영역 기반 영상 처리(2)
Region-based Image Processing

목 차

2차 미분을 이용한 에지 검출

❖2차 미분 마스크

- 1차 미분: 변화율을 측정
- $f'(x) = \frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = f(x+1) - f(x)$

- 2차 미분: 1차 미분의 변화율을 측정
- $f''(x) = \frac{\partial^2 f(x)}{\partial^2 x} = f'(x) - f'(x-1)$
$= f(x+1) - 2f(x) + f(x-1)$

라플라시안(Laplacian) 마스크

❖ Laplacian

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,\ y)}{\partial^2 x} + \frac{\partial^2 f(x,\ y)}{\partial^2 y}$$

$$\frac{\partial^2 f(x,y)}{\partial^2 x} = f(x+1,y) - 2f(x,y) + f(x-1,y)$$

$$\frac{\partial^2 f(x,y)}{\partial^2 y} = f(x,y+1) - 2f(x,y) + f(x,y-1)$$

1	-2	1

+

1
-2
1

=

0	1	0
1	-4	1
0	1	0

Laplacian

라플라시안(Laplacian) 마스크

0	−1	0
−1	4	−1
0	−1	0

마스크 1

0	1	0
1	−4	1
0	1	0

마스크 2

(a) 4방향

−1	−1	−1
−1	8	−1
−1	−1	−1

마스크 3

1	1	1
1	−8	1
1	1	1

마스크 4

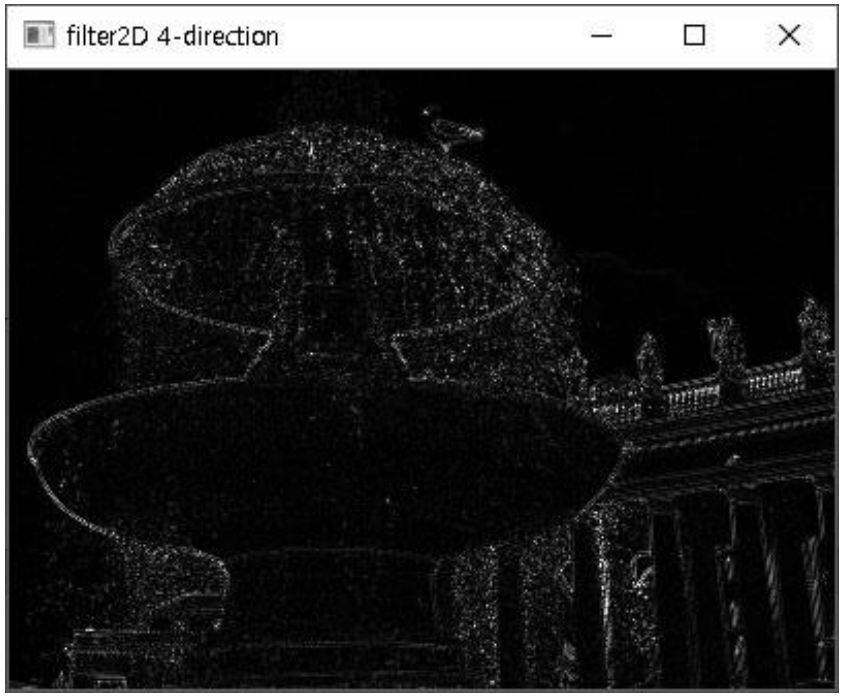
(b) 8방향

〈그림 7.2.8〉 라플라시안 마스크의 예

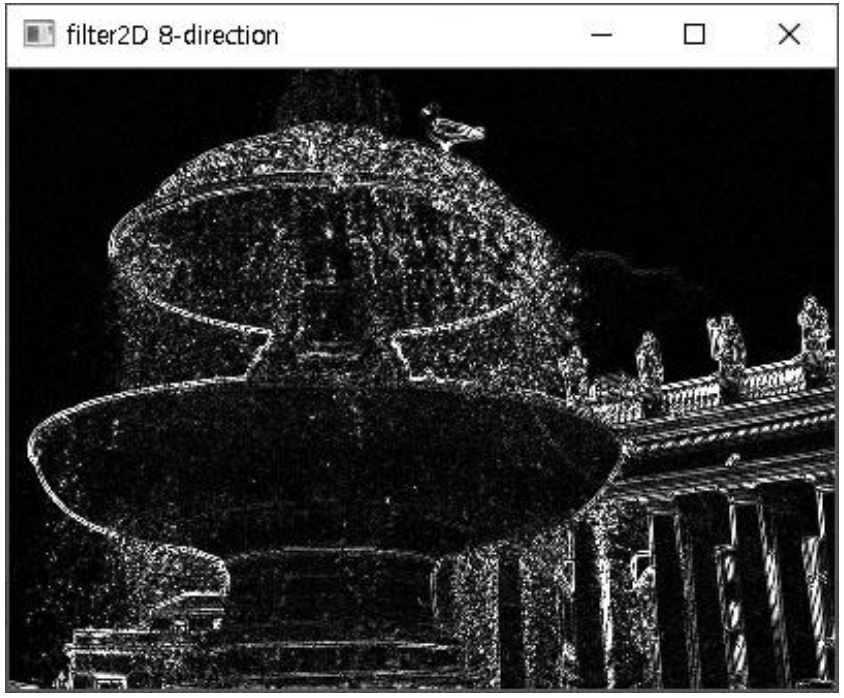
예제 7.2.6 라플라시안 에지 검출 – 06.edge_laplacian.py

```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/laplacian.jpg", cv2.IMREAD_GRAYSCALE)
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  data1 = [ [0,   1,   0],                          # 4 방향 필터
07            [1,   -4,  1],  G_x
08            [0,   1,   0]]
09  data2 = [ [-1,  -1,  -1],                         # 8 방향 필터
10            [-1,  8,   -1],  G_y
11            [-1,  -1,  -1]]
12  mask4 = np.array(data1, np.int16)                 # 음수로 인해 int16형 행렬 선언
13  mask8 = np.array(data2, np.int16)
14
15  dst1 = cv2.filter2D(image, cv2.CV_16S, mask4)     # OpenCV 회선 함수 호출
16  dst2 = cv2.filter2D(image, cv2.CV_16S, mask8)
17  dst3 = cv2.Laplacian(image, cv2.CV_16S, 1)        # OpenCV 라플라시안 수행 함수
18
19  cv2.imshow("image", image)
```


image


filter2D 4-direction


Laplacian_OpenCV


filter2D 8-direction

라플라시안(Laplacian) 마스크

❖ 2차 미분 필터링 (Laplacian filtering)

```
Laplacian (InputArray src,  OutputArray dst,
        int   ddepth,
        int   ksize[1]=1,     // 커널의 크기
        double scale[2]=1, double delta[3]=0,
        int borderType=BORDER_DEFAULT
)
```

1) ksize=1 or 3 일 때, 모두 3x3 커널이 생성됨
2) scale: dst 영상에 저장하기 전에 곱해지는 값
3) delta: dst 영상에 저장하기 전에 더해지는 값

- 에지 검출
  - 라플라시안 필터링 후 0교차의 위치가 에지 임

LoG (Laplacian of Gaussian)

- 영상에 2차 미분을 바로 적용하면 잡영(noise)에 민감함
  → Gaussian 스무딩 후에 2차 미분 수행
  결합법칙 성립: $\mathrm{L} * (\mathrm{G} * \mathrm{I}) = (\mathrm{L} * \mathrm{G}) * \mathrm{I}$

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- 커널의 크기 vs. 분산
  - 커널의 크기 $n = 2(3\sigma) + 1$
  - 분산 $\sigma = 0.3\left(\frac{n}{2} - 1\right) + 0.8$
- 에지 검출
  - LoG 필터링 후 0교차의 위치가 에지 임

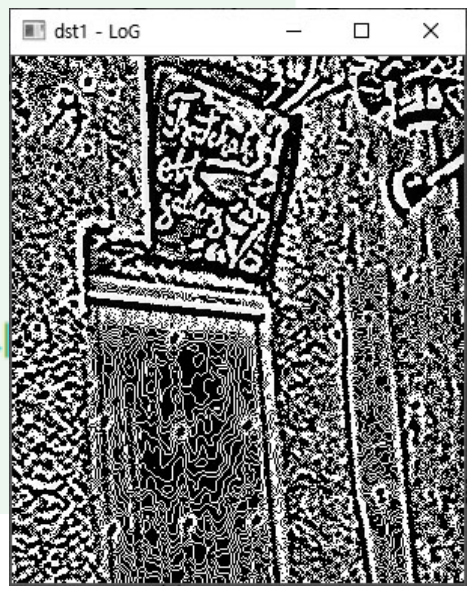
DoG (Difference of Gaussian)

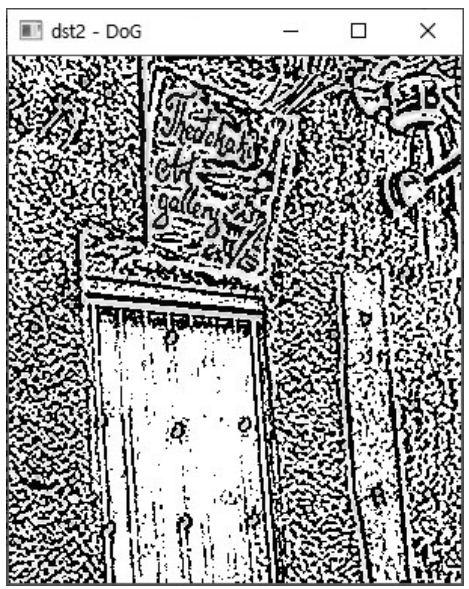
- 가우시안 필터링의 차이를 이용하여 에지를 추출

$$DoG(x,y) = \left(\frac{1}{2\pi\sigma_1^2}\exp\left(-\frac{x^2+y^2}{2\sigma_1^2}\right)\right) - \left(\frac{1}{2\pi\sigma_2^2}\exp(-\frac{x^2+y^2}{2\sigma_2^2})\right)$$

