-1}^1 m(i, j) f(x+i, y+j)$$

$$= m(-1, -1) f(x-1, y-1) + m(0, -1) f(x, y-1) + m(1, -1) f(x+1, y-1)$$

$$+ m(-1, 0) f(x-1, y) + m(0, 0) f(x, y) + m(1, 0) f(x+1, y)$$

$$+ m(-1, 1) f(x-1, y+1) + m(0, 1) f(x, y+1) + m(1, 1) f(x+1, y+1)$$

GaussianBlur :

Average Blurring

- Average Blurring: 필터의 value가 모두 같음
- 단점: 필터의 크기가 클 때 선명도가 떨어짐

| | | |
|-----|-----|-----|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

| | | | | | | |
|----|----|-----|-----|-----|----|----|
| 90 | 90 | 90 | 90 | 90 | 90 | 90 |
| 90 | 90 | 90 | 90 | 90 | 90 | 90 |
| 90 | 90 | 255 | 255 | 255 | 90 | 90 |
| 90 | 90 | 255 | 255 | 255 | 90 | 90 |
| 90 | 90 | 255 | 255 | 255 | 90 | 90 |
| 90 | 90 | 90 | 90 | 90 | 90 | 90 |
| 90 | 90 | 90 | 90 | 90 | 90 | 90 |

(a) 원본 영상

| | | | | | | |
|----|-----|-----|-----|-----|-----|----|
| 90 | 90 | 90 | 90 | 90 | 90 | 90 |
| 90 | 105 | 120 | 135 | 120 | 105 | 90 |
| 90 | 120 | 150 | 180 | 150 | 120 | 90 |
| 90 | 135 | 180 | 255 | 180 | 135 | 90 |
| 90 | 120 | 150 | 180 | 150 | 120 | 90 |
| 90 | 105 | 120 | 135 | 120 | 105 | 90 |
| 90 | 90 | 90 | 90 | 90 | 90 | 90 |

(b) 블러링 영상



(a) 입력 영상



(b) 3×3 평균 값 필터



(c) 5×5 평균 값 필터

GaussianBlur :

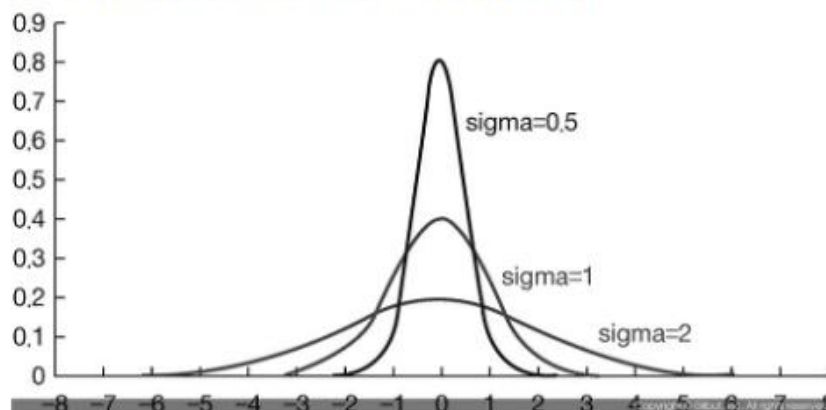
Gaussian Filtered Blurring

- Gaussian Filter: 자연현상을 가장 잘 표현하는 이산확률모형

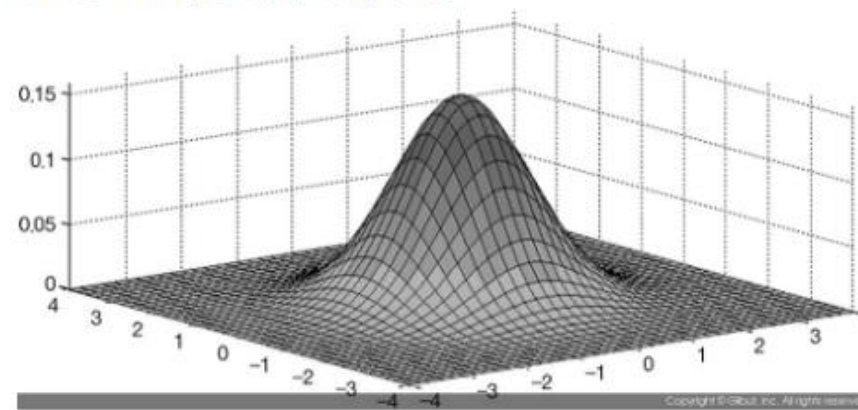
- 2D Gaussian Function:
$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Sigma 값을 조정하여 필터의 weight를 조절함. 영상을 부드럽게 하기 위해 가장 많이 사용됨

