

# CHAPTER 07 회선(CONVOLUTION)

1

## CONTENTS

- 7.1 회선(CONVOLUTION)
- 7.2 에지 검출
- 7.3 기타 필터링
- 7.4 모폴로지(MORPHOLOGY)

2

## 7.1 회선(convolution)

- ◆ 7.1.1 공간 영역의 개념과 회선
- ◆ 7.1.2 블러링
- ◆ 7.1.3 샤프닝

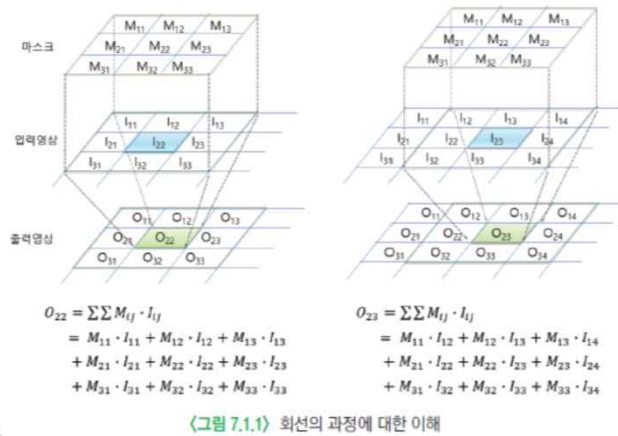
3

## 7.1.1 공간 영역의 개념과 회선

- ◆ 화소 기반 처리
  - ❖ 화소값 각각에 대해 여러 가지 연산 수행
- ◆ 영역 기반 처리
  - ❖ 마스크(mask)라 불리는 규정된 영역을 기반으로 연산 수행
    - 커널(kernel), 윈도우(window), 필터(filter)

4

## 7.1.1 공간 영역의 개념과 회선



5

## 7.1.2 블러링

## ◆ 블러링 현상

- ❖ 디지털 카메라로 사진 찍을 때, 초점이 맞지 않으면 → 사진 흐려짐
- ❖ 이 현상 이용 → 영상의 세밀한 부분 제거하는 아웃 포커싱(out focusing) 기법
  - 포토샵을 이용한 'ぼかし'
  - 사진 편집앱의 '보자사' 기능

## ◆ 블러링(blurring) 처리

- ❖ 영상에서 화소값이 급격하게 변하는 부분들을 감소시켜 점진적으로 변하게 함으로써 영상이 전체적으로 부드러운 느낌이 나게 하는 기술

## ◆ 화소값이 급격이 변화하는 것을 점진적으로 변하게 하는 방법

- ❖ → 블러링 마스크로 회선 수행

6

## 7.1.2 블러링

## ◆ 블러링 마스크

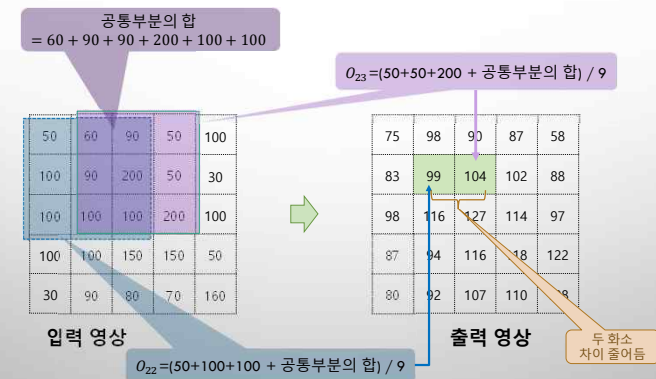
1	1	1
9	9	9
1	1	1
9	9	9
1	1	1
9	9	9

1	1	1	1	1
25	25	25	25	25
1	1	1	1	1
25	25	25	25	25
1	1	1	1	1
25	25	25	25	25
1	1	1	1	1
25	25	25	25	25

