Chapter 05

인공지능 영상 인식 맛보기

여기서는 바리스타 기능을 하는 파이썬 프로그램을 완성해 봅니다. 그 과정에서 프로그램 작성에 필요한 파이썬 언어의 구성 요소를 살펴보고 이해하고 활용할 수 있도록 합니다. 첫 번째로 sys.argv로 파이썬 프로그램에 인자를 넘기는 방법, import를 통한 모듈을 불러오는 방법을 이해해보고, 두 번째로 클래스를 문을 사용해보고, 그 역할을 정리하며 이해해 봅니다. 세 번째로 함수 인자로 단일 값 변수, 목록 값 변수, 객체 값 변수가 전달되는 과정에 대해 살펴보고 이해해 봅니다.

03 이미지 필터링 이해하기

필터링이란 이미지나 영상에서 원하는 정보만 걸러내는 작업을 말합니다. 이 과정에서 잡음을 걸러내어 영상을 깨끗하게 만들거나 잔선을 제거하여 흐리게 만들거나 부드러운 느낌을 걸러내어 선명한 느낌의 영상을 만들 수 있습니다.

필터링은 필터, 커널, 마스크, 윈도우 등으로 불리는 작은 크기의 행렬을 이용합니다. 필터링 연산의 결과는 행렬의 크기와 각 항목의 값에 의해 결정됩니다.

이미지 필터링은 수식을 이용하여 이미지를 구성하는 픽셀 값을 바꾸어 원하는 형태로 이미지를 바꾸는 것을 말합니다.

10 직선 인식하기

Line Detection

이미지 흐릿하게 만들기

Blur

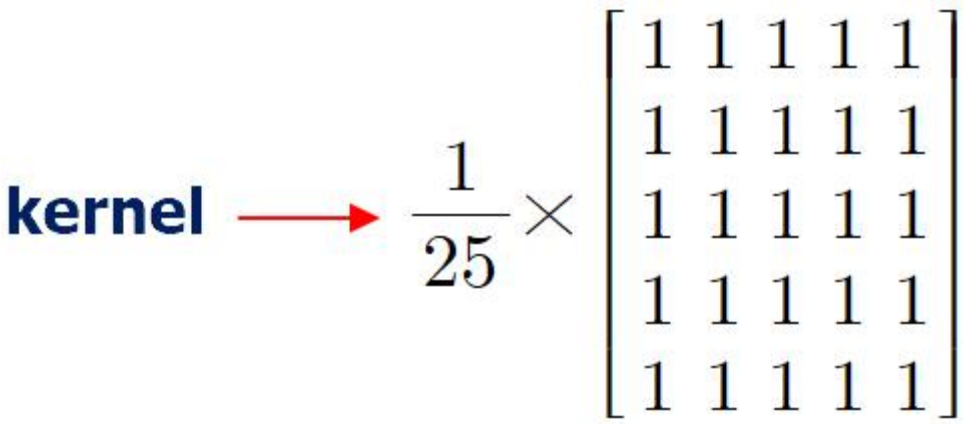
http://blog.naver.com/PostView.nhn?blogId=samsjang&logNo=220505080672&redirect=Dl

og&widgetTypeCall=true

```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('road.png')
05 : gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
06 :
07 : kernel = np.ones((5,5), np.float32)/25
08 : blur_5 = cv2.filter2D(gray, -1 , kernel)
09 :
10 : cv2.imshow('gray', gray)
11 : cv2.imshow('blur_5', blur_5)
12 :
13 : cv2.waitKey(0)
14 : cv2.destroyAllWindows()
```

2 : numpy 모듈을 np라는 이름으로 가져옵니다. numpy 모듈은 행렬이나 다차원 배열을 쉽게 처리 할 수 있도록 지원하는 파이썬의 라이브러리입니다. numpy 모듈은 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공합니다. 여기서는 7 줄에서 5x5 행렬을 만들기 위해 사용합니다.

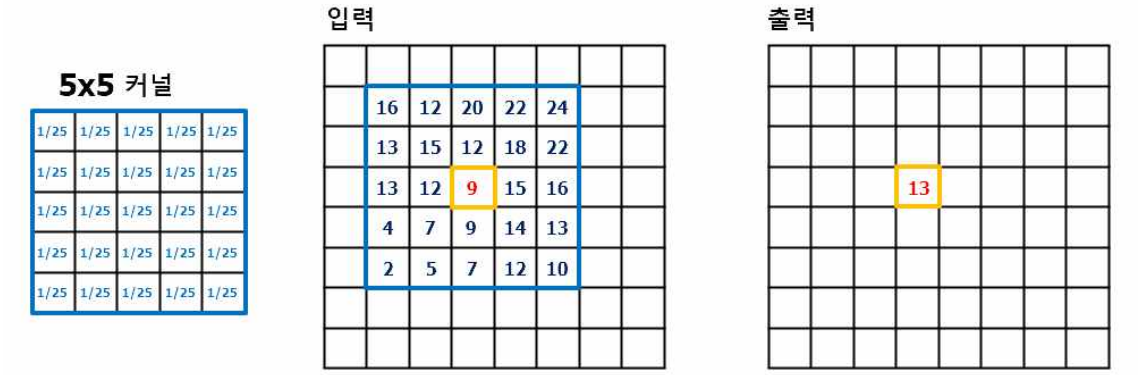
7 : np.ones 함수를 호출해 다음과 같이 실수 1.0으로 채워진 5x5 행렬을 만든 후, 25로 나눠준 후, kernel 변수가 가리키도록 합니다. kernel은 중요한 부분이라는 뜻입니다.



$$\frac{1}{25} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

np.ones 함수는 1로 채워진 주어진 모양(5x5)과 자료 형(np.float32)에 대한 배열을 내어주는 함수입니다.

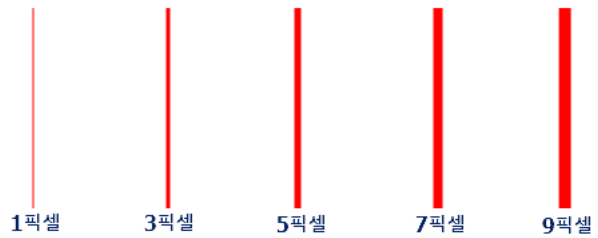
8 : cv2.filter2D 함수를 호출하여 gray 변수가 가리키는 그림을 kernel이 가리키는 정방 행렬(여기서는 5x5 행렬)을 이용하여 흐리게 만들어 결과 그림을 blur_5 변수로 가리키게 합니다. 이렇게 하면 얇은 선을 제거하고 굵은 선의 경계 부분을 흐리게 할 수 있습니다. 마치 4B 연필로 그려진 스케치 그림을 지우개로 살살 문지르면 얇은 선은 거의 없어지고 굵은 선의 경계 부분은 흐려지는 것과 같습니다. cv2.filter2D 함수의 2번째 인자는 결과 그림의 깊이(픽셀 하나의 데이터 크기)로 -1이면 입력 그림의 깊이와 같습니다. 다음은 5x5 커널을 이용하여 픽셀 하나에 대해 입력 그림을 출력 그림으로 바꾸는 과정을 나타냅니다.



입력 그림에 커널을 한 픽셀 씩 옮겨가며 연산을 수행하여 출력 그림을 완성합니다. 입력 그림의 빨간색 글씨 9로 표시된 픽셀은 5x5 커널로 덮이는 주변 픽셀을 모두 더한 후 25로 나누어 출력 픽셀 13으로 표시합니다. 즉, 9 값이 있는 픽셀은 주변 픽셀 값에 영향을 받습니다. 주변 픽셀의 평균값이 커지면 같이 커지며, 주변 픽셀의 평균값이 작아지면 같이 작아지게 됩니다.

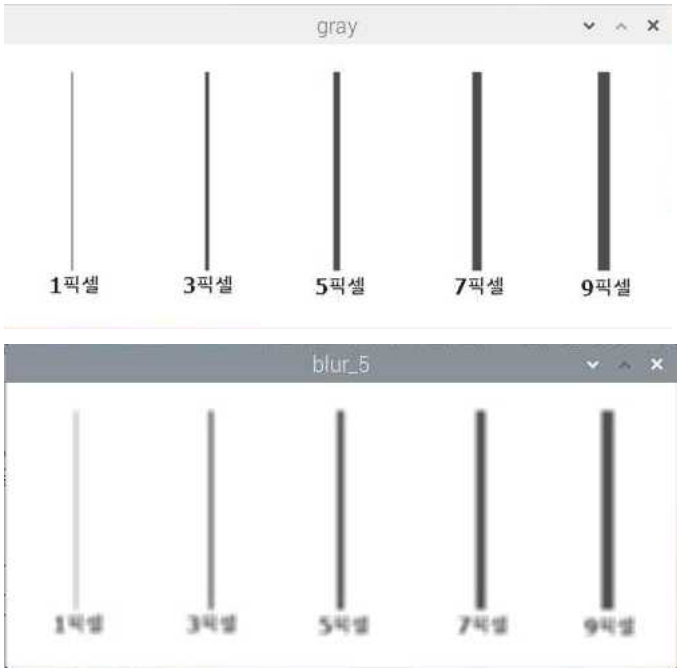
11 : cv2 모듈의 imshow 함수를 호출하여 blur_5 변수가 가리키는 그림을 화면에 보여줍니다. 첫 번째 인자인 'blur_5'는 화면에 표시된 그림의 제목을 나타내며 변경할 수 있습니다.


```
pi@raspberrypi:~/pyLabs $ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> kernel = np.ones((5,5), np.float32)/25
>>> kernel
array([[0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04],
       [0.04, 0.04, 0.04, 0.04, 0.04]], dtype=float32)
>>> quit()
pi@raspberrypi:~/pyLabs $
```



line.png

```
04 : img = cv2.imread('line.png')
```



이미지 흐릿하게 만들기 : 평균값 필터

입력 영상에서 특정 픽셀과 주변 픽셀들의 산술평균을 결과 영상의 픽셀 값으로 설정하는 필터입니다. 평균값 필터에 의해 생성되는 결과 영상은 픽셀 값의 급격한 변화가 줄어들어 날카로운 경계가 무뎌지고 잡음의 영향이 크게 사라지는 효과가 있습니다. 즉, 해당 픽셀은 주변의 값에 가까워지게 됩니다. 마치 편평한 땅 위에 뾰족하게 튀어 나온 부분이 편평해지게 되는 원리와 같습니다. 경계부분은 부드럽게 처리되어 뭉개지는 현상이 있는데, 이것은 단층 부분이 무너져 비스듬한 산 모양이 되는 것과 같습니다. 과도한 평균값 필터 적용은 사물의 경계가 흐릿해지고 사물의 구분이 어려워집니다.

```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('road.png')
05 : gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
06 :
07 : blur_5 = cv2.blur(gray, (5,5))
08 : blur_11 = cv2.blur(gray, (11,11))
09 :
10 : cv2.imshow('gray', gray)
11 : cv2.imshow('blur_5', blur_5)
12 : cv2.imshow('blur_11', blur_11)
```