▼그림 8-9 다양한 값에 따른 1차원 가우시안 함수의 모습



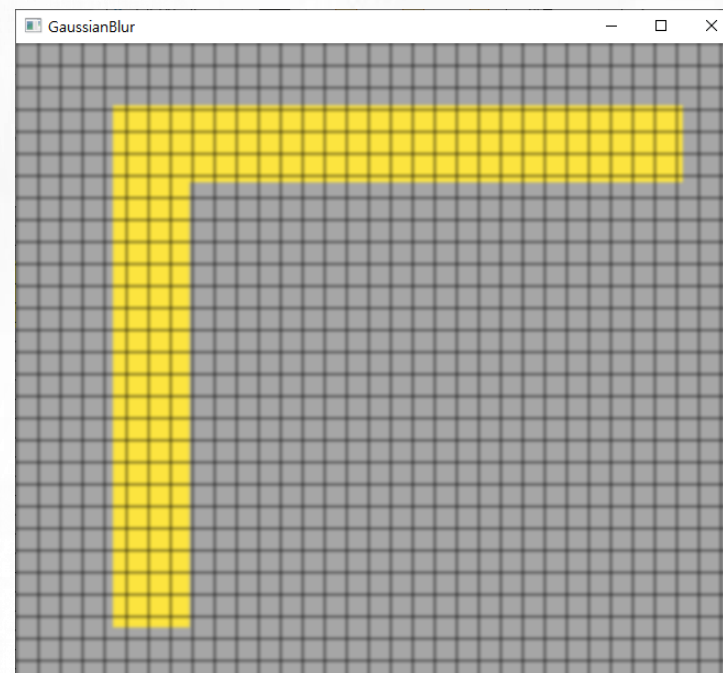
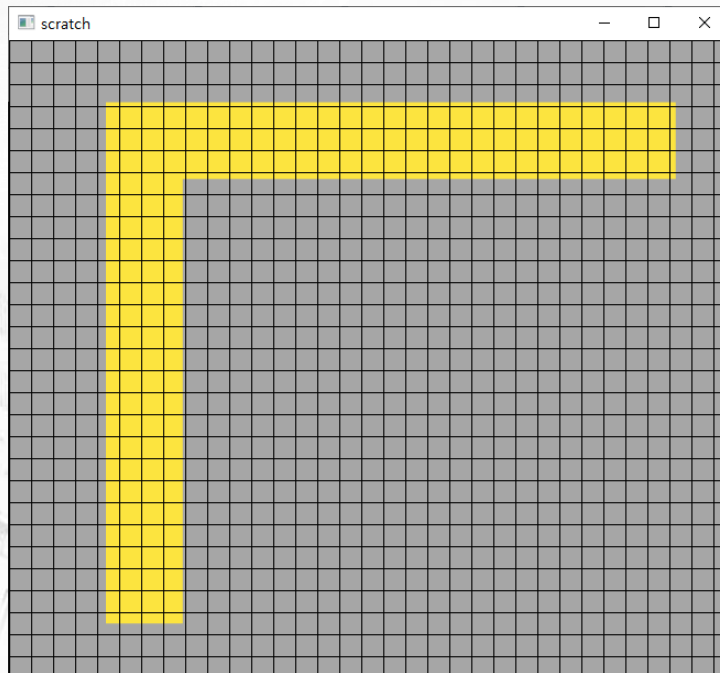
▼그림 8-10 2차원 가우시안 함수의 모습