〈그림 7.1.2〉 블러링 마스크의 예

7

## 7.1.2 블러링



8

## 7.1.2 블러링

## 예제 7.1.1 화선이용 블러링 - 01.blurring.py

```

01 import numpy as np, cv2
02
03 ## 화선 수행 함수 - 행렬 처리 방식(속도 면에서 유리)
04 def filter(image, mask):
05     rows, cols = image.shape[:2]
06     dst = np.zeros((rows, cols), np.float32) # 화선 결과 저장 행
07     ycenter, xcenter = rows//2, cols//2 # 마스크 중심 좌표
08
09     for i in range(ycenter, rows - ycenter): # 입력 행렬
10         for j in range(xcenter, cols - xcenter): # 마스크 원소 순회
11             y1, y2 = i - ycenter, i + ycenter + 1 # 관심 영역
12             x1, x2 = j - xcenter, j + xcenter + 1 # 관심 영역 내비 범위
13             roi = image[y1:y2, x1:x2].astype("float32") # 관심 영역 행변환
14             tmp = cv2.multiply(roi, mask) # 화선 적용-원소간 곱셈
15             dst[i, j] = cv2.sumElems(tmp)[0] # 출력 화소 저장
16     return dst # 자료형 변환하여 반환
17
18 ## 화선 수행 함수- 화소 직접 접근
19 def filter2(image, mask):
20     rows, cols = image.shape[:2]
21     dst = np.zeros((rows, cols), np.float32) # 화선 결과 저장 행
22     ycenter, xcenter = rows//2, cols//2 # 마스크 중심 좌표
23
24     for i in range(ycenter, rows - ycenter): # 입력 행렬 반복 순
25         for j in range(xcenter, cols - xcenter):
26             sum = 0.0
27             for u in range(mask.shape[0]): # 마스크 원소 순회
28                 for v in range(mask.shape[1]):
29                     y, x = i + u - ycenter, j + v - xcenter
30                     sum += image[y, x] * mask[u, v] # 화선
31             dst[i, j] = sum
32     return dst

```

9

## 7.1.2 블러링

```

34 image = cv2.imread("images/filter_blur.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
35 if image is None: raise Exception("영상파일 읽기 오류")
36
37 data = [ 1/9, 1/9, 1/9, # 블러링 마스크 원소 지정
38         1/9, 1/9, 1/9,
39         1/9, 1/9, 1/9]
40 mask = np.array(data, np.float32).reshape(3, 3) # 마스크 행렬 생성
41 blur1 = filter(image, mask) # 화선 수행- 행렬 처리 방식
42 blur2 = filter2(image, mask) # 화선 수행- 화소 직접 접근
43 blur1 = blur1.astype("uint8") # 행렬 표시위해 uint8형 변환
44 blur2 = cv2.convertScaleAbs(blur2) # 0-255 스케일 및 절대값
45
46 cv2.imshow("image", image)
47 cv2.imshow("blur1", blur1)
48 cv2.imshow("blur2", blur2)
49 cv2.waitKey(0)

```

10

## 7.1.2 블러링

## ◆ 실행결과



블러링 수행

11

## 7.1.3 샤프닝

## ◆ 샤프닝(sharpening) 처리

- ❖ 출력화소에서 이웃 화소끼리 차이를 크게 해서 날카로운 느낌이 나게 만드는 것
- ❖ 영상의 세세한 부분을 강조할 수 있으며, 경계 부분에서 명암대비가 증가되는 효과