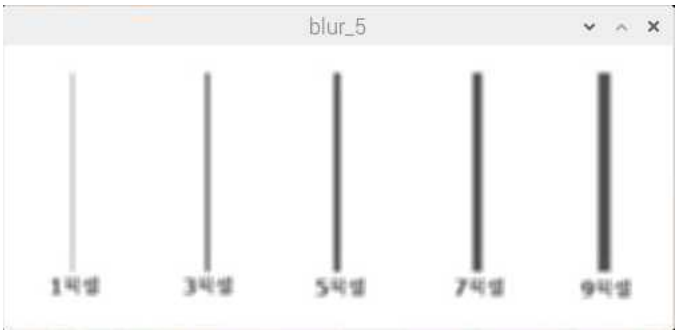
```
13 :
14 : cv2.waitKey(0)
15 : cv2.destroyAllWindows()
```

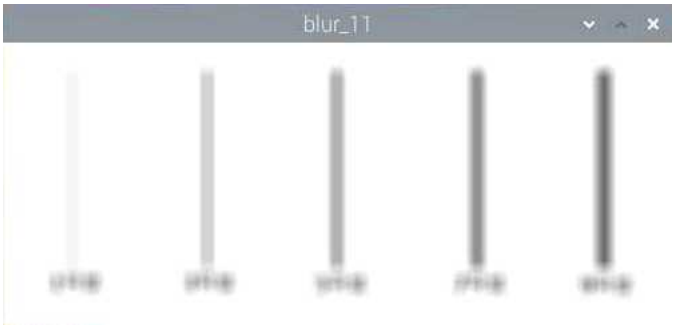
7 : cv2.blur 함수를 호출하여 gray 변수가 가리키는 그림을 5x5 크기의 커널을 이용하여 흐리게 만든 후, 결과 그림을 blur_5 변수로 가리키게 합니다. blur 함수는 커널 영역 내의 평균값으로 해당 픽셀을 대체합니다. blur 함수에서 커널은 균일한 값을 가지며, 앞에서 본 예제와 같은 결과 그림을 만들게 됩니다.

8 : cv2.blur 함수를 호출하여 gray 변수가 가리키는 그림을 11x11 크기의 커널을 이용하여 흐리게 만듭니다. 커널의 크기가 클수록 그림이 흐려지는 정도가 더 커집니다.

12 : cv2 모듈의 imshow 함수를 호출하여 blur_11 변수가 가리키는 그림을 화면에 보여줍니다.


```
04 : img = cv2.imread('line.png')
```





이미지 흐릿하게 만들기 : 메디안 - 눈이 뿌려지는 그림

입력영상에서 자기 자신 픽셀과 주변 픽셀 값의 중간값을 결과영상 픽셀 값으로 지정하는 필터로 잡음 픽셀 값이 주변 픽셀 값과 큰 차이가 있는 경우(소금 또는 후추 잡음, 픽셀값이 0 또는 255로 변경되는 형태의 잡음)에 효과적입니다.

```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('road.jpg')
05 : img_noise = img.copy()
06 : N =50000
07 : idx1 = np.random.randint(img.shape[0], size =N)
08 : idx2 = np.random.randint(img.shape[1], size =N)
09 : img_noise[idx1, idx2] =255
10 :
11 : gray_noise = cv2.cvtColor(img_noise, cv2.COLOR_BGR2GRAY)
12 :
13 : blur_5 = cv2.medianBlur(gray_noise, 5)
14 : blur_11 = cv2.medianBlur(gray_noise, 11)
15 :
16 : cv2.imshow('gray_noise', gray_noise)
17 : cv2.imshow('blur_5', blur_5)
18 : cv2.imshow('blur_11', blur_11)
19 :
20 : cv2.waitKey(0)
21 : cv2.destroyAllWindows()
```

5 : img에 대해 copy 함수를 호출하여 img가 가리키는 그림을 복사한 후, img_noise 변수로 가리키게 합니다.

6 : N 변수를 생성한 후, 50000으로 초기화합니다.

7 : np.random.randint 함수를 호출한 후, img 변수가 가리키는 그림의 세로 크기 범위 내에서 N 개의 임의의 정수 배열을 생성한 후, idx1 변수로 가리키게 합니다.

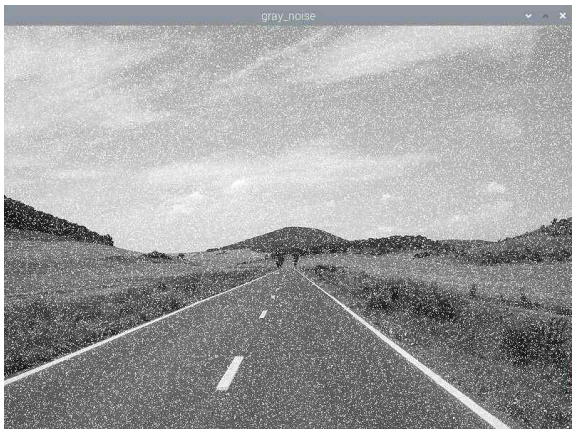
8 : np.random.randint 함수를 호출한 후, img 변수가 가리키는 그림의 가로 크기 범위 내에서 N 개의 임의의 정수 배열을 생성한 후, idx2 변수로 가리키게 합니다.

9 : img_noise 변수가 가리키는 그림의 idx1, idx2 정수 배열이 가리키는 위치의 픽셀 값을 255로 설정합니다. 이렇게 하면 NxN 개의 픽셀이 하얀 색으로 바뀝니다.

13 : cv2.medianBlur 함수를 호출하여 gray_noise 변수가 가리키는 그림을 5x5 크기의 커널을 이용하여 흐리게 만든 후, 결과 그림을 blur_5 변수로 가리키게 합니다. 커널의 크기는

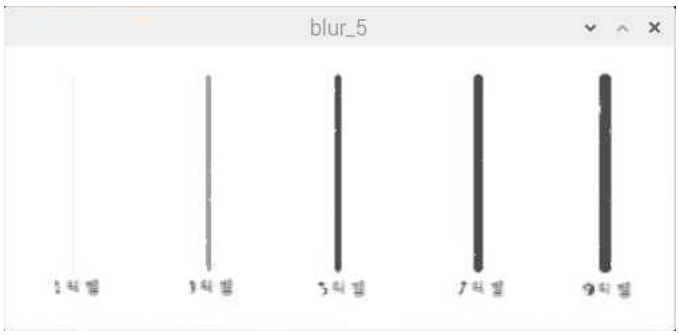
홀수로 1보다 큰 값이어야 합니다. 즉, 3, 5, 7, ...과 같은 값이어야 합니다. medianBlur 함수는 평균이 아닌 커널 영역 내의 중앙값으로 해당 픽셀을 대체합니다. 이 방법은 작은 점 모양의 잡음을 제거하는데 효과적입니다.
14 : cv2.medianBlur 함수를 호출하여 gray_noise 변수가 가리키는 그림을 11 크기의 커널을 이용하여 흐리게 만듭니다. 커널의 크기가 클수록 그림이 흐려지는 정도가 더 커집니다.

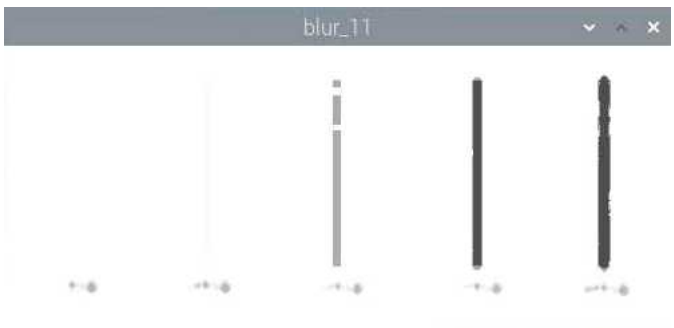
점 잡음이 있는 그림



```
04 : img = cv2.imread('line.png')
05 : img_noise = img.copy()
06 : N =5000
```

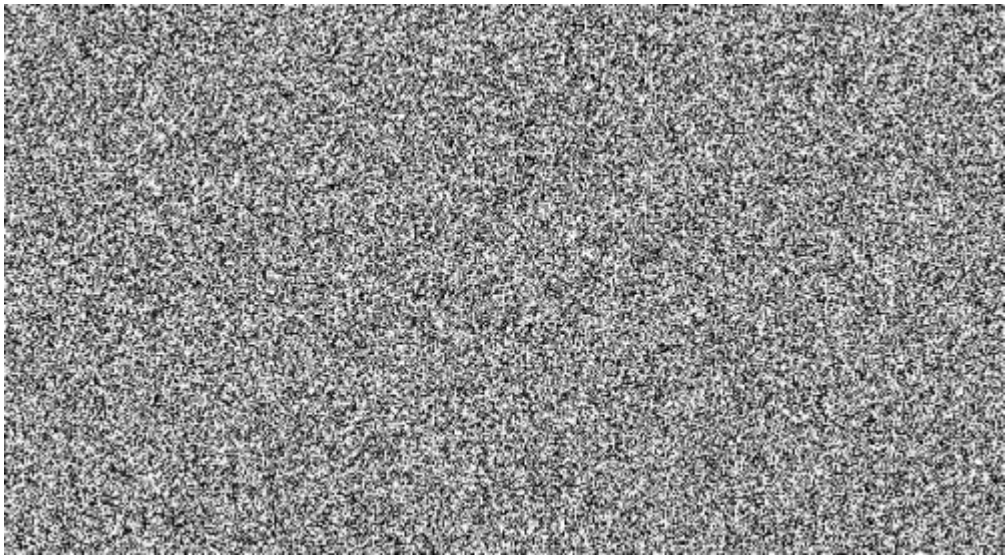
점 잡음이 있는 그림 (N=5000)





이미지 흐릿하게 만들기 : 가우시안 필터 - 전기적 잡음

가우시안 필터는 평균값 필터보다 자연스러운 블러링 결과를 생성합니다. 가우시안 분포함수로 생성한 필터 마스크를 사용하는 필터링 기법입니다. 커널의 중앙부는 비교적 큰 값을 가지며 주변부로 갈수록 0에 가까운 작은 값을 가집니다. 필터링 대상 픽셀과 멀리 떨어져 있는 주변부에 가중치를 조금 주는 것으로 가중 평균을 구하는 것과 같습니다. 커널이 가중 평균을 구하기 위한 가중치 행렬 역할을 합니다.