GaussianBlur :

linedetect.py

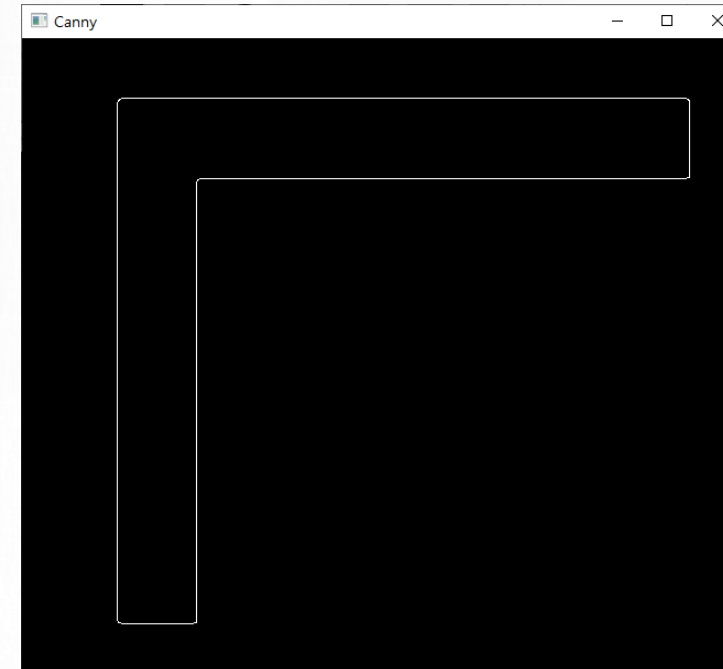
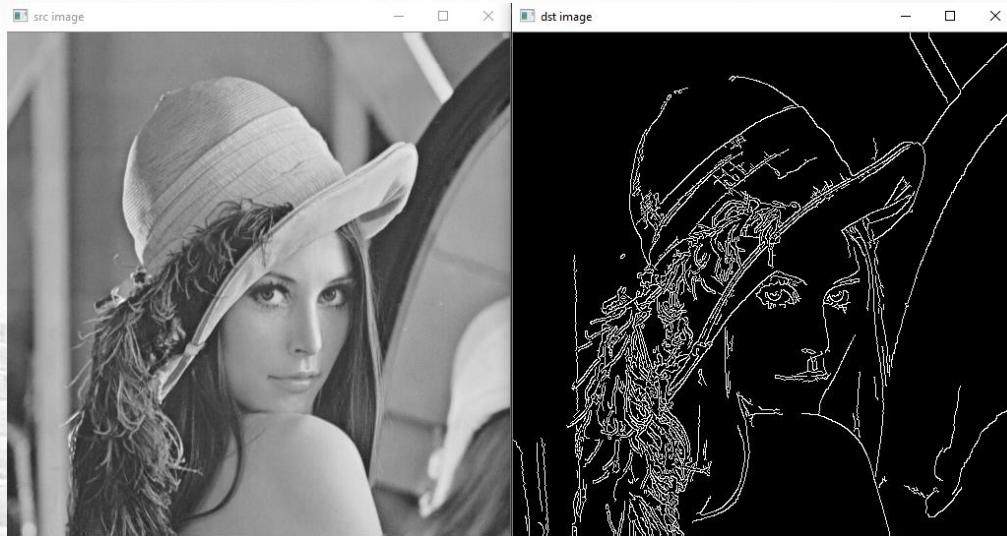
- `cv2.GaussianBlur(image, kernelsize, sigma)`
- 원본이미지에 임의의 noise를 주고 kernel size=7인 gaussianblur 진행
- Non-Blur vs Blur



Canny Edge Detection :

linedetect.py

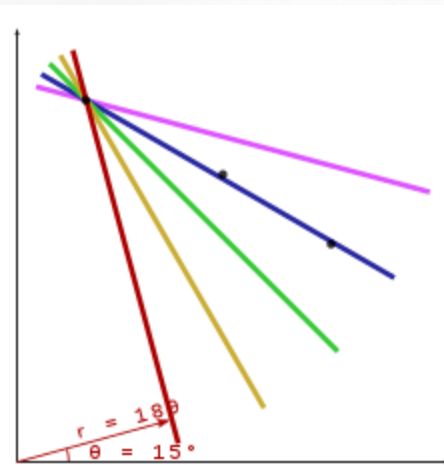
- `cv2.Canny(maskedimage, min threshold, max threshold, apertureSize)`
- pixel 색상 변화 gradient를 이용하여 경계선 검출



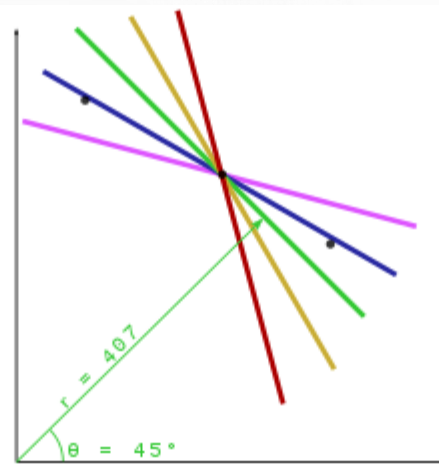
Hough Transform :