예제 7.2.7 LoG/DoG 에지 검출 -- 07.edge_DOG.py

```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/dog.jpg", cv2.IMREAD_GRAYSCALE)
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  gaus = cv2.GaussianBlur(image, (7, 7), 0, 0)          # 가우시안 마스크 적용
07  dst1 = cv2.Laplacian(gaus, cv2.CV_16S, 7)             # 라플라시안 수행
08
09  gaus1 = cv2.GaussianBlur(image, (3, 3), 0)            # 가우시안 블러링
10  gaus2 = cv2.GaussianBlur(image, (9, 9), 0)
11  dst2 = gaus1 - gaus2                                  # DoG 수행
12
13  cv2.imshow("image", image)
14  cv2.imshow("dst1- LoG", dst1.astype('uint8'))         # 형변환 후 영상 표시
15  cv2.imshow("dst2- DoG", dst2)
16  cv2.waitKey(0)
```





캐니(Canny) 에지 검출

❖ Canny edge detection algorithm

1. 블러링을 통한 노이즈 제거 (가우시안 블러링)
2. 화소 기울기(gradiant)의 강도와 방향 검출 (소벨 마스크)
3. 비최대치 억제(non-maximum suppression) → 1 pixel-width edge
4. 이력 임계값(hysteresis threshold)으로 에지 결정

각 화소가 4.1 높은 임계값보다 크면 에지
4.2 낮은 임계값보다 작으면 에지 아님.
4.3 두 임계값 사이에 있으면 에지 추적

캐니(Canny) 에지 검출

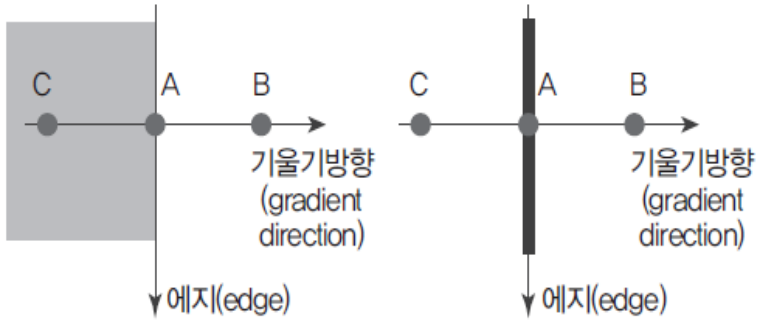
1) 블러링: 5×5 크기의 가우시안 필터 적용
   - 불필요한 잡음 제거 및 필터 크기는 변경가능

2) 화소 기울기(gradient) 검출
   - 가로 방향과 세로 방향의 소벨 마스크로 회선 적용
   - 회선 완료된 행렬로 화소 기울기의 크기(magnitude)와 방향(direction) 계산
   - 기울기 방향은 4개 방향(0, 45, 90, 135)으로 근사하여 단순화

캐니(Canny) 에지 검출

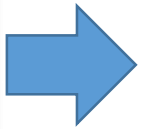
3) 비최대치 억제(non-maximum suppression)

- 기울기의 방향과 에지의 방향은 수직



- 기울기 방향에 따른 이웃 화소 선택

〈그림 7.2.8〉 비최대치 억제를 위한 이웃 화소 선택



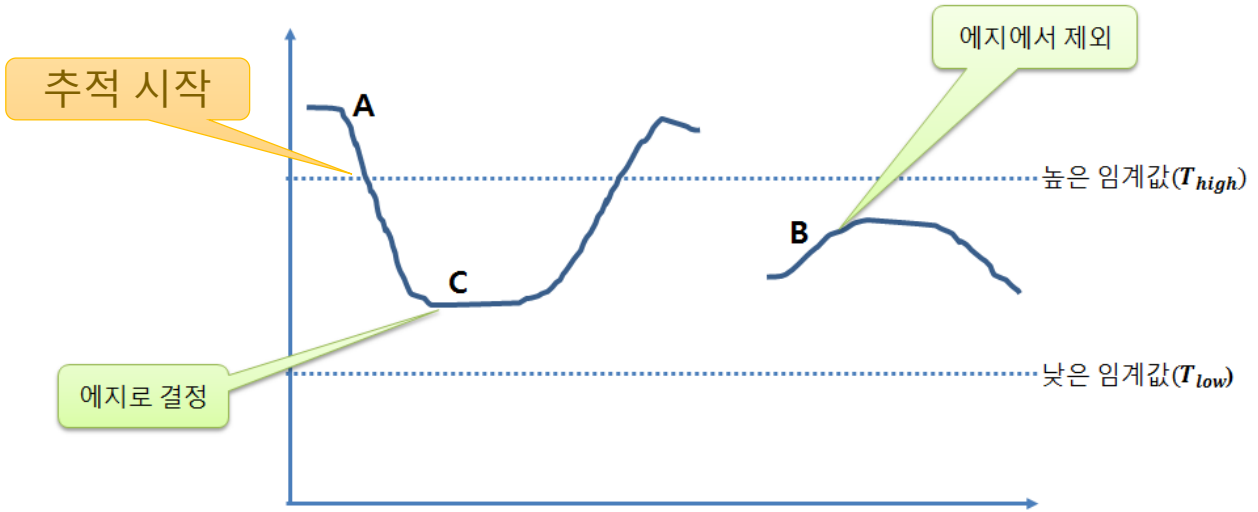
1 pixel-width edge

캐니(Canny) 에지 검출

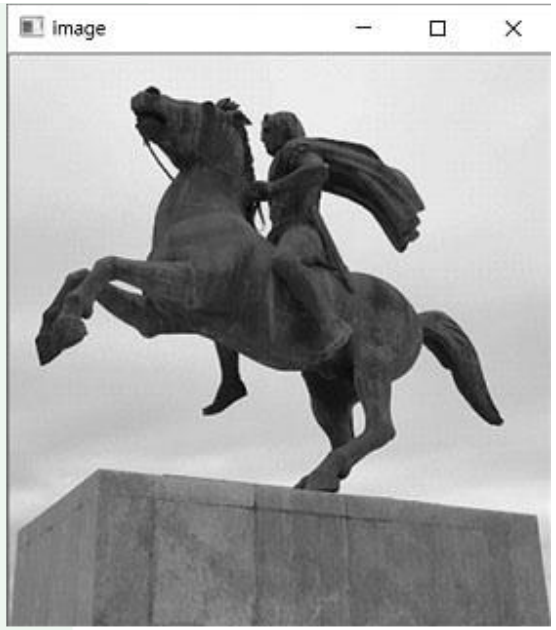
4) 이력 임계값 방법 (hysteresis thresholding)

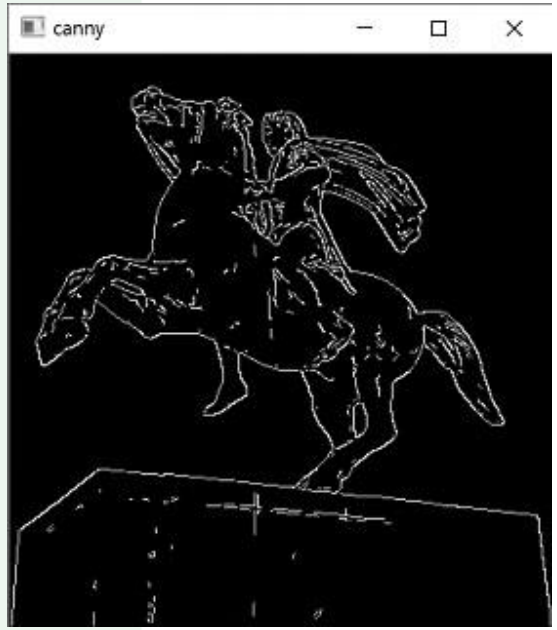
- 두 개의 임계값을 사용해 에지 이력 추적으로 에지 결정
- 각 화소에서 높은 임계값보다 크면 에지 추적 시작

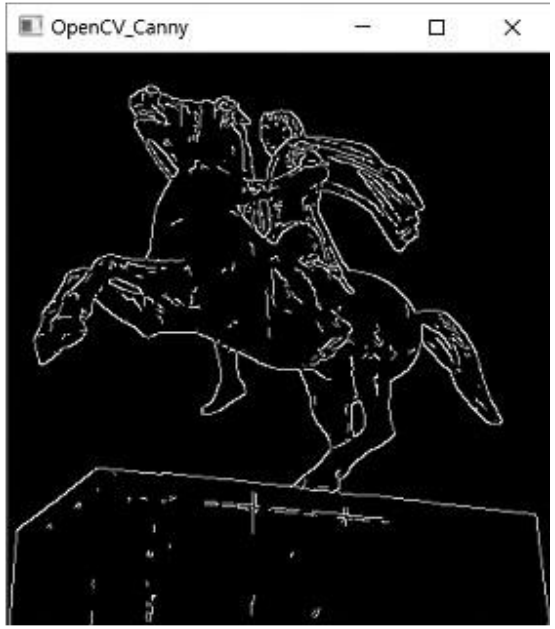
  → 낮은 임계값보다 큰 화소 중 높은 임계값 보다 큰 에지와 연결되어 있다면 에지로 결정



```python
image = cv2.imread("images/canny.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 오류")

pos_ck = np.zeros(image.shape[:2], np.uint8)          # 추적 완료 점검 행렬
canny = np.zeros(image.shape[:2], np.uint8)           # 캐니 에지 행렬

## 캐니 에지 검출
gaus_img = cv2.GaussianBlur(image, (5, 5), 0.3)
Gx = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 1, 0, 3)      # x방향 마스크
Gy = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 0, 1, 3)      # y방향 마스크
sobel = cv2.magnitude(Gx, Gy)                                  # 두 행렬 벡터 크기

directs = cv2.phase(Gx, Gy) / (np.pi/4)               # 에지 기울기 계산 및 근사
directs = directs.astype(int)) % 4                    # 8방향 → 4방향 축소
max_sobel = nonmax_suppression(sobel, directs)        # 비최대치 억제
hysteresis_th(max_sobel, 100, 150)                    # 이력 임계값

canny2 = cv2.Canny(image, 100, 150)                   # OpenCV 캐니 에지 검출

cv2.imshow("image", image)
cv2.imshow("canny", canny)                            # 사용자 정의 캐니
cv2.imshow("OpenCV_Canny", canny2)                    # OpenCV 캐니 에지
cv2.waitKey(0)
```