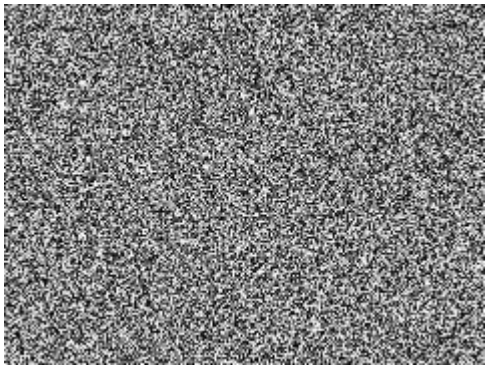


가우시언 잡음(gaussian noise) : 전기적 잡음에 의해 발생하는 자연스러운 잡음생성으로 잡음의 분포가 종모양(정규분포 곡선 모양)을 띈다. 어떠한 환경에서건 촬영된 영상은 크든 작든 원치 않는 잡음을 포함할 수 있다. 이러한 잡음은 광학적 신호를 전기적 신호로 변환하는 센서에서 주로 추가된다. 빛에 대한 정보를 전기신호로 변환, 물리적 신호를 전기적 신호로 변환

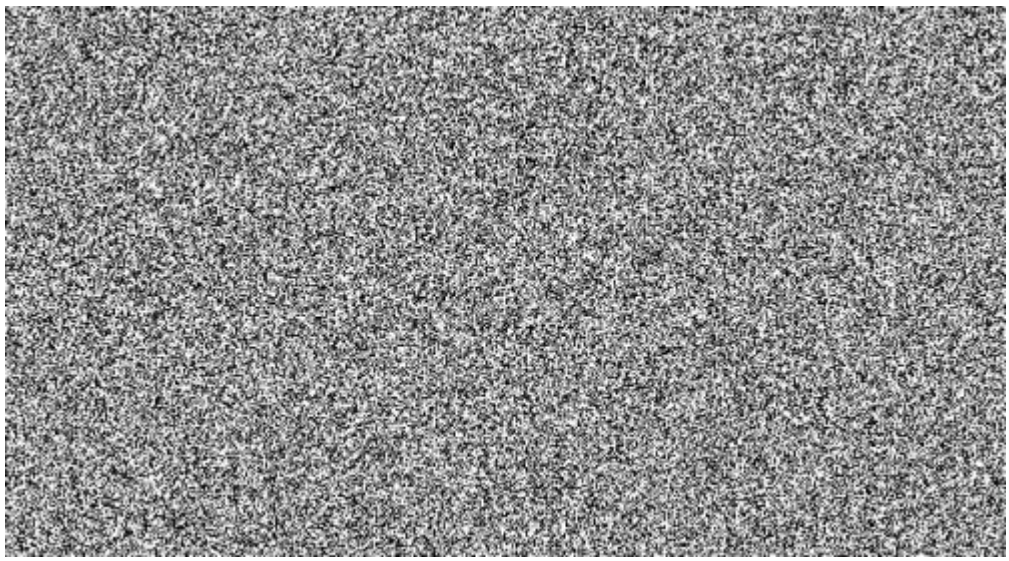
필터링(거르기)이란 영상에서 원하는 정보만 통과시키고 나머지는 걸러내는 작업이다. 이를 통해 잡음을 걸러내어 영상을 깔끔하게 만들거나 부드러운 느낌을 걸러내서 날카로운 느낌의 영상을 만들 수 있다.

필터링은 마스크, 커널, 윈도우, 필터 등으로 불리는 작은 크기의 행렬을 이용하는데 필터링 연산의 결과는 행렬의 모양과 원소의 값에 의해 결정되며 진한 색을 갖는 고정점을 기준 픽셀로 필터링 작업을 수행한다.

백색소음 - 소리 - 음파 (정보:잡음)
전파(정보:잡음)
영상 전송은 전파를 통해(정보+잡음)
잡음은 종모양의 정규분포 곡선을 그린다. 주 잡음을 중심으로 부 잡음들이 정규분포 곡선을 그린다.
잡음의 분포는 정규분포 곡선을 그린다.
가우시안 필터 - 산 모양의 필터, 종 모양의 필터

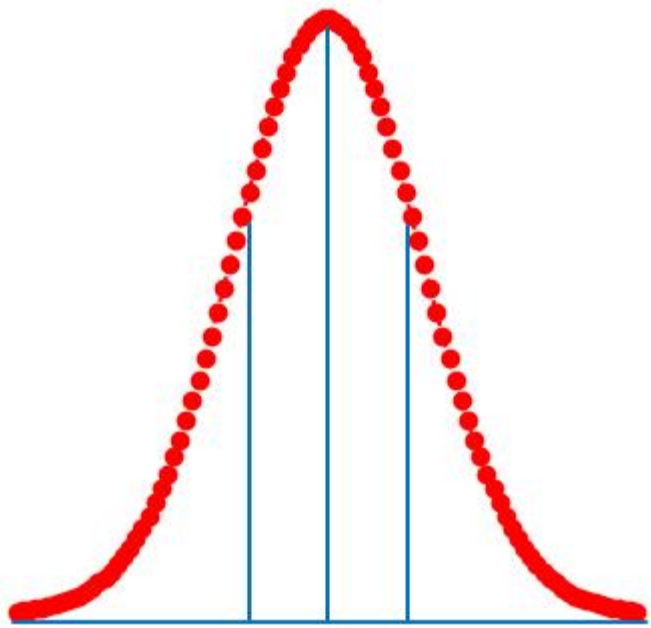


텔레비전이나 라디오에서 들을 수 있는 치~익 소리를 백색소음 또는 백색잡음이라고 합니다. 이러한 종류의 백색소음에는 고주파가 섞여 있어 듣기에 편하지 않습니다. 빗소리, 파도소리, 물 흐르는 소리, 장작 타는 소리 등은 저주파의 백색소음으로 듣기에 편안하며 안정감을 줍니다. 백색소음이란 여러 주파수의 소리가 합쳐진 패턴 없는 소음의 일종으로 인간의 귀로 들을 수 있는 모든 주파수의 소리를 합쳐놓은 소리라는 뜻입니다. 여러 가지 빛을 섞으면 흰색이 되는 것처럼 모든 소리를 다 합쳤다는 뜻에서 '백색(白色)'이란 이름이 붙습니다. 소리에 백색소음이 있는 것처럼 영상에도 백색잡음이 있습니다. 다음은 백색잡음 영상을 나타냅니다.

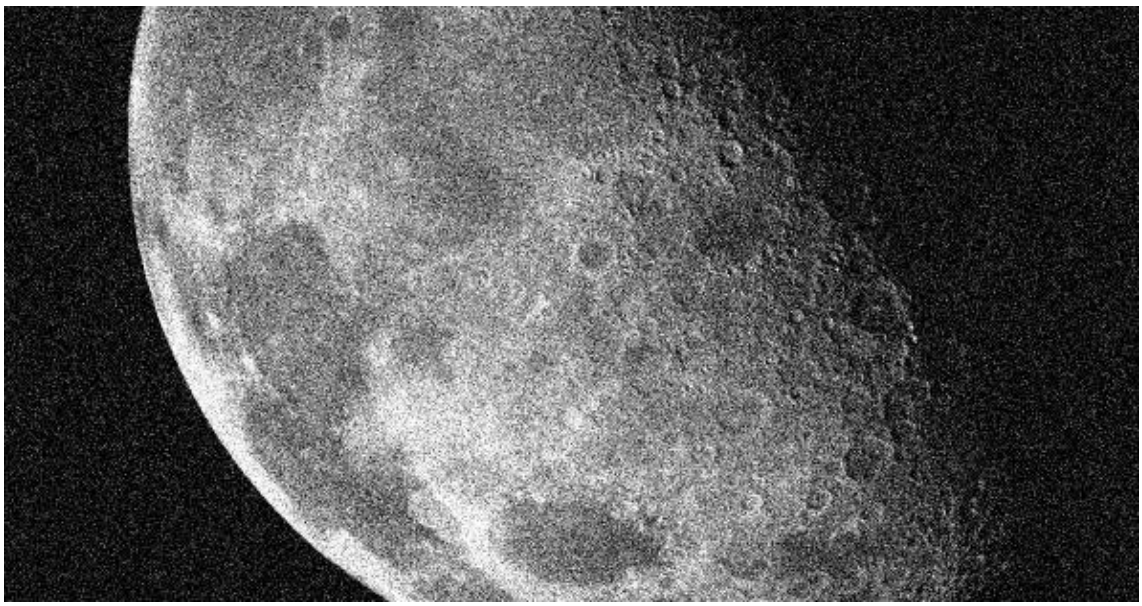


이러한 잡음은 전기적 잡음에 의해 발생하는 자연스러운 잡음입니다. 어떤 환경에서든 촬영된 영상은 크든 작든 원치 않는 잡음을 포함합니다. 이러한 잡음은 카메라 센서에서 빛을 전기적 신호로 변환하는 과정 또는 영상 데이터를 전송하는 과정에서 전도체 내부에 있는 전자의 자유운동에 의해 발생합니다. 열에 따른 전자들의 불규칙한 움직임, 즉, 열 교란에 의한 전자들의 불규칙한 움직임이 잡음으로 나타나게 됩니다. 이러한 잡음은 모든 형태의 전자장비와 매체에서 나타나며 그 크기는 절대 온도에 비례합니다.

백색잡음의 분포는 다음과 같이 가운데가 봉긋한 예쁜 산 모양 또는 종 모양과 같은 정규분포 곡선의 형태를 띱니다.



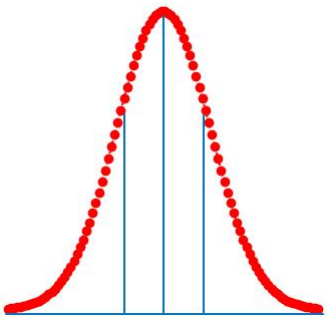
정규분포는 가우스분포라고도 하며 연속확률 분포의 하나로, 혈압, 키, 몸무게 등 현실 세계나 자연 현상에서 많이 관측되거나 측정되는 자료의 분포 형태를 나타냅니다. 예를 들어, 한국인 20대 여자의 키, 50대 남자의 혈압 수치 등은 정규분포 곡선을 그리게 됩니다. 이러한 정규분포는 평균값을 기준으로 좌우로 대칭 형태를 이룹니다. 백색잡음도 물리적 현상의 하나로 잡음의 주파수가 정규분포 곡선을 그립니다. 다음은 가우스잡음이 섞인 달 영상입니다.