## ◆ 샤프닝 마스크

- ❖ 마스크 원소들의 값 차이가 커지도록 구성
- ❖ 마스크 원소 전체합 = 1 → 입력영상 밝기 손실 없이 출력영상 밝기 유지

0	-1	0
-1	5	-1
0	-1	0

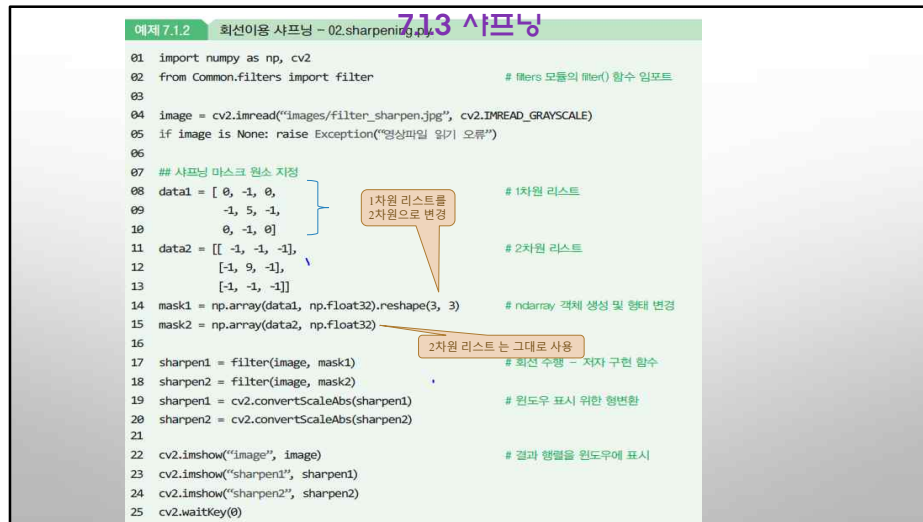
-1	-1	-1
-1	9	-1
-1	-1	-1

1	-2	1
-2	5	-2
1	-2	1

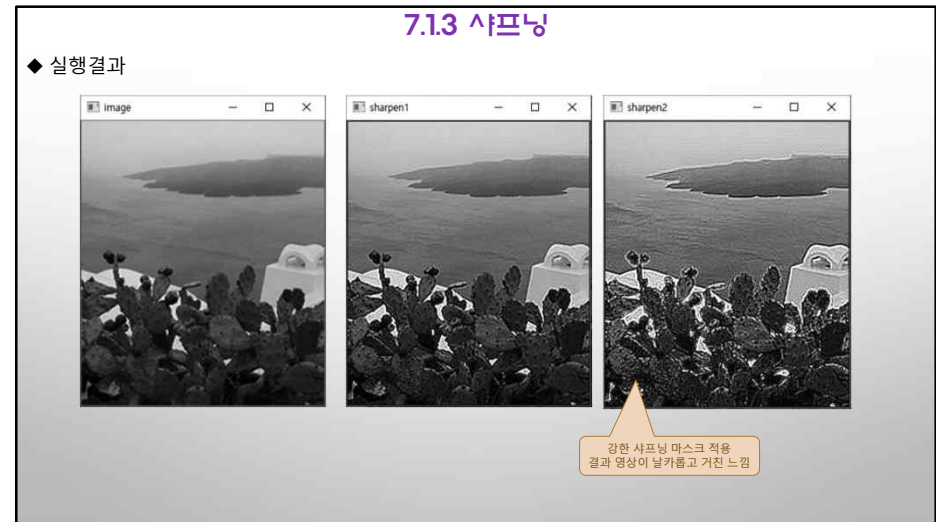
마스크 중심계수

〈그림 7.1.4〉 샤프닝 마스크의 예

12



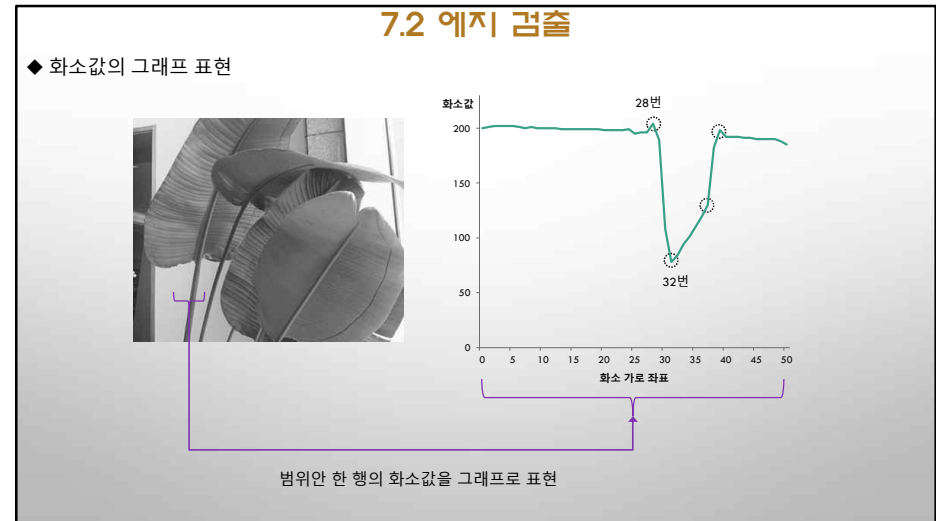
13



14



15



16

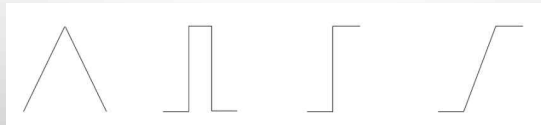
## 7.2 에지 검출

## ◆ 2) 에지 검출

## ❖ 에지(Edge)란

- 가장자리, 윤곽선, '엣지 있다'
- 영상의 밝기 값이 변화하는 부분

## ❖ 에지의 패턴

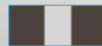


지붕형 roof

선형 line

계단형 step

경사형 ramp



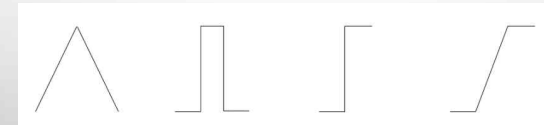
## 7.2.2 1차 미분 마스크

## ◆ 2) 에지 검출

## ❖ 에지(Edge)란

- 가장자리, 윤곽선, '엣지 있다'
- 영상의 밝기 값이 변화하는 부분

## ❖ 에지의 패턴

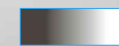
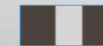


지붕형 roof

선형 line

계단형 step

경사형 ramp



17

18

## 7.2.2 1차 미분 마스크

## ◆ 미분

- ❖ 함수의 순간 변화율을 구하는 계산 과정을 의미
- ❖ 에지가 화소의 밝기가 급격히 변하는 부분이기 때문에 함수의 변화율을 취하는 미분 연산을 이용해서 에지 검출 가능

## ◆ 밝기의 변화율을 검출하는 방법

- ❖ 밝기에 대한 기울기(gradient)를 계산

$$G[f(x, y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

$$G_x = \frac{f(x+dx, y) - f(x, y)}{dx} \approx f(x+1, y) - f(x, y), \quad dx = 1$$

$$G_y = \frac{f(x, y+dy) - f(x, y)}{dy} \approx f(x, y+1) - f(x, y), \quad dy = 1$$