캐니(Canny) 에지 검출

```
void   Canny (
          InputArray     image,     // 8bits 명도 영상
          OutputArray   edges,
          double threshold1,   double threshold2,
          int   apertureSize=3,              // Sobel 필터의 크기
          bool L2gradient[1)]=false
)
```

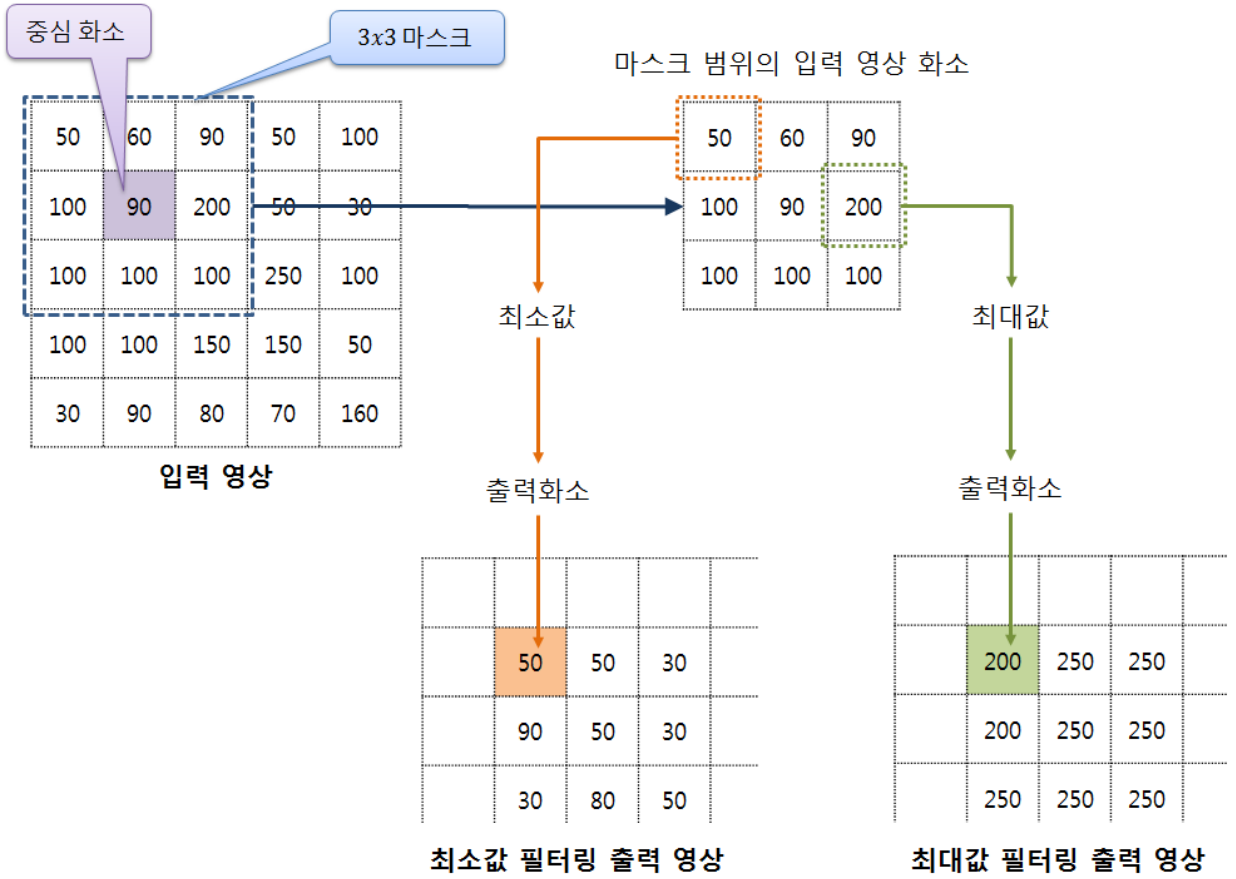
1) $Gradient\ magnitude = \begin{cases} \sqrt{\left(\frac{dI}{dx}\right)^2 + \left(\frac{dI}{dy}\right)^2}, if\ L2gradient = true \\ \left|\frac{dI}{dx}\right| + \left|\frac{dI}{dy}\right|, if\ L2gradient = false \end{cases}$

3. 기타 필터링

- ❖최대값/최소값 필터링
- ❖평균값 필터링
- ❖미디언 필터링
- ❖가우시안 필터링

최대값/최소값 필터링

- 입력 영상의 해당 화소(중심화소)에서 마스크로 씌워진 영역의 입력화소들을 가져와서 그 중에 최대값/최소값을 출력 화소로 결정하는 방법



최대값/최소값 필터링

❖최대값 필터링
- 가장 큰 값인 밝은 색들로 출력화소를 구성
- 돌출되는 어두운 값이 제거 전체적으로 밝은 영상이 됨

❖최소값 필터링
- 가장 작은 값들인 어두운 색들로 출력화소를 구성
- 돌출되는 밝은 값들이 제거되며, 전체적으로 어두운 영상 됨

예제 7.3.1 최솟값-최댓값 필터링 – 09.filter_minMax.py

0이면 최소값 필터링 수행,
1이면 최대값 필터링 수행

```
01  import numpy as np, cv2
02
03  def minmax_filter(image, ksize, mode):                  # 최솟값&최댓값 필터링 함수
04      rows, cols = image.shape[:2]
05      dst = np.zeros((rows, cols), np.uint8)
06      center = ksize // 2                                 # 마스크 절반 크기
07
08      for i in range(center, rows - center):              # 입력 영상 순회
09          for j in range(center, cols - center):
10              ## 마스크 영역 행렬 처리 방식
11              y1, y2 = i - center, i + center + 1         # 마스크 높이 범위
12              x1, x2 = j - center, j + center + 1         # 마스크 너비 범위
13              mask = image[y1:y2, x1:x2]                  # 마스크 영역
14              dst[i, j] = cv2.minMaxLoc(mask)[mode]       # 최소 or 최대
15      return dst
16
17  image = cv2.imread("images/min_max.jpg", cv2.IMREAD_GRAYSCALE)
18  if image is None: raise Exception("영상파일 읽기 오류")
19
20  minfilter_img = minmax_filter(image, 3, 0)              # 3×3 마스크 최솟값 필터링
21  maxfilter_img = minmax_filter(image, 3, 1)              # 3×3 마스크 최댓값 필터링
22
23  cv2.imshow("image", image)
24  cv2.imshow("minfilter_img", minfilter_img)
25  cv2.imshow("maxfilter_img", maxfilter_img)
26  cv2.waitKey(0)
```

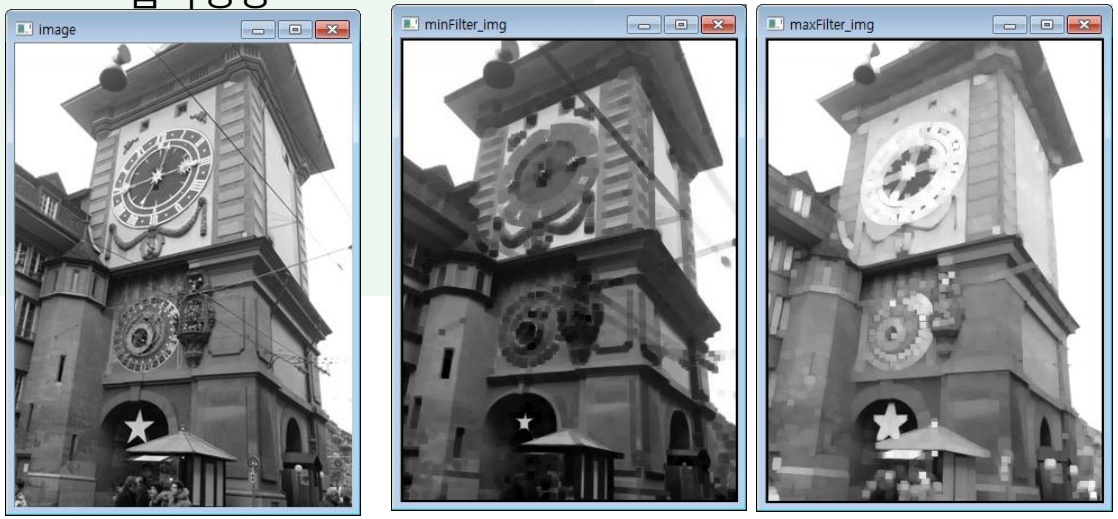
[예제 7.3.1] 최소&최대값 필터링

```
25  int main()
26  {
27      Mat image = imread("../image/min_max.jpg", 0);
28      CV_Assert(image.data);
29
30      Mat min_img, max_img;
31      minMaxFilter(image, min_img, 5, 0);      // 5x5 마스크 최솟값 필터링
32      minMaxFilter(image, max_img, 5, 1);      // 5x5 마스크 최댓값 필터링
33
34      imshow("image", image);
35      imshow("minFilter_img", min_img);
36      imshow("maxFilter_img", max_img);
37      waitKey();
38      return 0;
39  }
```