영상에 포함되는 대부분의 잡음은 가우시안 잡음으로 이 잡음을 제거하기 위해 가우시안 필터를 사용합니다. 가우시안 필터는 잡음제거와 함께 영상이나 이미지를 흐리게 처리하는데 픽셀 값의 변화가 크지 않는 영역, 즉, 평탄한 영역에서는 잡음을 크게 줄여주지만 급하게 변하는 영역, 즉, 경계 영역에서는 윤곽이 흐려지는 단점이 있습니다.

```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('road.png')
05 : img_noise = (img/255 + np.random.normal(scale=0.2, size=img.shape))*255
06 : img_noise = np.clip(img_noise, 0, 255).astype('uint8')
07 : gray_noise = cv2.cvtColor(img_noise, cv2.COLOR_BGR2GRAY)
08 :
09 : blur_5 = cv2.GaussianBlur(gray_noise, (5,5), 2)
10 : blur_11 = cv2.GaussianBlur(gray_noise, (11,11), 2)
11 :
12 : cv2.imshow('gray_noise', gray_noise)
13 : cv2.imshow('blur_5', blur_5)
14 : cv2.imshow('blur_11', blur_11)
15 :
16 : cv2.waitKey(0)
17 : cv2.destroyAllWindows()
```

5 : img/255는 img가 가리키는 모든 그림을 255로 나누어 0.0~1.0 사이의 값으로 바꾸는 부분입니다. np.random.normal 함수는 표준정규분포를 이루는 임의의 값들을 생성해 냅니다. 첫 번째 인자인 0.2는 표준편차를 두 번째 인자는 생성해 낼 정규분포 값의 크기를 나타냅니다.

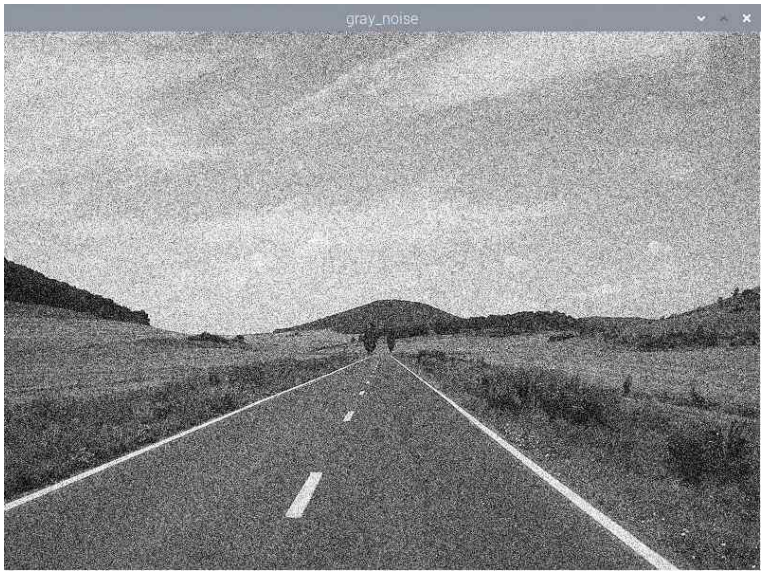


여기서는 표준정규분포를 이루는 백색잡음을 생성해 내는 역할을 합니다. 표준정규분포를 이루는 백색잡음을 255로 나눈 기존의 그림에 더해주어 백색잡음을 그림에 추가한 후, 255를 곱해 원래 그림의 형태로 변경한 후, img_noise 변수로 가리키도록 합니다. 이제 img_noise 변수는 백색잡음이 더해진 그림을 가리키게 됩니다. 그림의 픽셀 값은 실수 값 형태입니다.

6 : np.clip 함수를 이용해 img_noise가 가리키는 그림의 픽셀 값을 0~255 사이 값이 되도록 합니다. np.clip 함수는 양쪽 범위를 지정해 범위 바깥 부분을 잘라내는 함수입니다. 여기서는 0보다 작은 값은 0으로 보정하고, 255보다 큰 값은 255로 보정합니다. astype 함수는 값의 형태를 변환하는 함수로 여기서는 픽셀 값을 8비트 부호 없는 양의 정수 값으로 변경합니다. 즉, 백색잡음을 더하는 과정에서 변환된 실수 값 형태의 픽셀 값을 8비트 양의 정수 값으로 바꿉니다.

9 : cv2.GaussianBlur 함수를 호출하여 gray_noise 변수가 가리키는 백색잡음이 더해진 그림을 흐리게 만듭니다. 두 번째 인자는 커널의 크기로 5x5이며, 세 번째 인자는 가우시안

필터의 표준편차입니다.
10 : cv2.GaussianBlur 함수를 호출하여 gray_noise 변수가 가리키는 백색잡음이 더해진 그림을 11x11 크기의 커널로 흐리게 만듭니다.

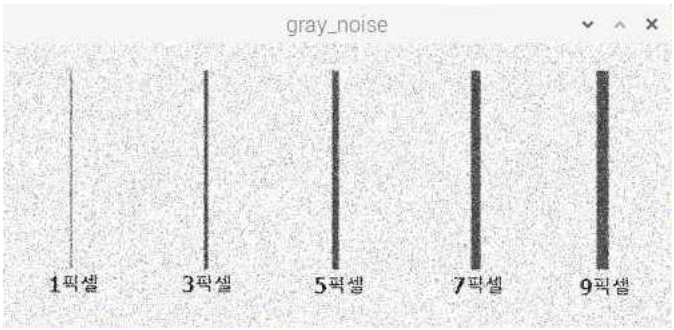


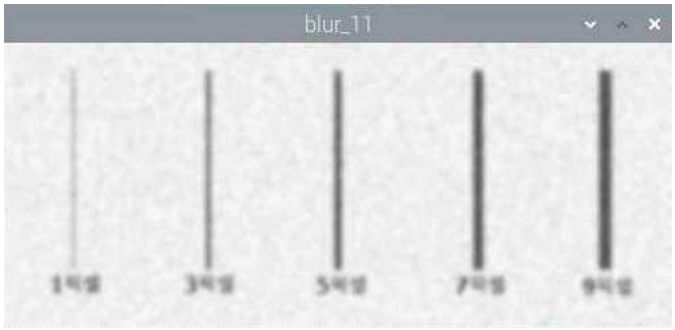
<백색잡음이 있는 그림>

<가우시안 필터(5x5)를 적용한 그림>

<가우시안 필터(11x11)를 적용한 그림>

```
04 : img = cv2.imread('line.png')
```



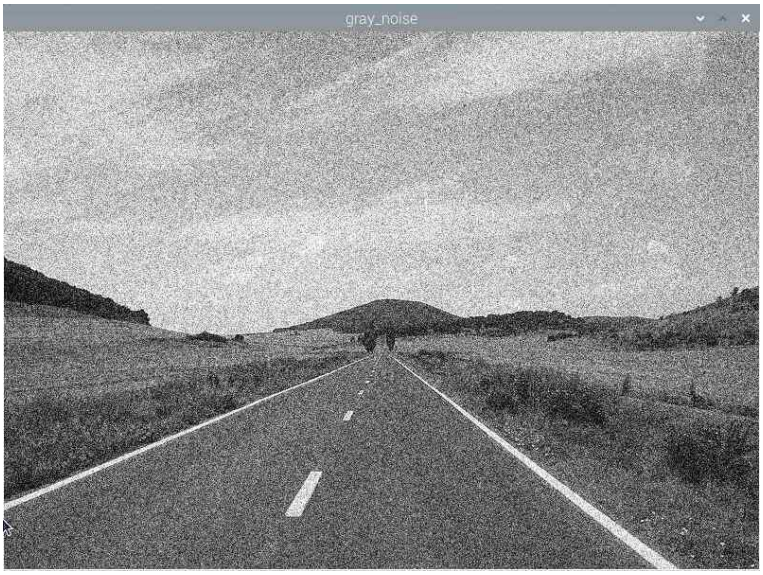


이미지 흐릿하게 만들기 : 양방향 필터

잡음과 경계를 함께 제거하는 가우시안 필터의 단점을 보완하기 위한 필터입니다. 경계 보전 잡음 제거 필터로 토마시가 제안한 알고리즘입니다.

```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('road.png')
05 : img_noise = (img/255 + np.random.normal(scale=0.2, size=img.shape))*255
06 : img_noise = np.clip(img_noise, 0, 255).astype('uint8')
07 : gray_noise = cv2.cvtColor(img_noise, cv2.COLOR_BGR2GRAY)
08 :
09 : blur_g = cv2.GaussianBlur(gray_noise, (11,11), 2)
10 : blur_b = cv2.bilateralFilter(gray_noise, 11, 75, 75)
11 :
12 : cv2.imshow('gray_noise', gray_noise)
13 : cv2.imshow('blur_g', blur_g)
14 : cv2.imshow('blur_b', blur_b)
15 :
16 : cv2.waitKey(0)
17 : cv2.destroyAllWindows()
```

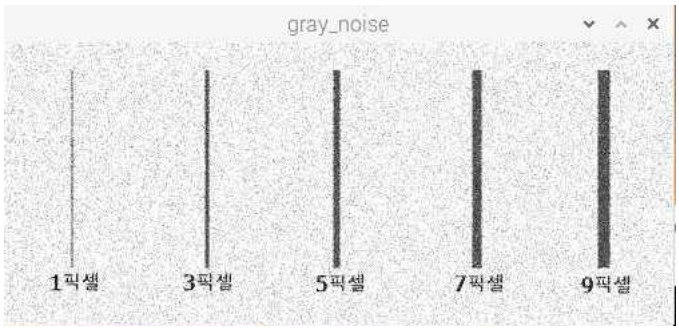
10 : cv2.bilateralFilter 함수를 호출하여 gray_noise 변수가 가리키는 백색잡음이 더해진 그림을 흐리게 만듭니다. bilateralFilter 함수는 경계선은 명확하게 만듭니다. 두 번째 인자는 커널의 크기로 11이며, 세 번째 인자는 색 공간 표준편차로 값이 크면 색이 많이 달라도 픽셀들이 서로 영향을 미칩니다. 네 번째 인자는 거리 공간 표준편차로 값이 크면 멀리 떨어져 있는 픽셀들도 서로 영향을 미칩니다.