$$G[f(x, y)] \approx \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$

$$\theta = \tan^{-1} \left( \frac{G_y}{G_x} \right)$$

디지털 영상 :  
1픽셀씩으로 근사

밝기 변화율 : 에지 강도

기울기 : 에지의 방향

## 1) 로버츠(Roberts) 마스크

## ◆ 대각선 방향으로 1과 -1을 배치하여 구성

$$G_x = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

대각방향 마스크 1

$$G_y = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

대각방향 마스크 2

$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

〈그림 7.2.3〉 3×3 크기의 로버츠 마스크

19

20

## 1) 로버츠(Roberts) 마스크

예제 7.2.3 로버츠 에지 검출 - 03.edge\_roberts.py

```

01 import numpy as np, cv2
02 from Common.filters import filter # filters 모듈의 filter() 함수 임포트
03
04 def differential(image, data1, data2):
05     mask1 = np.array(data1, np.float32).reshape(3, 3) # 입력 인자로 마스크 원소 초기화
06     mask2 = np.array(data2, np.float32).reshape(3, 3)
07
08     dst1 = filter(image, mask1) # 저자 구현 회선 함수 호출
09     dst2 = filter(image, mask2)
10     dst1, dst2 = np.abs(dst1), np.abs(dst2) # 회선 결과 행렬 양수 변경
11     dst = cv2.magnitude(dst1, dst2) # 두 행렬의 크기 계산
12
13     dst = np.clip(dst, 0, 255).astype('uint8') # 윈도우 영상 표시위한
14     dst1 = np.clip(dst1, 0, 255).astype('uint8') # 행변환 및 클램핑
15     dst2 = np.clip(dst2, 0, 255).astype('uint8')
16     return dst, dst1, dst2 # 3개 결과 행렬 반환
17