극좌표계로 변환

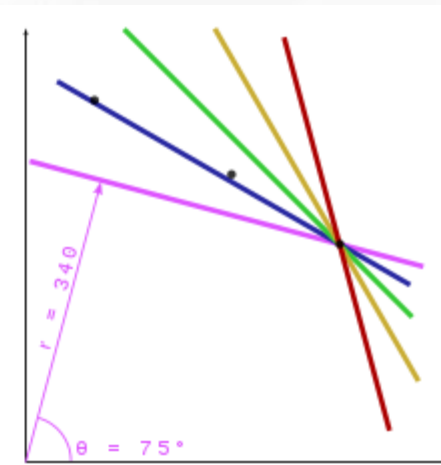
- 하나의 점을 지나는 직선의 방정식을 $y=mx+n$ 이 아닌 $r = x \cos \theta + y \sin \theta$, 로 생각
- $\theta=1 \sim 180$ 로 변화하면서 근접한 기울기를 찾는 방식



| θ | r |
|----------|-------|
| 15 | 189.0 |
| 30 | 282.0 |
| 45 | 355.7 |
| 60 | 407.3 |
| 75 | 429.4 |



| θ | r |
|----------|-------|
| 15 | 318.5 |
| 30 | 376.8 |
| 45 | 407.3 |
| 60 | 409.8 |
| 75 | 385.3 |

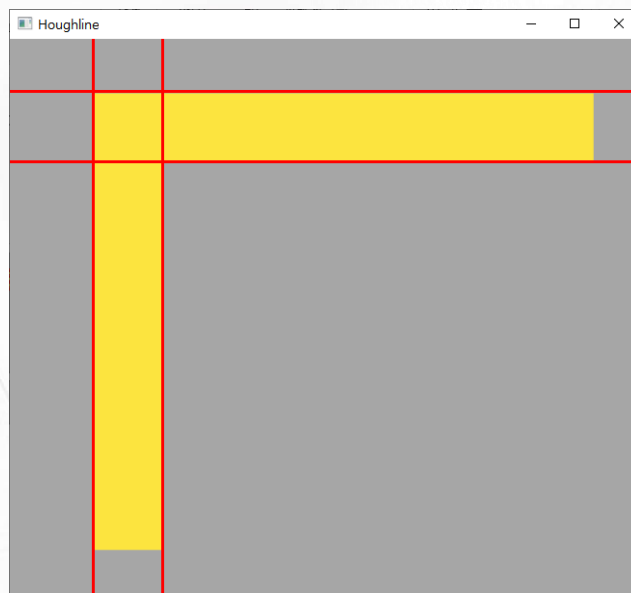


| θ | r |
|----------|-------|
| 15 | 419.0 |
| 30 | 443.6 |
| 45 | 438.4 |
| 60 | 402.9 |
| 75 | 340.1 |

Hough Transform :