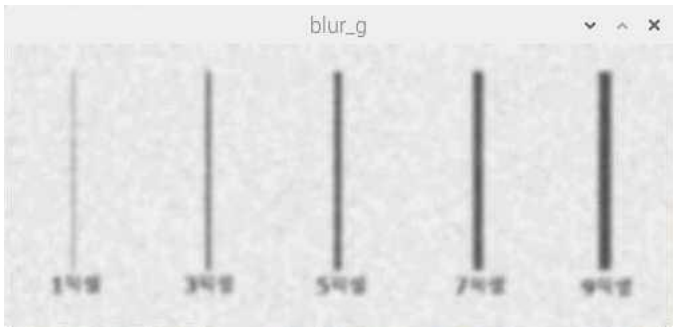


<백색잡음이 있는 그림>

<가우시안 필터(5x5)를 적용한 그림>

<양방향 필터(11)를 적용한 그림>



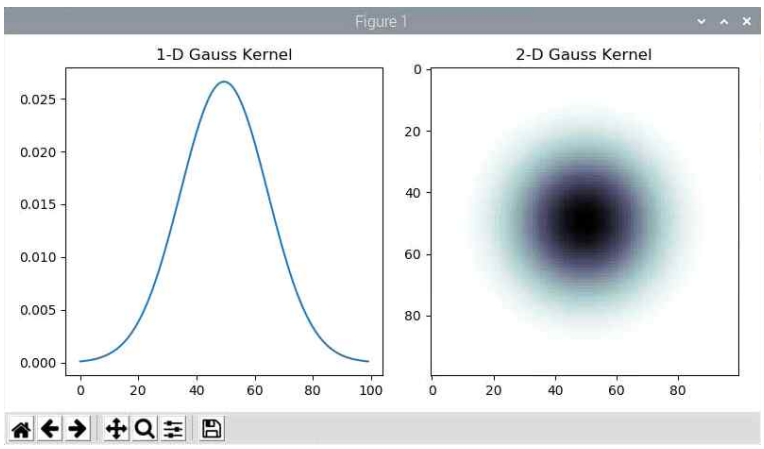


```
pi@raspberrypi:~/pyLabs $ sudo apt install python3-matplotlib
Do you want to continue? [Y/n] y
```

```
01 : import cv2
02 : import matplotlib as mpl
03 : import matplotlib.pylab as plt
04 :
05 : gauss1d =cv2.getGaussianKernel(100,15)
06 : gauss2d = gauss1d @ gauss1d.T
07 :
08 : plt.figure(figsize =(8,4))
09 : plt.subplot(1,2,1)
10 : plt.plot(gauss1d) #, 'ro:')
11 : plt.title("1-D Gauss Kernel")
12 :
13 : plt.subplot(122)
14 : plt.imshow(gauss2d, cmap =mpl.cm.bone_r)
15 : plt.title("2-D Gauss Kernel")
16 :
17 : plt.tight_layout()
18 : plt.show()
```

#파이썬 3.5에서 @ 연산자는 행렬 곱셈을위한 중위 연산자로 추가되었습니다.

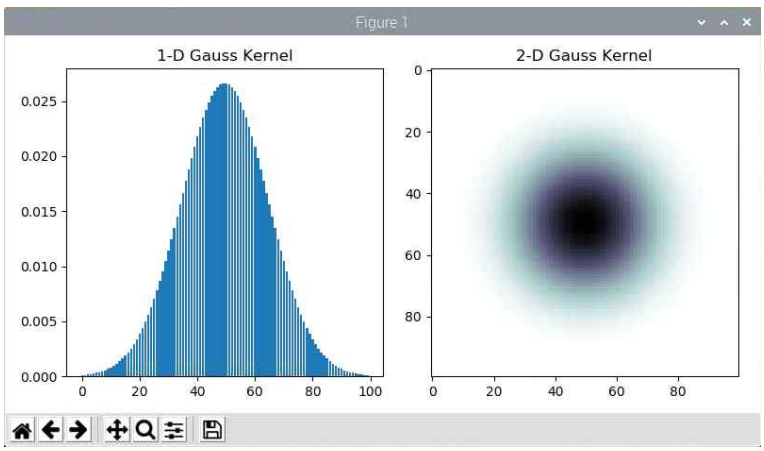
5 : cv2.getGaussianKernel 함수를 호출하여 가우시안 분포 형태의 커널을 생성합니다. 커널을 필터가 됩니다. 첫 번째 인자는 커널의 크기를 나타내며, 두 번째 인자는 가우시안 표준편차로 첫 번째 인자인 커널의 크기를 기준으로 합니다.




```
01 : import cv2
02 : import matplotlib as mpl
03 : import matplotlib.pylab as plt
```

```
04 :
05 : gauss1d =cv2.getGaussianKernel(100,15)
06 : gauss2d = gauss1d @ gauss1d.T
07 : print(len(gauss1d))
08 : print(gauss1d[99][0])
09 : print(type(gauss1d))
10 :
11 : plt.figure(figsize=(8,4))
12 : plt.subplot(1,2,1)
13 :
14 : x = [i for i in range(0,100)]
15 : y = [gauss1d[i][0] for i in range(0,100)]
16 : plt.bar(x, y)
17 : plt.title("1-D Gauss Kernel")
18 :
19 : plt.subplot(122)
20 : plt.imshow(gauss2d, cmap=mpl.cm.bone_r)
21 : plt.title("2-D Gauss Kernel")
22 :
23 : plt.tight_layout()
24 : plt.show()
```

```
pi@raspberrypi:~/pyLabs $ sudo python3 test_2.py
100
0.00011493633990452316
<class 'numpy.ndarray'>
```



선 찾기 : 캐니 알고리즘

edge

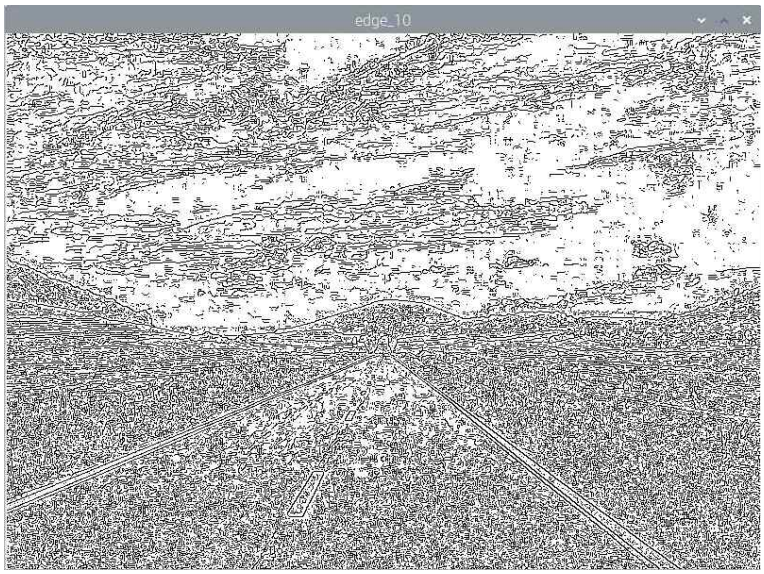
https://blog.naver.com/samsjang/220507996391

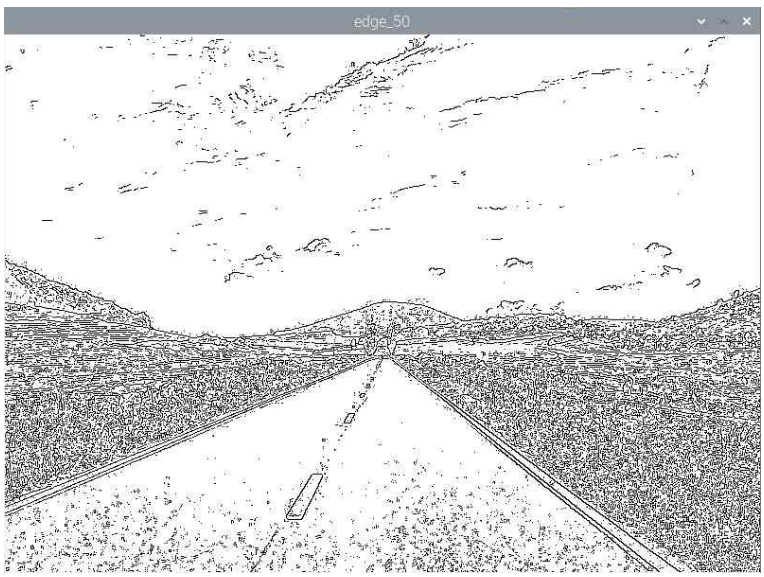
```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('road.png')
05 : gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
06 :
07 : edge_10 = cv2.Canny(gray, 10, 10)
08 : edge_50 = cv2.Canny(gray, 50, 50)
09 : edge_100 = cv2.Canny(gray, 100, 100)
10 : edge_200 = cv2.Canny(gray, 200, 200)
11 :
12 : edge_10 = cv2.bitwise_not(edge_10)
13 : edge_50 = cv2.bitwise_not(edge_50)
14 : edge_100 = cv2.bitwise_not(edge_100)
15 : edge_200 = cv2.bitwise_not(edge_200)
16 :
17 : cv2.imshow('gray', gray)
18 : cv2.imshow('edge_10', edge_10)
19 : cv2.imshow('edge_50', edge_50)
20 : cv2.imshow('edge_100', edge_100)
21 : cv2.imshow('edge_200', edge_200)
22 :
23 : cv2.waitKey(0)
24 : cv2.destroyAllWindows()
```

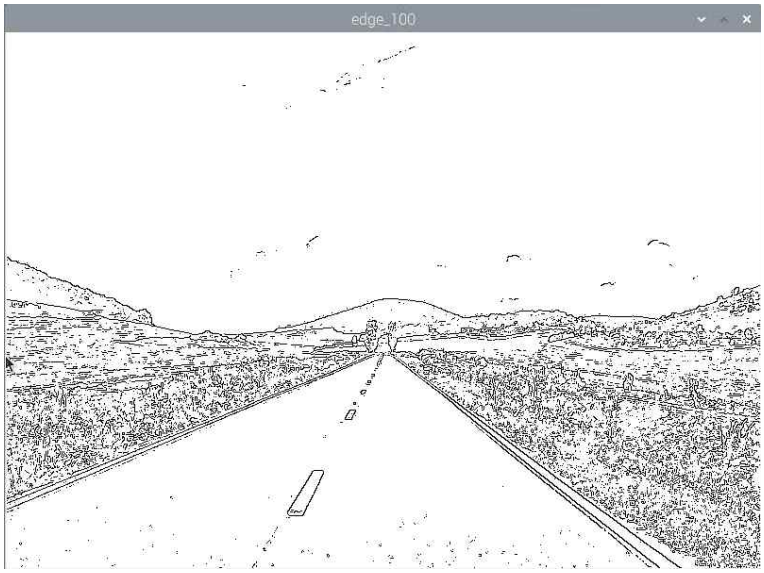
7~10 : cv2 모듈의 Canny 함수를 호출하여 gray 변수가 가리키는 그림을 Canny 알고리즘을 이용하여 모서리 경계선을 찾아 edge_10, edge_50, edge_100, edge_200 변수가 가리키도록 합니다. 두 번째, 세 번째 인자는 x, y 축에 대한 threshold 값으로 값이 클수록 명확한 경계만 추출합니다.