0: 최솟값 필터링

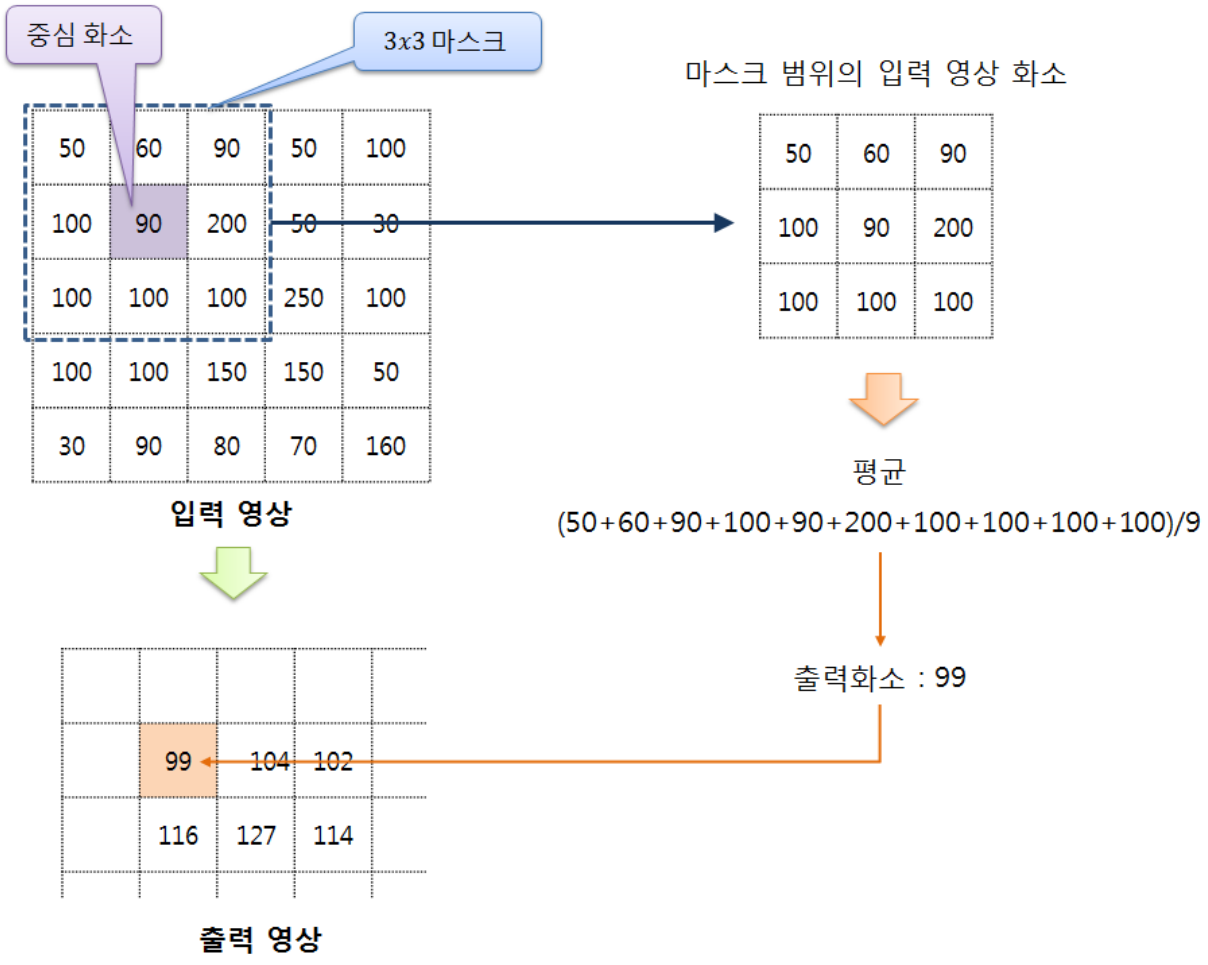
1: 최댓값 필터링

입력영상



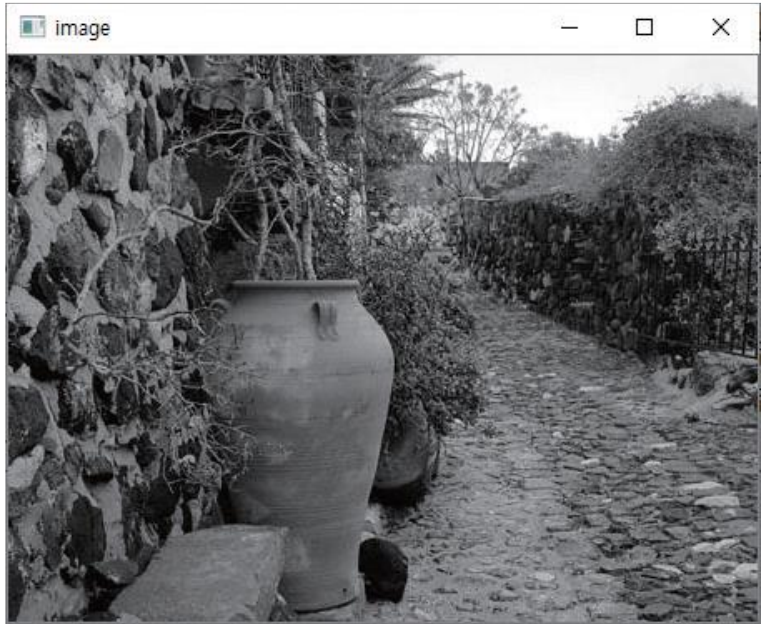
평균값 필터링

- 마스크 영역 입력화소들의 평균을 구하여 출력화소로 지정하는 방법

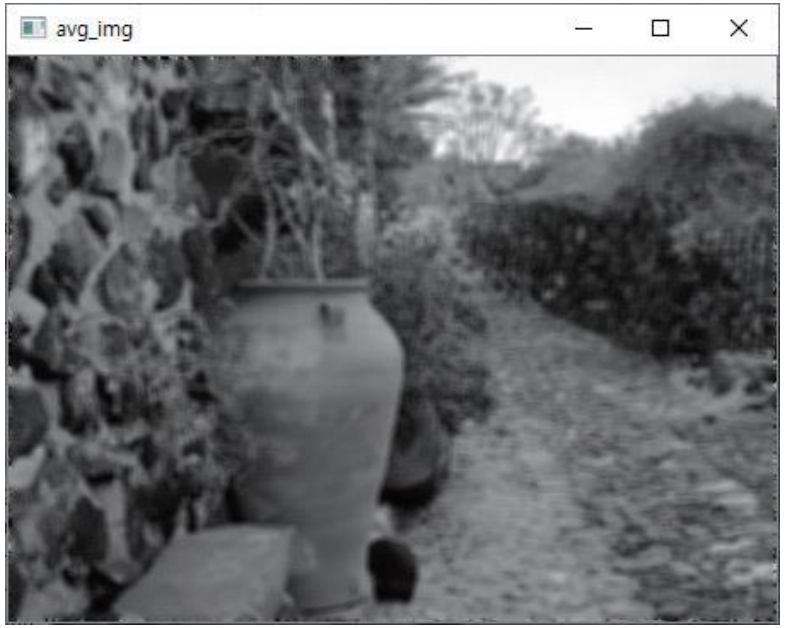


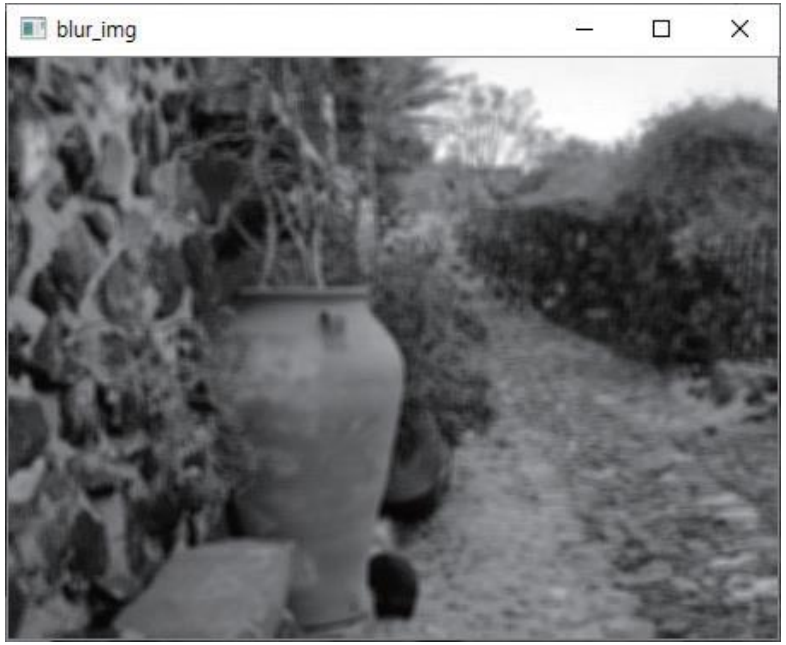
예제 7.3.2 평균값 필터링 – 10.filter_average.py

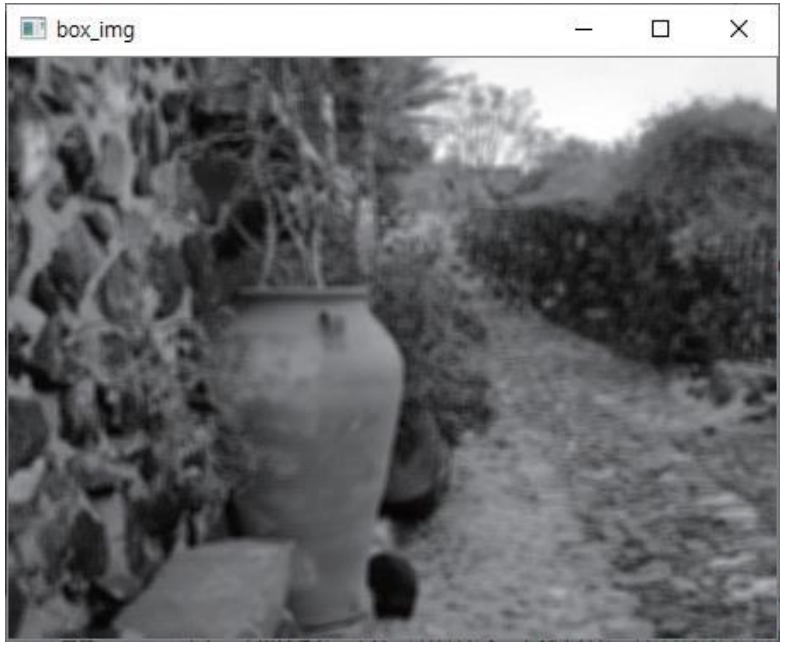
```python
import numpy as np, cv2

def average_filter(image, ksize):                       # 평균값 필터링 함수
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2                                 # 마스크 절반 크기

    for i in range(rows):                               # 입력 영상 순회
        for j in range(cols):
            y1, y2 = i - center, i + center + 1         # 마스크 높이 범위
            x1, x2 = j - center, j + center + 1         # 마스크 너비 범위
            if y1 < 0 or y2 > rows or x1 < 0 or x2 > col
                dst[i, j] = image[i, j]                 # cv2.B
            else:
                mask = image[y1:y2, x1:x2]              # 범위
                dst[i, j] = cv2.mean(mask)[0]           # np.m
    return dst

image = cv2.imread("images/avg_filter.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 오류")

avg_img  = average_filter(image, 5)                             # 사용자 정의 평균값 필터 함수
blur_img = cv2.blur(image, (5, 5), (-1,-1), cv2.BORDER_REFLECT)  # OpenCV의 블러링
box_img  = cv2.boxFilter(image, ddepth=-1, ksize=(5, 5) )       # OpenCV의 박스 필터 함수

cv2.imshow("image", image)
cv2.imshow("avg_img", avg_img)
cv2.imshow("blur_img", box_img)
cv2.imshow("box_img", box_img)
cv2.waitKey(0)
```



image


avg_img


blur_img


box_img

cv2.boxFilter & cv2.blur

```
boxFilter (InputArray src,  OutputArray dst,  int ddepth,
        Size ksize,    // 커널의 크기