```

대각 방향 마스크 2개

두 행렬 원소의 벡터 크기로 에지 강도

21

## 1) 로버츠(Roberts) 마스크

```

18 image = cv2.imread("images/edge.jpg", cv2.IMREAD_GRAYSCALE)
19 if image is None: raise Exception("영상파일 읽기 오류")
20
21 data1 = [ -1, 0, 0, # 마스크 원소 - 1차원 리스트
22           0, 1, 0,
23           0, 0, 0]
24 data2 = [ 0, 0, -1,
25           0, 1, 0,
26           0, 0, 0]
27 dst, dst1, dst2 = differential(image, data1, data2) # 회선 수행 및 두 방향의 크기 계산
28
29 cv2.imshow("image", image)
30 cv2.imshow("roberts edge", dst)
31 cv2.imshow("dst1", dst1)
32 cv2.imshow("dst2", dst2)
33 cv2.waitKey(0)

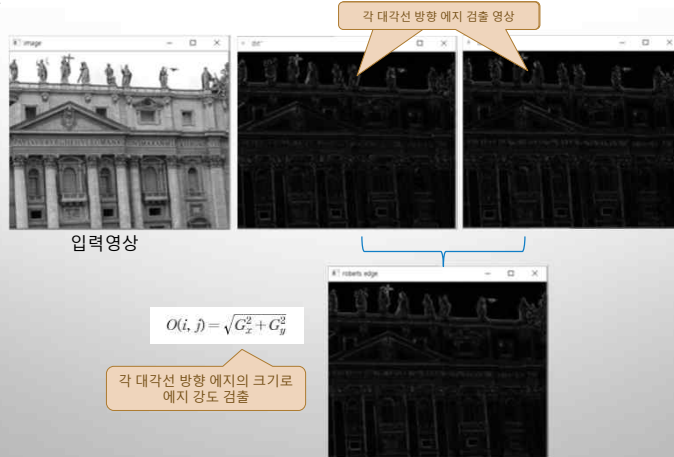
```

로버츠 마스크 원소

22

## 1) 로버츠(Roberts) 마스크

## ◆ 실행결과



23

## 2) 프리윗(Prewitt) 마스크

## ◆ 로버츠 마스크의 단점을 보완하기 위해 고안

- ❖ 수직 마스크 - 원소의 배치가 수직 방향으로 구성, 에지의 방향도 수직
- ❖ 수평 마스크 - 원소의 배치가 수평 방향으로 구성, 에지의 방향도 수평

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

수직 마스크                      수평 마스크

$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

〈그림 7.2.4〉 3×3 크기의 로버츠 마스크

24

## 2) 프리윗(Prewitt) 마스크

예제 7.2.4 프리윗 에지 검출 - 04.edge\_prewitt.py

```
01 import numpy as np, cv2
02 from Common.filters import filter          # filters 모듈의 filter() 함수 임포트
03
04 def differential(image, data1, data2):
05     mask1 = np.array(data1, np.float32).reshape(3, 3)
06     mask2 = np.array(data2, np.float32).reshape(3, 3)
07
08     dst1 = filter(image, mask1)              # 사용자 정의 함수 호출
09     dst2 = filter(image, mask2)
10     dst = cv2.magnitude(dst1, dst2)         # 최선 결과 두 행렬의 크기 계산
11
12     dst = cv2.convertScaleAbs(dst)           # 절대값 및 형변환
13     dst1 = cv2.convertScaleAbs(dst1)
14     dst2 = cv2.convertScaleAbs(dst2)
15     return dst, dst1, dst2
16
```