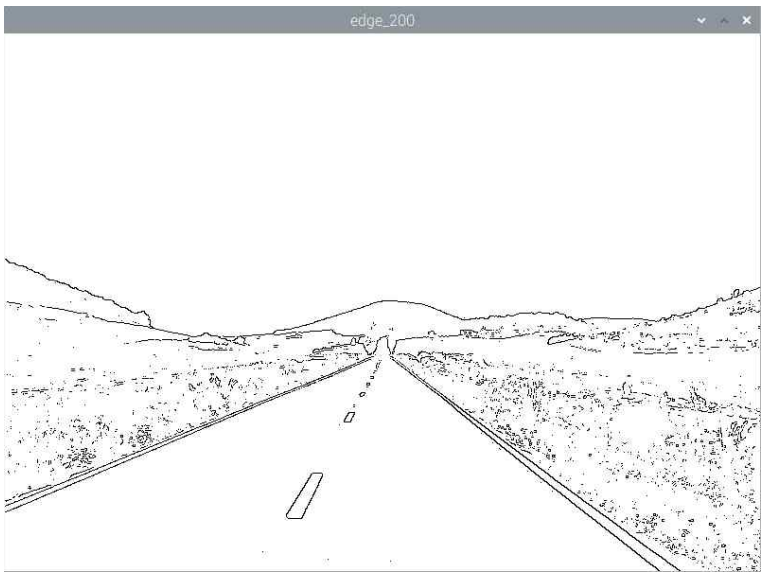
12~15 : cv2 모듈의 bitwise_not 함수를 호출하여 edge_10, edge_50, edge_100, edge_200 변수가 가리키는 그림을 반전시킵니다.


gray

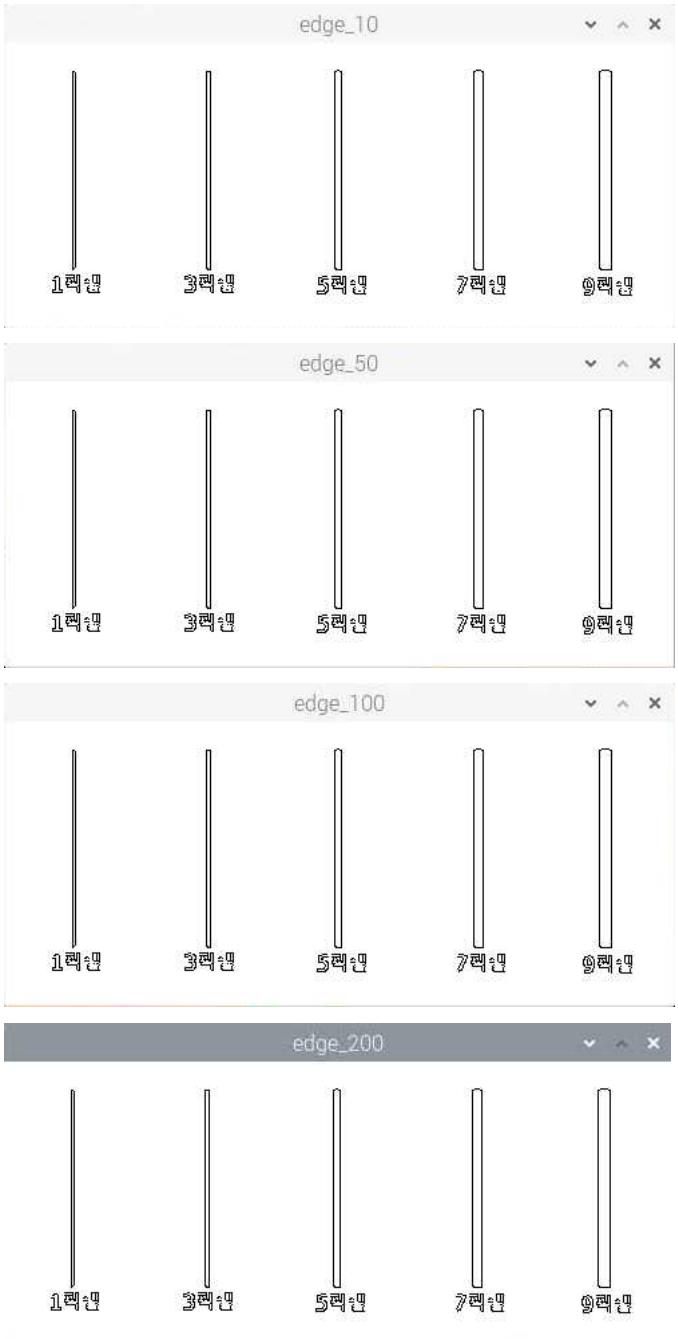

edge_10


edge_50



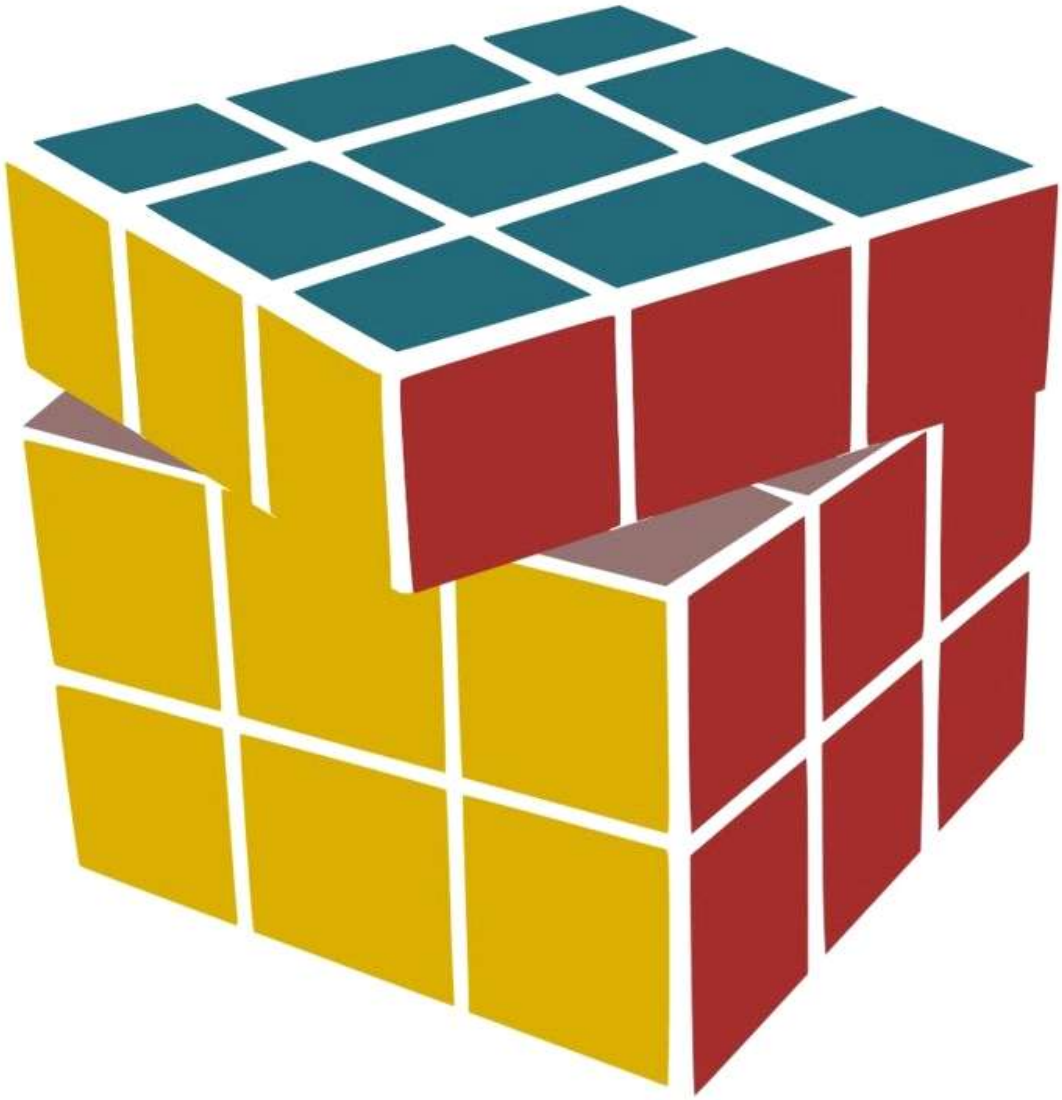



```
04 : img = cv2.imread('line.png')
```



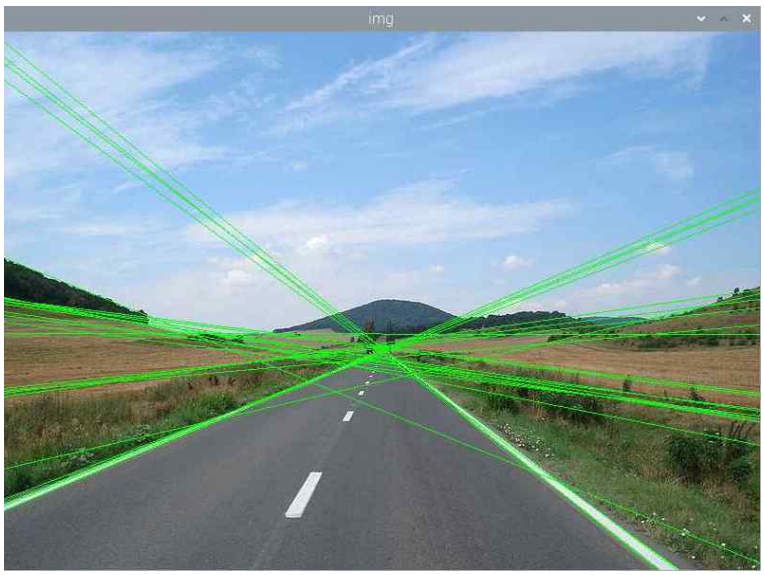
직선 찾기

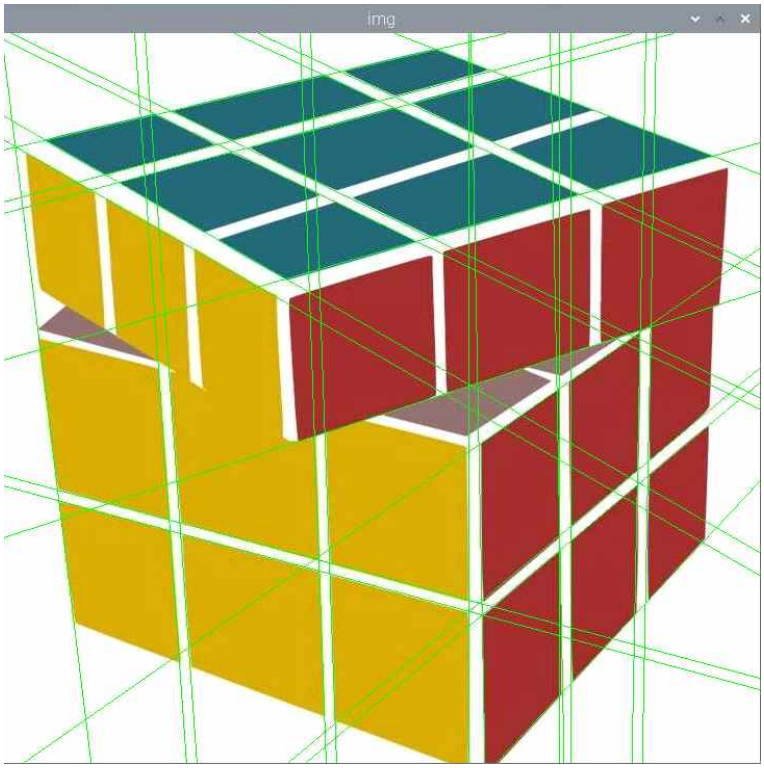
houghlines
https://m.blog.naver.com/samsjang/220588392347



```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('cube.png')
05 : gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
06 :
07 : edges = cv2.Canny(gray, 150, 150, apertureSize =3)
08 : lines = cv2.HoughLines(edges, 1, np.pi /180, 140)
09 :
10 : for line in lines:
11 :     r,theta = line[0]
12 :     a = np.cos(theta)
13 :     b = np.sin(theta)
```