        Point anchor=Point(-1, -1),  bool normalize=true,
        int borderType[1]=BORDER_DEFAULT[1]
)
```

1) borderType BORDER_DEFAULT = BORDER_REFLECT_101 **borderType → p.324 [표 7.3.1]**

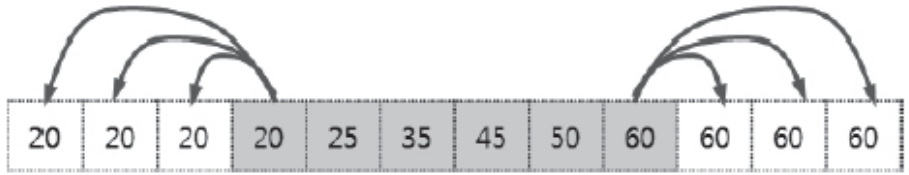
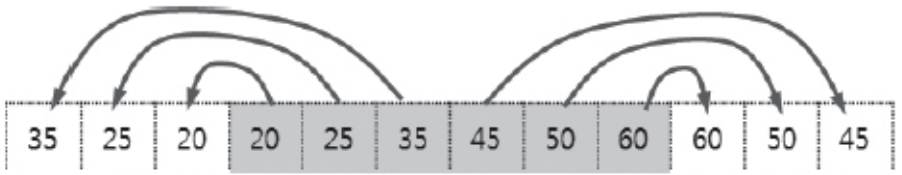
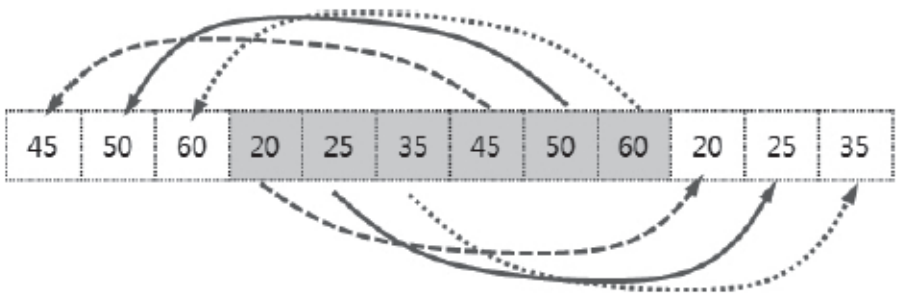
BORDER_REFLECT ex) c b a | a b c d e | e d c

BORDER_REFLECT_101 ex) d c b | a b c d e | d c b

```
blur (InputArray src,  OutputArray dst, Size ksize,
        Point anchor=Point(-1, -1), int borderType=BORDER_DEFAULT
)
```

→ 정규화된 박스 필터를 사용

〈표 7.3.1〉 경계타입 결정 옵션상수

옵션 상수	값	설명
BORDER_CONSTANT	0	특정 상수값으로 대체 
BORDER_REPLICATE	1	계산 가능한 경계의 출력화소 하나만으로 대체 
BORDER_REFLECT	2	계산 가능한 경계의 출력화소로부터 대칭되게 한 화소씩 지정 
BORDER_WRAP	3	영상의 왼쪽 끝과 오른쪽 끝이 연결되어 있다고 가정하여 한 화소씩 가져와서 지정 

cv2.copyMakeBorder

❖영상 경계 채우기(boundary padding)

```
copyMakeBorder (InputArray src, OutputArray dst,
        int top, int bottom, int left, int right,      // padding 폭
        int borderType[1],  const Scalar& value=Scalar()
)
```

1) borderType BORDER_DEFAULT = BORDER_REFLECT_101

[예제 7.3.2평균값 필터링

```
27  int main()
28  {
29      Mat image = imread("../image/min_max.jpg", 0);          // 명암도 영상 로드
30      CV_Assert(image.data);                                  // 영상 파일 예외처리
31
32      Mat avg_img, blur_img, box_img;
33      averageFilter(image, avg_img, 5);                       // 사용자 정의 평균필터 함수
34      blur(image, blur_img, Size(5, 5));                      // OpenCV 제공 블러링 함수
35      boxFilter(image, box_img, -1, Size(5, 5));              // OpenCV 제공 박스 필터 함수
36
37      imshow("image", image), imshow("avg_Filter_img", avg_img);
38      imshow("blur_img", box_img), imshow("box_img", box_img);
39      waitKey();
40      return 0;
41  }
```