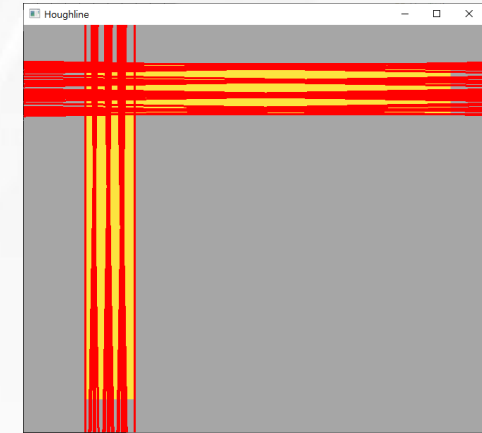
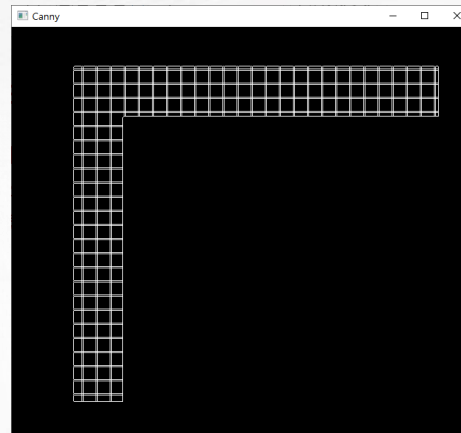
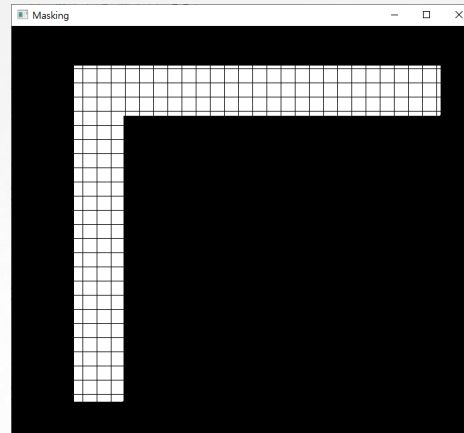
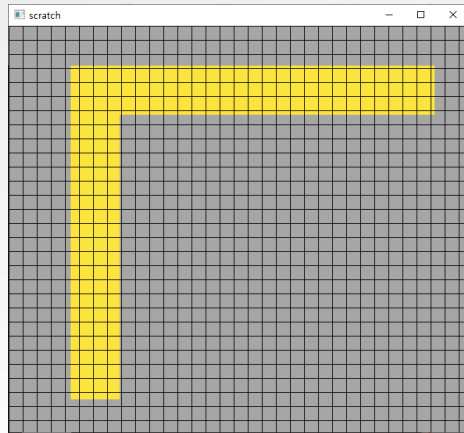
linedetect.py

- `cv2.HoughLines(canny image, rho, theta, threshold)`
 - rho = 거리 측정 해상도(pixel 간격), theta = $\pi/180$
- HoughLine을 찾은 뒤에 line을 그려주는 작업 진행
- `cv2.line(image, startpoint, endpoint, color, thickness)` 로 라인 그리기

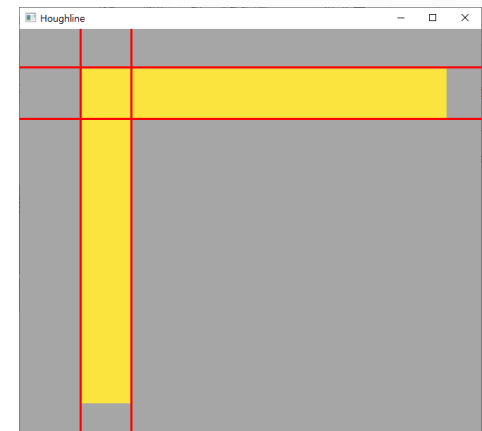
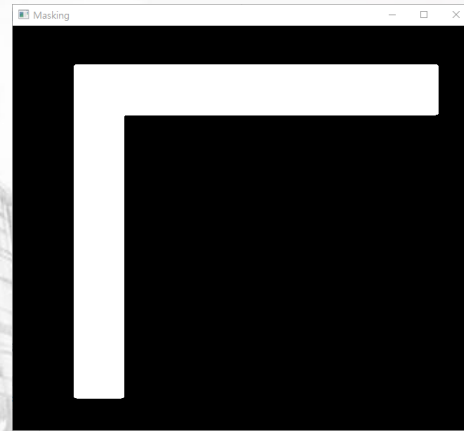
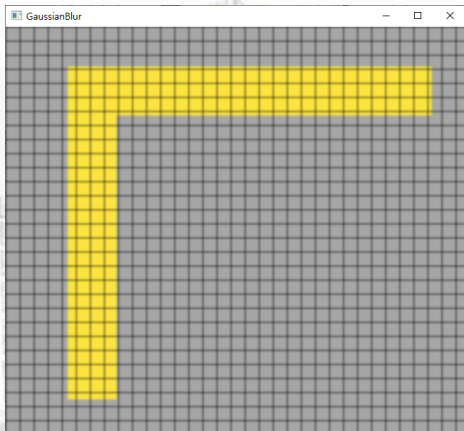


Results of Non-Blur vs Blur : Masking, Canny, HoughLines

Non-Blur



Blur



과제3 :

tline.png에서 라인 가장자리를 검출 후 직선을 다음과 같이 그리시오.

- 제목을 example3.py로 새 python file 생성
- Hint:
- Cv2.HoughLines의 threshold 숫자 조절
- 1(좌) 2(우). 직선의 개수는 표시된 그림보다 많을 수도 있음