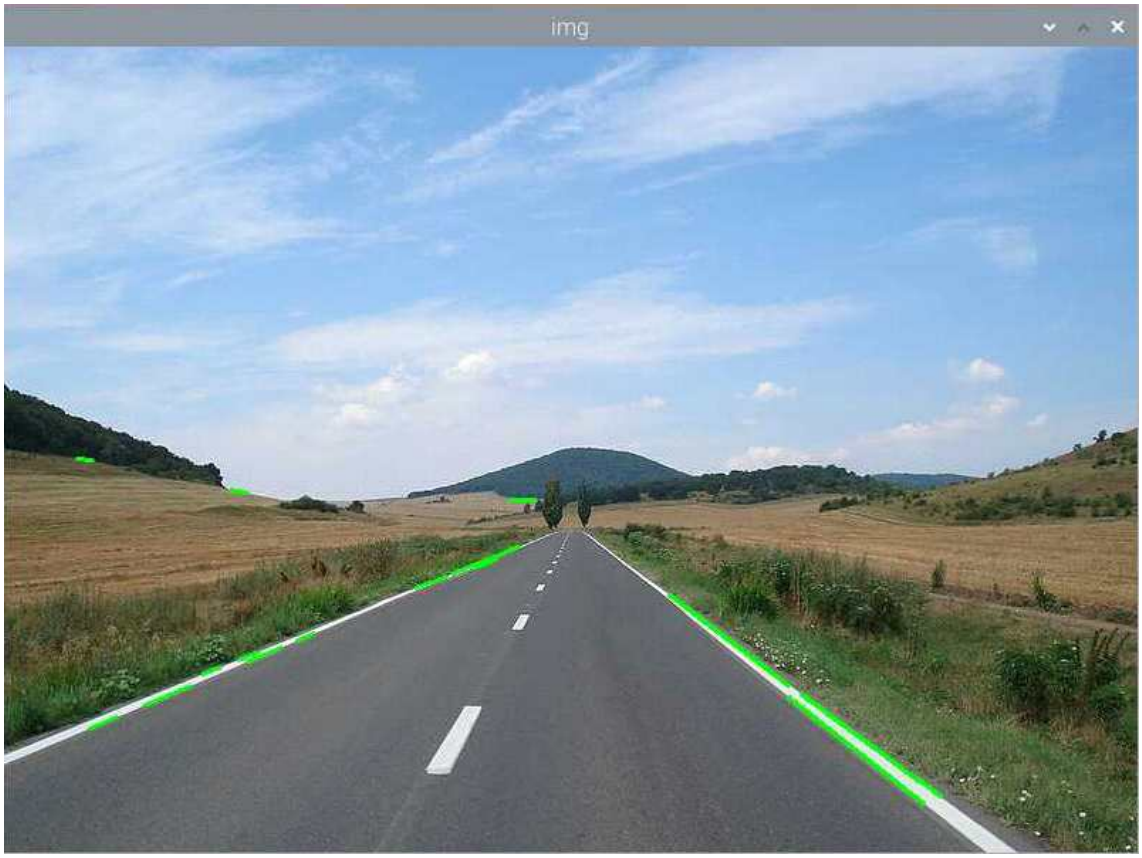
```
14 :     x0 = a *r
15 :     y0 = b *r
16 :     x1 =int(x0 +1000 *(-b))
17 :     y1 =int(y0 +1000 *a)
18 :     x2 =int(x0 -1000 *(-b))
19 :     y2 =int(y0 -1000 *a)
20 :
21 :     cv2.line(img, (x1,y1), (x2,y2), (0,255,0), 1)
22 :
23 : cv2.imshow('img', img)
24 :
25 : cv2.waitKey(0)
26 : cv2.destroyAllWindows()
```






```
01 : import cv2
02 : import numpy as np
03 :
04 : img = cv2.imread('road.png')
05 : gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
06 :
07 : edges = cv2.Canny(gray, 150, 150, apertureSize =3)
08 : lines = cv2.HoughLinesP(edges, 1, np.pi /180, 140, 100, 10)
09 :
10 : for line in lines:
11 :     x1,y1,x2,y2 = line[0]
12 :     cv2.line(img, (x1,y1), (x2,y2), (0,255,0), 2)
13 :
14 : cv2.imshow('img', img)
15 :
16 : cv2.waitKey(0)
```

04 카메라로 차선 인식하기



10 직선 인식하기

Line Detection

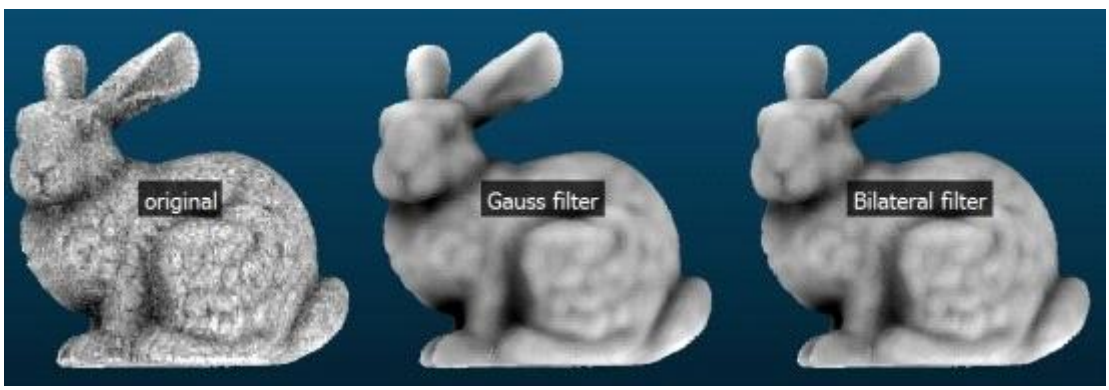
실시간 직선 인식하기

종합

```
01 : import cv2
02 : import numpy as np
03 : import math
04 :
05 : minLineLength =5
06 : maxLineGap =10
07 :
08 : cap = cv2.VideoCapture(0)
```

```
09 :
10 : while cap.isOpened():
11 :     ret,img = cap.read()
12 :
13 :     if ret:
14 :             gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
15 :             blurred = cv2.GaussianBlur(gray, (5, 5), 0)
16 :             edged = cv2.Canny(blurred, 85, 85)
17 :             lines = cv2.HoughLinesP(edged,1,np.pi /180,10,
18 :                     minLineLength,maxLineGap)
19 :             if len(lines):
20 :                     for x in range(0, len(lines)):
21 :                             for x1,y1,x2,y2 in lines[x]:
22 :                                     cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
23 :
24 :             cv2.imshow('Video Capture', img)
25 :
26 :             key = cv2.waitKey(1) &0xFF #1ms wait
27 :             if key ==27: #ESC
28 :                     break
29 :
30 : cap.release()
31 : cv2.destroyAllWindows()
```

15 : cv2 모듈의 GaussianBlur 함수를 호출하여 gray 변수가 가리키는 그림을 가우시안 필터로 흐릿하게 처리한 후, 바뀐 그림을 blurred 변수가 가리키도록 합니다. 가우시안 필터로 그림을 흐릿하게 처리하는 이유는 세세한 선을 없애고 굵은 선만 남기기 위해서입니다. 마치 4B 연필로 그려진 스케치 그림을 지우개로 살살 문지르는 것과 같습니다. GaussianBlur 함수의 두 번째 인자는 가우시안 커널의 크기로 흐릿하게 처리하는 영역의 크기를 결정합니다. 값은 양의 홀수로만 줄 수 있으며 값이 클수록 흐릿한 정도가 심해집니다. 커널의 크기는 스케치 그림을 흐릿하게 처리하는 지우개의 크기와 같습니다. 세 번째 인자는 표준편차입니다.



https://m.blog.naver.com/PostView.nhn?blogId=ai5479&logNo=221013064633&proxyRefe

rer=https:%2F%2Fwww.google.com%2F

https://iskim3068.tistory.com/41

https://datascienceschool.net/view-notebook/c4121d311aa34e6faa84f62ef06e43b0/

16 : cv2 모듈의 Canny 함수를 호출하여 blurred 변수가 가리키는 그림을 Canny 알고리즘을 이용하여 모서리 경계선을 찾아 edged 변수가 가리키도록 합니다. 두 번째, 세 번째 인자는 x, y 축에 대한 threshold 값으로 값이 클수록 명확한 경계만 추출합니다.

https://blog.naver.com/samsjang/220507996391

17 : cv2 모듈의 HoughLinesP 함수를 호출하여 edged 변수가 가리키는 그림을 확률적 허프 변환 알고리즘을 이용하여 선 조각을 찾아 lines 변수가 가리키도록 합니다. 두 번째 인자

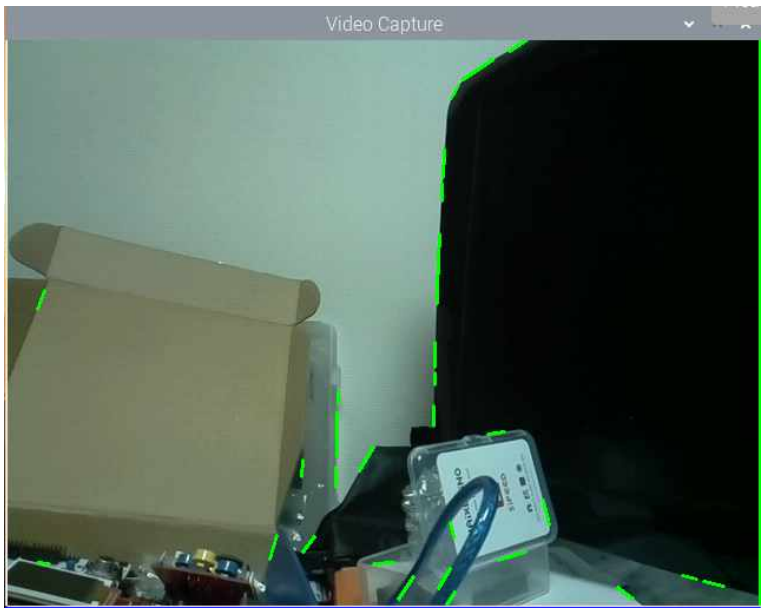
는 선 조각을 찾기 위한 픽셀 단위의 거리, 세 번째 인자는 라디안 단위의 각도로 2도, 네 번째 인자는 선 조각으로 판단된 횟수를 나타냅니다. 다섯 번째 인자는 추출할 최소 선의 길이, 여섯 번째 인자는 추출할 선상에서의 점간의 최대 거리를 나타냅니다.