입력영상

미디언 (Median) 필터링 cv2.medianBlur

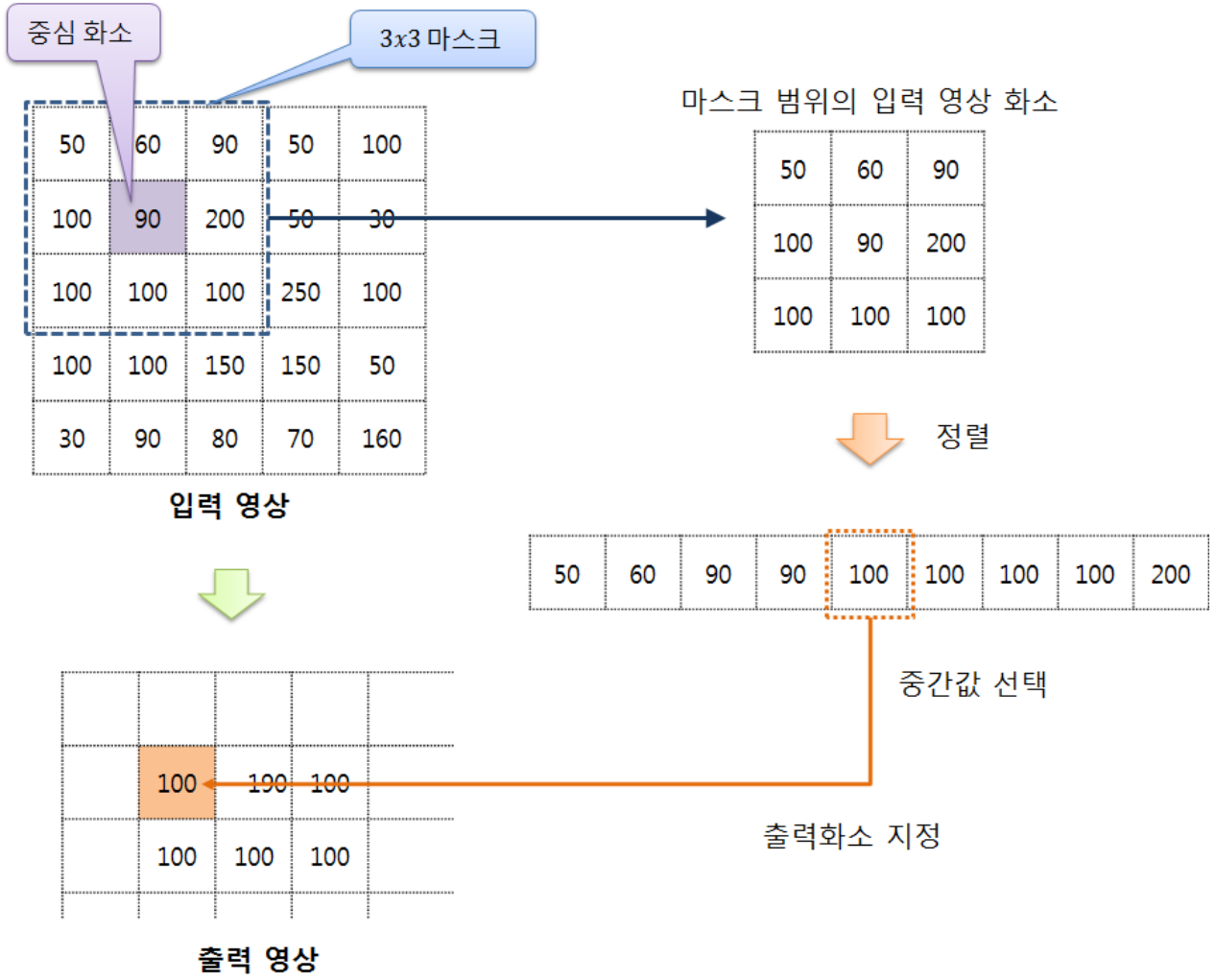
❖미디언 필터링: Median Filter

- 임펄스 잡음(impulse noise), 소금-후추 잡음(salt & pepper noise) 제거에 효과적
- 평균 필터에 비하여 블러링 현상이 적다.

```
cv2.medianBlur (InputArray src,  OutputArray dst, Size ksize)
```

→ 미디언(중위수)로 대체

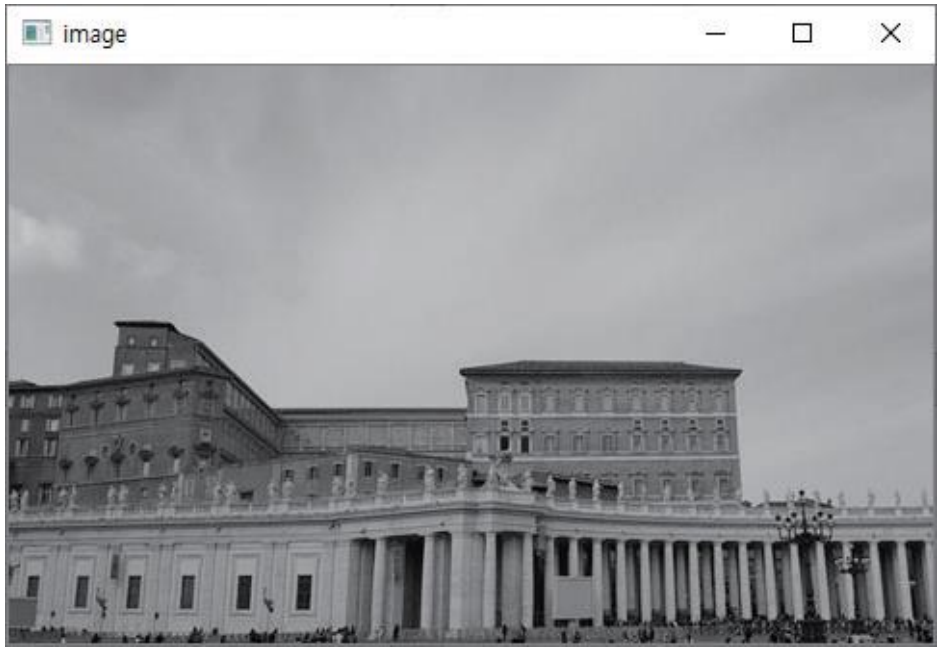
미디언 (Median) 필터링



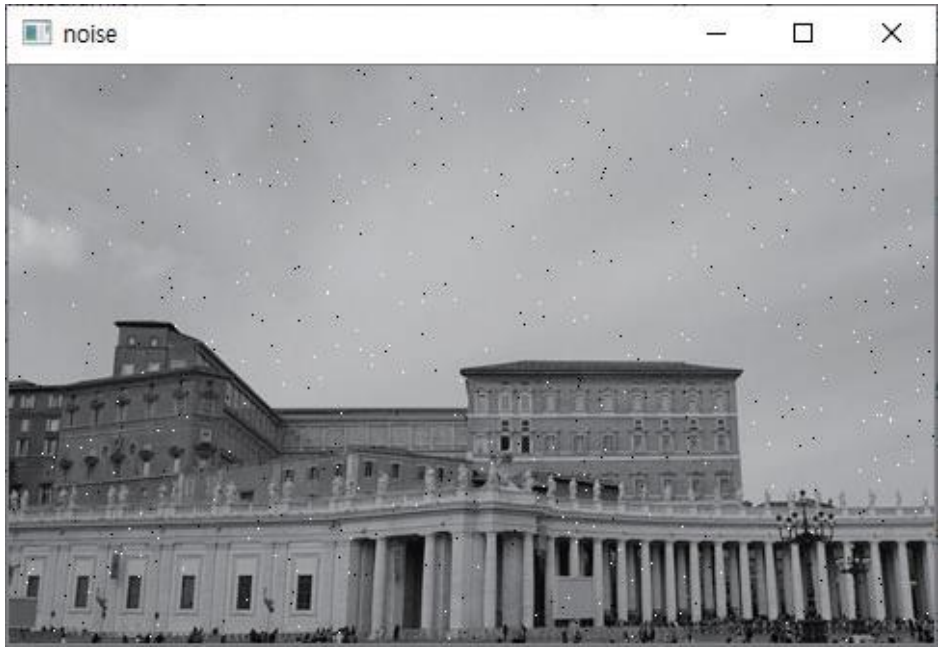
예제 7.3.3 미디언 필터링 – 11.filter_median.py

```python
import numpy as np, cv2

def median_filter(image, size):                          # 미디
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2                                  # 마스
    for i in range(center, rows - center):               # 입력
        for j in range(center, cols - center):
            y1, y2 = i - center, i + center + 1          # 마스
            x1, x2 = j - center, j + center + 1          # 마스
            mask = image[y1:y2, x1:x2].flatten()         # 관심
            sort_mask = cv2.sort(mask, cv2.SORT_EVERY_COLUMN)        # 정렬 수행
            dst[i, j] = sort_mask[sort_mask.size//2] # 출력 화소로 지정
    return dst

def salt_pepper_noise(img, n):                           # 소금 후추 잡음 생성 함수
    h, w = img.shape[:2]
    x, y = np.random.randint(0, w, n), np.random.randint(0, h, n)
    noise = img.copy()
    for (x, y) in zip(x, y):
        noise[y, x] = 0 if np.random.rand() < 0.5 else 255
    return noise
```

```python
image = cv2.imread("images/median.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 오류")

noise = salt_pepper_noise(image, 500)                    # 소금-후추 잡음 영상 생성
med_img1 = median_filter(noise, 5)                       # 사용자 정의 함수
med_img2 = cv2.medianBlur(noise, 5)                      # OpenCV 제공 함수

cv2.imshow("image", image),
cv2.imshow("noise", noise),
cv2.imshow("median - User", med_img1)
cv2.imshow("median - OpenCV", med_img2)
cv2.waitKey(0)
```



image


noise


median - User

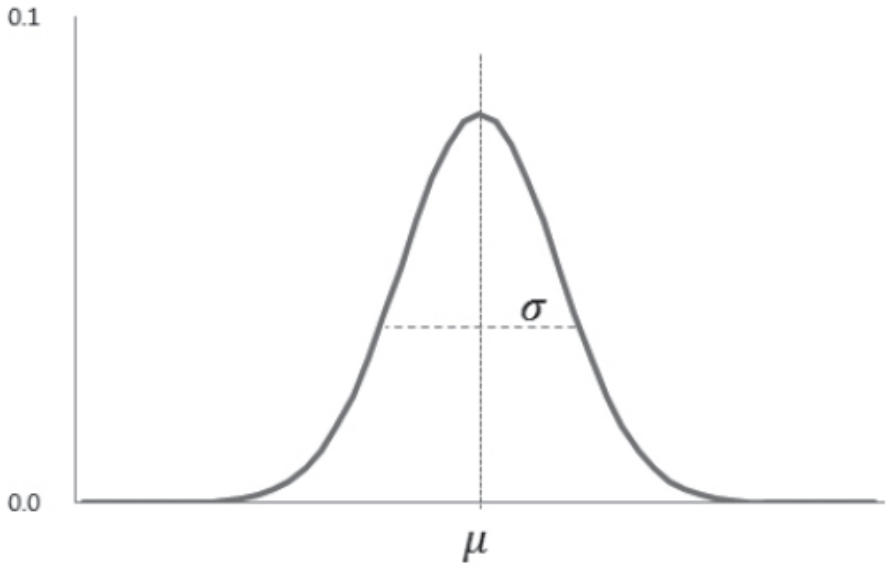

median - OpenCV

가우시안 (Gaussian) 필터링

❖가우시안 분포(정규 분포)

- 특정 값의 출현 비율을 그래프로 그렸을 때, 평균에서 가장 큰 수치 가짐
- 평균을 기준으로 좌우 대칭 형태
- 양끝으로 갈수록 수치가 낮아지는 종 모양
  - 평균과 표준 편차로 그래프 모양 변경

$$N(\mu,\sigma)(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

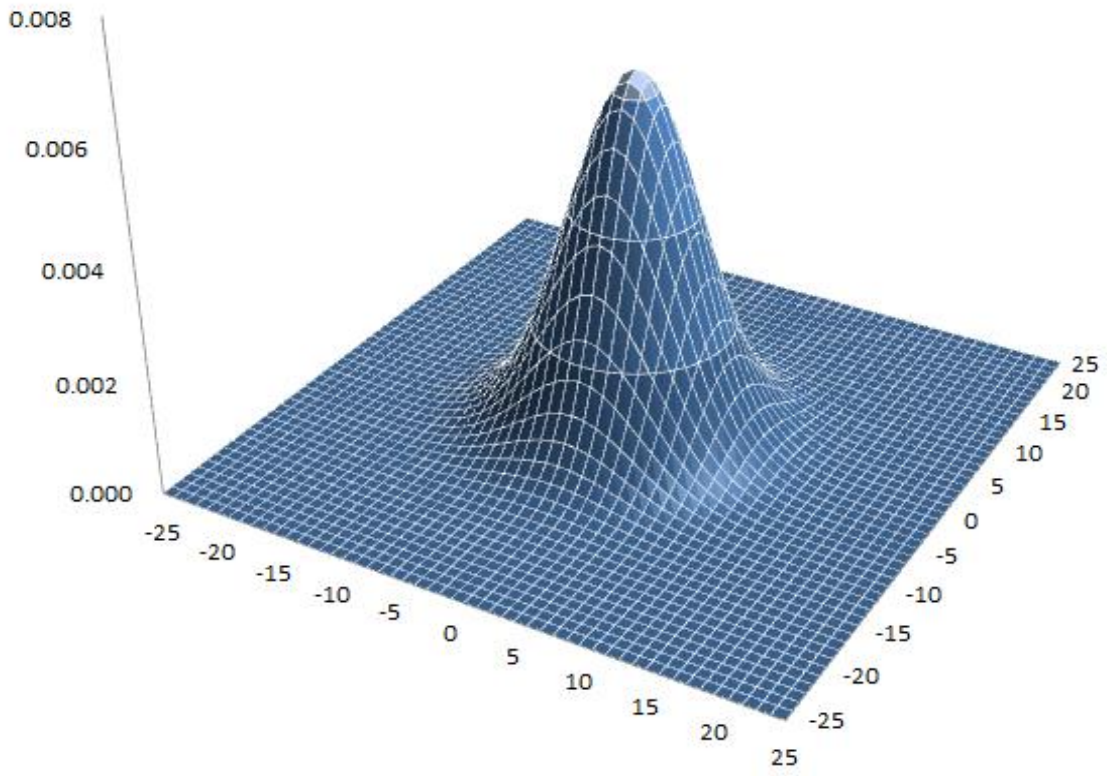


〈그림 7.3.4〉 정규 분포 그래프

가우시안 (Gaussian) 필터링

❖2차원 가우시안 분포

$$N(\mu, \sigma_x, \sigma_y)(x, y) = \frac{1}{\sigma_x \sigma_y 2\pi} exp\left[-\left(-\frac{(x-\mu)^2}{2\sigma_x^2} + \frac{(y-\mu)^2}{2\sigma_y^2}\right)\right]$$



cv2.GaussianBlur

```
GaussainBlur (InputArray src,  OutputArray dst,
        Size  ksize,
        double  sigmaX,          # x축 방향의 가우시안 커널 표준편차
        double  sigmaY = 0[1),   # y축 방향의 가우시안 커널 표준편차
        int borderType=BORDER_DEFAULT
)
```

1) sigmaX≠0, sigmaY=0 이면, sigmaY = sigmaX
   sigmaX=0, sigmaY=0이면, sigmaX=0.3(ksize.width-1)/2-1) +0.8
   sigmaY=0.3(ksize.heogjt-1)/2-1) +0.8

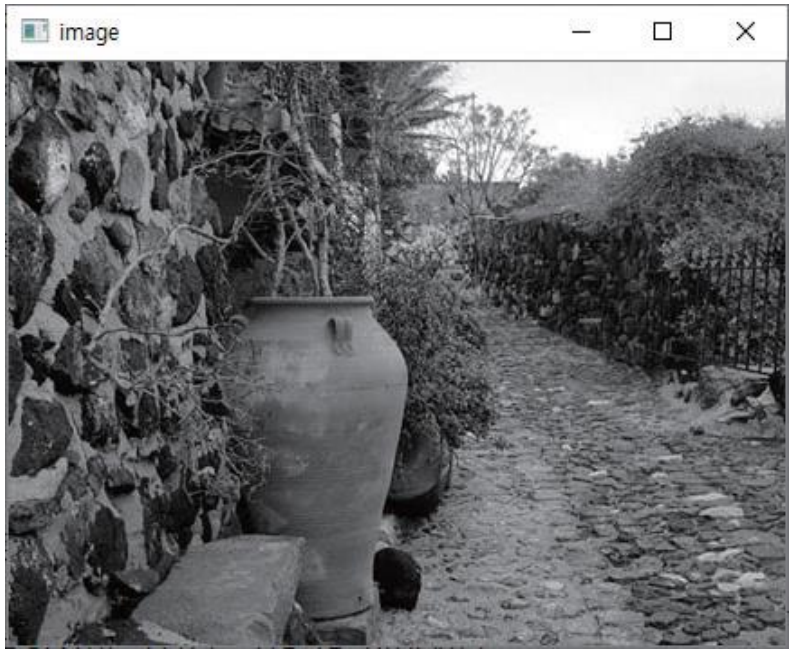
getGaussianKernel() : 1차원 가우시안 계수 생성

getGaussianMask() : 2차원 가우시안 계수 생성

예제 7.3.4 가우시안 필터링 – 12.filter_gausian.py

```python
import numpy as np, cv2

def getGaussianMask(ksize, sigmaX, sigmaY):          # 가우사안 마스크 생성 함수
    sigma = 0.3 * ((np.array(ksize) - 1.0) * 0.5 - 1.0) + 0.8
    if sigmaX <= 0: sigmaX = sigma[0]
    if sigmaY <= 0: sigmaY = sigma[1]

    u = np.array(ksize)//2
    x = np.arange(-u[0], u[0]+1, 1)
    y = np.arange(-u[1], u[1]+1, 1)
    x, y = np.meshgrid(x, y)

    ratio = 1 / (sigmaX * sigmaX * 2 * np.pi)
    v1 = x ** 2 / (2 * sigmaX ** 2)
    v2 = y ** 2 / (2 * sigmaY ** 2)
    mask = ratio * np.exp(-(v1+v2))
    return mask / np.sum(mask)
```

```python

image = cv2.imread("images/smoothing.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 오류")

ksize = (17, 5)                                          # 커널 크기: 가로×세로
gaussian_2d = getGaussianMask(ksize, 0, 0)
gaussian_1dX = cv2.getGaussianKernel(ksize[0], 0, cv2.CV_32F)    # 가로 방향 마스크
gaussian_1dY = cv2.getGaussianKernel(ksize[1], 0, cv2.CV_32F)    # 세로 방향 마스크

gauss_img1 = cv2.filter2D(image, -1, gaussian_2d)        # 사용자 생성 마스크 적용
gauss_img2 = cv2.GaussianBlur(image, size, 0)            # OepnCV 제공1–가우시안 블러링
gauss_img3 = cv2.sepFilter2D(image, -1, gaussian_1dX, gaussian_1dY)    # OpenCV 제공2

titles = ['image', 'gauss_img1', 'gauss_img2', 'gauss_img3']
for t in titles: cv2.imshow(t, eval(t))          # 문자열 리스트로 행렬들을 윈도우 표시
cv2.waitKey(0)
```