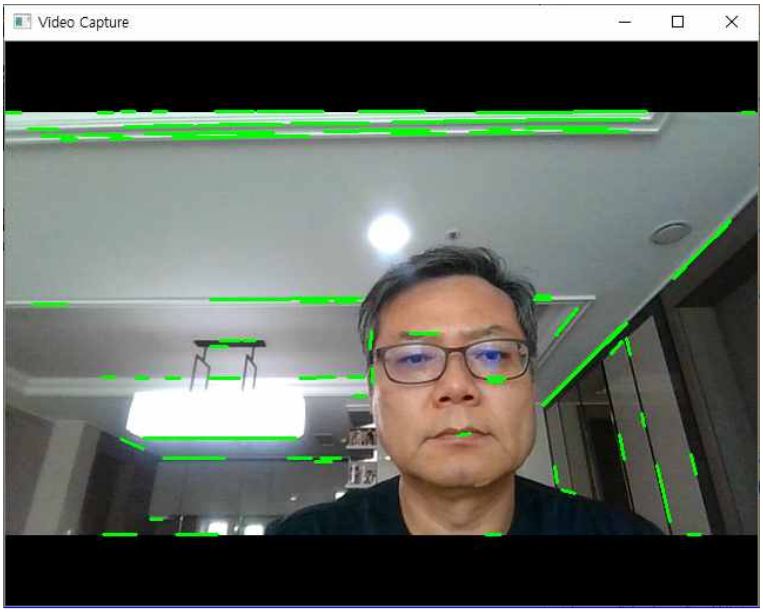
https://m.blog.naver.com/samsjang/220588392347

https://m.blog.naver.com/samsjang/220588392347

18 : 추출된 선들이 있으면
19 : 추출된 선들에 대해
20 : 각 직선에 대한 양쪽 끝점에 대해
21 : cv2 모듈의 line 함수를 호출하여 img가 가리키는 그림의 해당 위치에 선을 그립니다.

https://darkpgmr.tistory.com/137





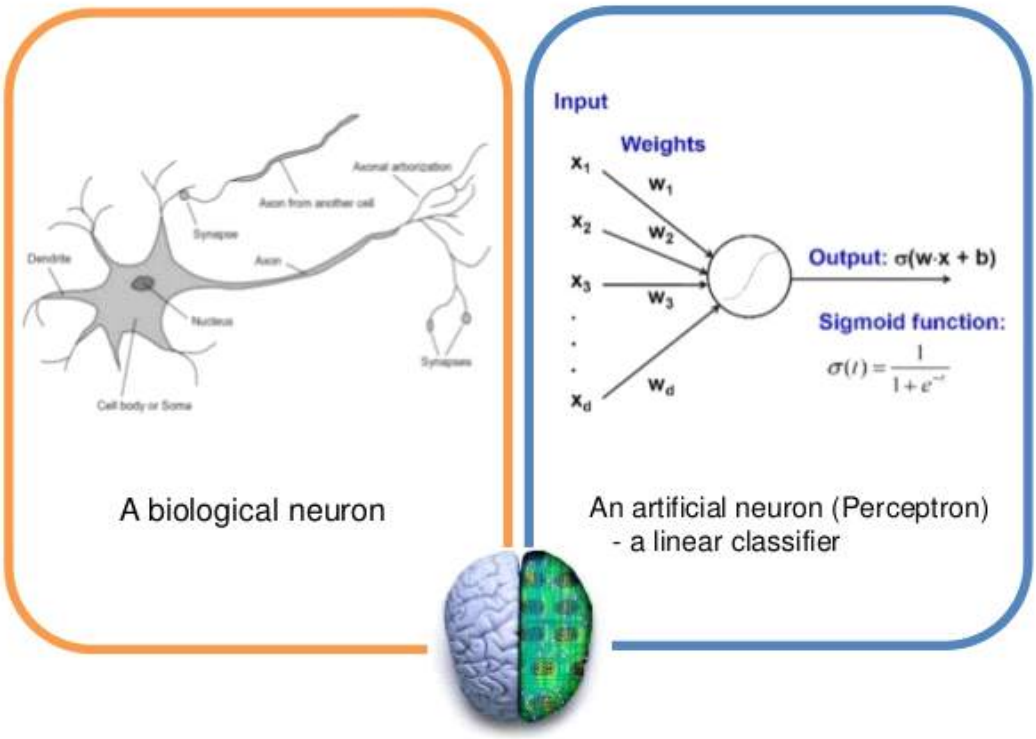
차선 인식 예제

```
1  : import cv2
2  : import numpy as np
3  : import math
4  : import RPi.GPIO as GPIO
5  :
6  : minLineLength =5
7  : maxLineGap =10
8  : theta =0
9  :
10 : GPIO.setmode(GPIO.BOARD)
11 : GPIO.setup(7, GPIO.OUT)
12 : GPIO.setup(8, GPIO.OUT)
13 :
14 : cap = cv2.VideoCapture(0)
15 :
16 : while cap.isOpened():
17 :     ret,img = cap.read()
18 :
19 :     if ret:
20 :             gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
21 :             blurred = cv2.GaussianBlur(gray, (5, 5), 0)
22 :             edged = cv2.Canny(blurred, 85, 85)
23 :             lines = cv2.HoughLinesP(edged,1,np.pi
/180,10,minLineLength,maxLineGap)
```

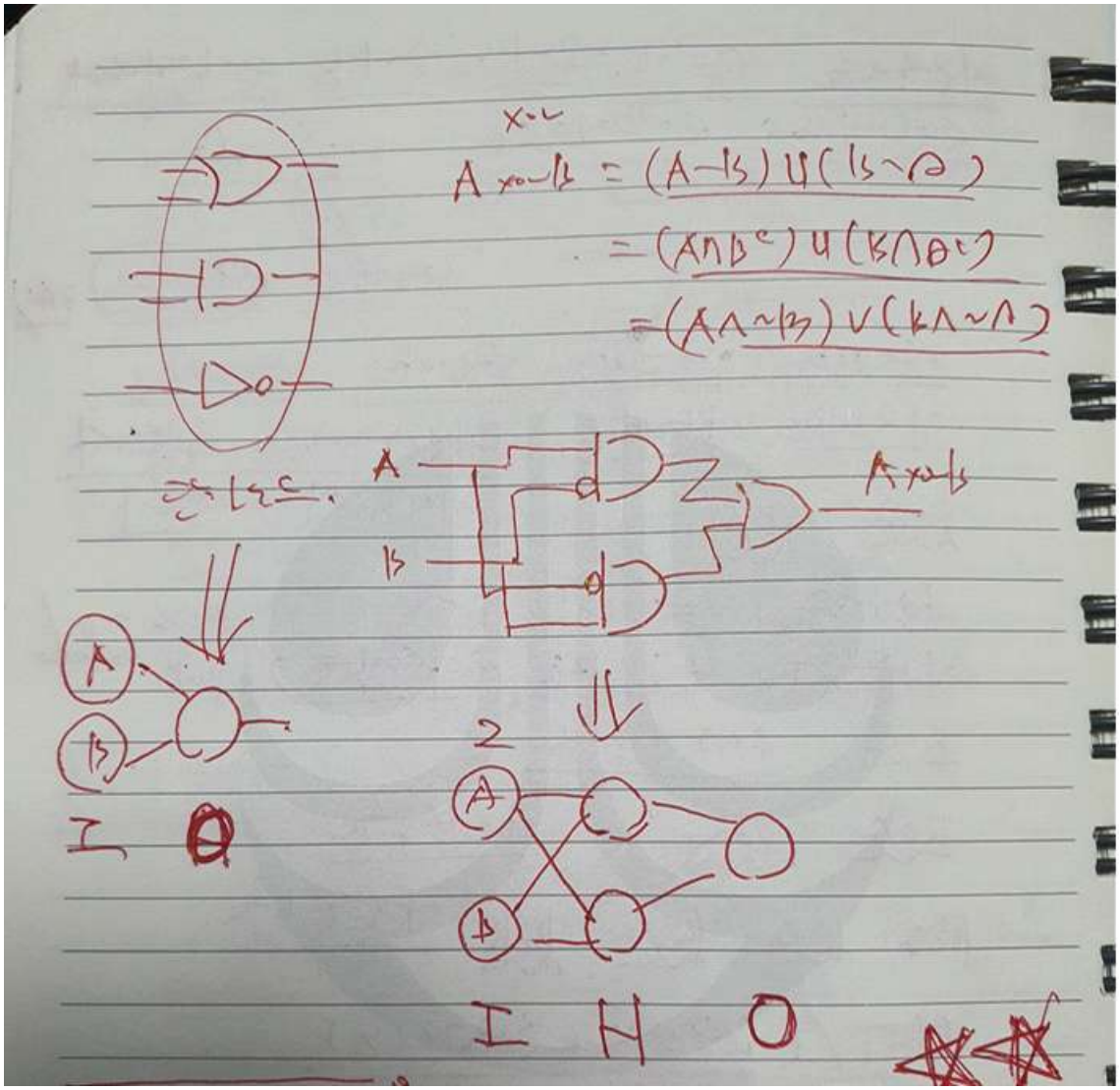
```
24 :                if len(lines):
25 :                        for x in range(0, len(lines)):
26 :                                for x1,y1,x2,y2 in lines[x]:
27 :                                        cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
28 :                                        theta =theta +math.atan2((y2 -y1),(x2
-x1))
29 :                                        #print(theta)
30 :
31 :                threshold =6
32 :                if(theta >threshold):
33 :                        GPIO.output(7,True)
34 :                        GPIO.output(8,False)
35 :                        print("left")
36 :                if(theta <-threshold):
37 :                        GPIO.output(8,True)
38 :                        GPIO.output(7,False)
39 :                        print("right")
40 :                if(abs(theta)<threshold):
41 :                        GPIO.output(8,False)
42 :                        GPIO.output(7,False)
43 :                        print("straight")
44 :
45 :                theta =0
46 :
47 :                cv2.imshow('Video Capture', img)
48 :
49 :                key = cv2.waitKey(1) &0xFF #1ms wait
50 :                if key ==27: #ESC
51 :                        break
52 :
53 : cap.release()
54 : cv2.destroyAllWindows()
55 : GPIO.cleanup()
```

https://www.hackster.io/Abhinav_Abhi/road-lane-detection-with-raspberry-pi-a4711f

Biological neuron and Perceptrons



https://sacko.tistory.com/10


A
B
A
B
I
H
O