image


gauss_img1

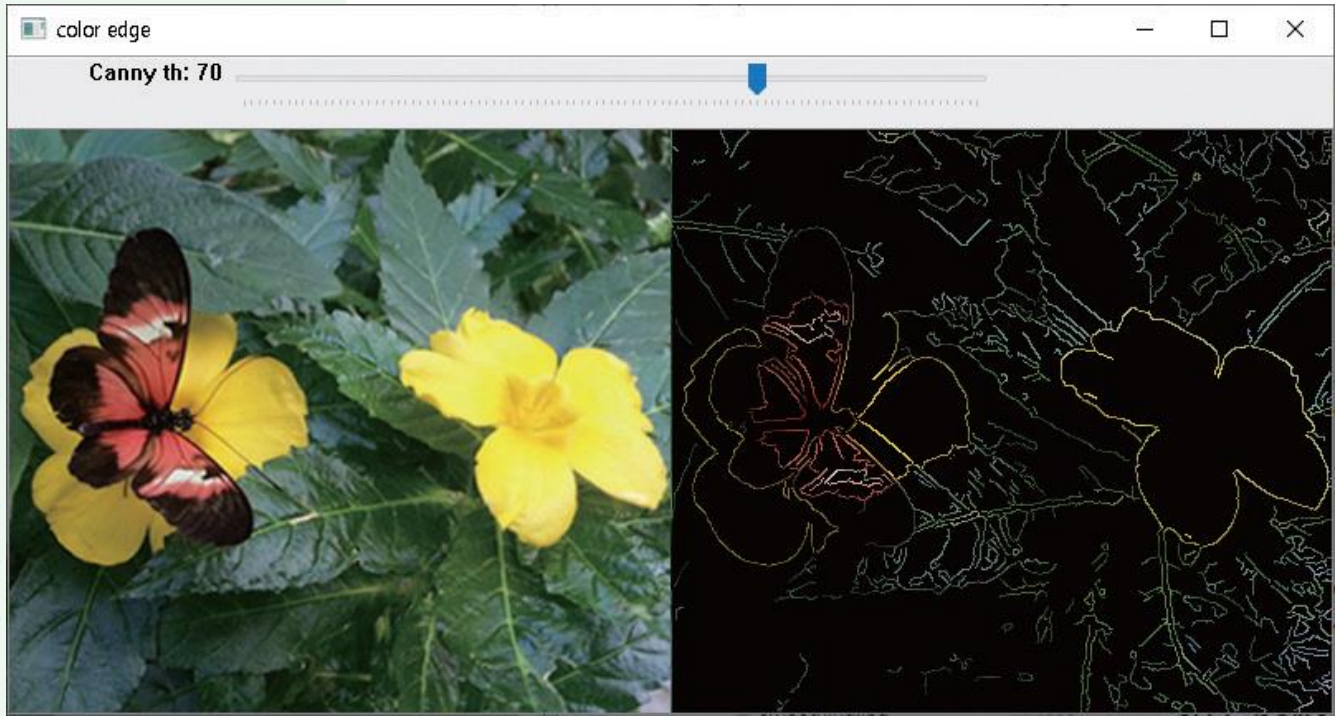

gauss_img2


gauss_img3

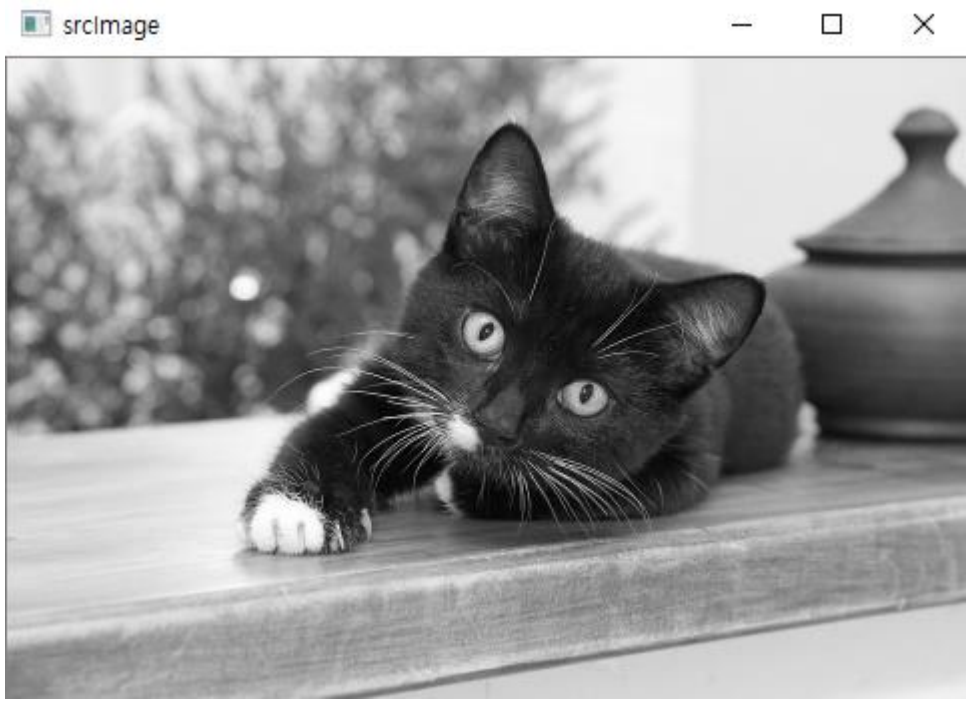
컬러 에지를 검출해 보자.

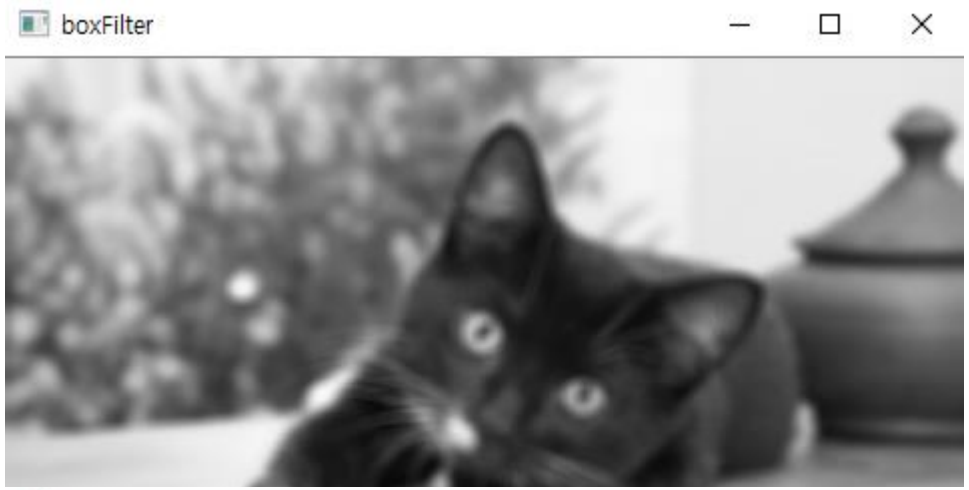
심화예제 7.3.5 블러링과 캐니 에지를 이용한 컬러 에지 검출 – 13.edge_color_canny.py

```
01  import cv2
02
03  def onTrackbar(th):                                          # 트랙바 콜백 함수
04      rep_edge = cv2.GaussianBlur(rep_gray, (5, 5), 0)         # 가우시안 블러링
05      rep_edge = cv2.Canny(rep_edge, th, th*2, 5)              # 캐니 에지 검출
06      h, w = image.shape[:2]
07      cv2.rectangle(rep_edge, (0, 0, w, h), 255, -1)           # 흰색 사각형 그리기
08      color_edge = cv2.bitwise_and(rep_image, rep_image, mask=rep_edge)
09      cv2.imshow("color edge", color_edge)
10
11  image = cv2.imread("images/color_edge.jpg", cv2.IMREAD_COLOR)
12  if image is None: raise Exception("영상파일 읽기 오류")
13
14  th = 50
15  rep_image = cv2.repeat(image, 1, 2)                          #
16  rep_gray = cv2.cvtColor(rep_image, cv2.COLOR_BGR2GRAY)       #
17
18  cv2.namedWindow("color edge", cv2.WINDOW_AUTOSIZE)           #
19  cv2.createTrackbar("Canny th", "color edge", th, 100, onTrackbar) #
20  onTrackbar(th)                                               #
21  cv2.waitKey(0)
```

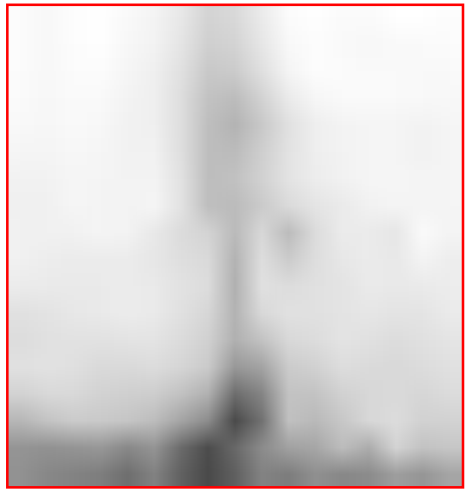


에지를 유지하는 Smoothing





에지를 유지하는 Smoothing



에지를 유지하는 Smoothing

❖Bilateral filter

```
bilateralFilter (InputArray src, OutputArray dst,
        int  d[1],      // 각 화소의 이웃을 결정하는 지름
        double sigmaColor,    // 컬러공간의 필터 표준편차
        double sigmaSpace,   // 좌표공간의 필터 표준편차
        int borderType[1]=BORDER_DEFAULT
)
```

1) d > 0 이면, sigmaSpace는 무시됨
   d < 0 이면, d=2 x (3 x sigmaSpace) + 1

Summary

❖2차 미분 연산

- 라플라시안(Laplacian), LoG(Laplacian of Gaussian), DoG(Difference of Gaussian)

❖캐니 에지 알고리즘

1. 블러링을 통한 노이즈 제거 (가우시안 블러링)
2. 화소 기울기(gradiant)의 강도와 방향 검출 (소벨 마스크)
3. 비최대치 억제(non-maximum suppression)
4. 이력 임계값(hysteresis threshold)으로 에지 결정

Summary

❖비선형 공간 필터링의 방법
- 최소값, 최대값 필터링, 평균값 필터링, 미디언 필터링
  - 최소값 필터링을 적용하면 영상이 전반적으로 어두워지며
  - 최대값 필터링을 적용하면 영상이 밝아진다.

❖평균값 필터링
- 입력화소들을 평균하여 출력화소를 결정하
- 블러링과 같은 효과가 난다.

❖미디언 필터링
- 입력화소들을 정렬하여 중간값을 출력화소로 결정
- 소금 후추 잡음 제거에 효과적이다.

Summary

❖가우시안 블러링(Gaussian Blurring)

- 정규분포 곡선을 갖는 마스크를 가우시안 수식에 따라서 생성하고 이 마스크로 회선을 수행
- 평균과 표준 편차로 정규분포 마스크를 생성 → 표준편차가 크면 클수록 많이 흐려진 영상을 생성한다.

❖Bilateral 필터링

- 현재 화소와 이웃화소의 색상/밝기 값의 차이에 따라 출력 화소의 값을 계산
- 어떤 이웃화소들을 포함시킬지의 여부를 결정함으로써, 에지를 유지하면서 블러링을 수행한다.