윈도우 영상 표시위해

## 2) 프리윗(Prewitt) 마스크

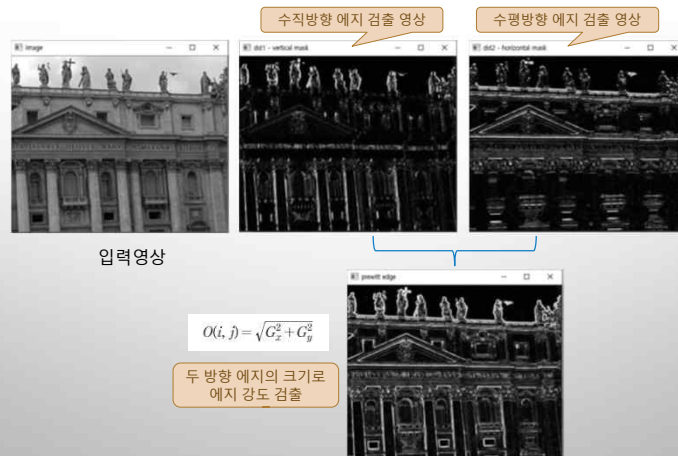
```
17 image = cv2.imread("images/edge.jpg", cv2.IMREAD_GRAYSCALE)
18 if image is None: raise Exception("영상파일 읽기 오류")
19
20 data1 = [ -1, 0, 1,                # 프리윗 수직 마스크
21          -1, 0, 1,
22          -1, 0, 1]
23 data2 = [ -1,-1,1,                # 프리윗 수평 마스크
24          0, 0, 0,
25          1, 1, 1]
26 dst, dst1, dst2 = differential(image, data1, data2)
27
28 cv2.imshow("image", image)
29 cv2.imshow("prewitt edge", dst)
30 cv2.imshow("dst1 - vertical mask", dst1)
31 cv2.imshow("dst2 - horizontal mask", dst2)
32 cv2.waitKey(0)
```

25

26

## 2) 프리윗(Prewitt) 마스크

### ◆ 실행결과



27

## 3) 소벨(Sobel) 마스크

### ◆ 프리윗 마스크와 유사, 중심화소의 차분에 대한 비중을 2배 키운 것이 특징

- ❖ 수직, 수평 방향 에지 추출
- ❖ 특히, 중심화소의 차분 비중을 높였기 때문에 대각선 방향 에지 검출

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

수직마스크                      수평마스크

$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

〈그림 7.2.5〉 3×3 크기의 소벨 마스크

28



### 예제 7.2.5 소벨 에지 검출 - 05.edge\_sobel.py

```

01 import numpy as np, cv2
02 from Common.filters import differential # filters 모듈의 differential() 함수 임포트
03
04 image = cv2.imread("images/edge.jpg", cv2.IMREAD_GRAYSCALE)
05 if image is None: raise Exception("영상파일이 읽기 오류")
06
07 data1 = [ -1, 0, 1, # 수직 소벨 마스크
08          -2, 0, 2,
09          -1, 0, 1]
10 data2 = [ -1,-2,-1, # 수평 소벨 마스크
11          0, 0, 0,
12          1, 2, 1]
13 dst, dst1, dst2 = differential(image, data1, data2) # 두 방향 회전 및 크기(에지 강도) 계산
14
15 ## OpenCV 제공 소벨 에지 계산
16 dst3 = cv2.Sobel(np.float32(image), cv2.CV_32F, 1, 0, 3) # x 방향 미분-수직 마스크
17 dst4 = cv2.Sobel(np.float32(image), cv2.CV_32F, 0, 1, 3) # y 방향 미분-수평 마스크
18 dst3 = cv2.convertScaleAbs(dst3) # 절댓값 > uint8(CV_8U) 형변환
19 dst4 = cv2.convertScaleAbs(dst4)
20
21 cv2.imshow("dst1- vertical_mask", dst1)
22 cv2.imshow("dst2- horizontal_mask", dst2)
23 cv2.imshow("dst3- vertical_OpenCV", dst3)
24 cv2.imshow("dst4- horizontal_OpenCV", dst4)
25 cv2.waitKey(0)

```

3) 소벨(Sobel) 마스크


x방향 미분

y방향 미분


29

### 3) 소벨(Sobel) 마스크

◆ 실행결과



입력영상



저자 구현 함수 적용

OpenCV 제공 함수 적용

30

### 7.2.4 2차 미분 마스크

◆ 1) 라플라시안 에지 검출

- 피에르시몽 라플라스 라는 프랑스의 수학자 이름을 따서 지은 것
- 함수  $f$  에 대한 그래디언트의 발산으로 정의

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f \quad \Rightarrow \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

2차원 좌표계

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

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

$$\nabla^2 f(x, y) = f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4 \cdot f(x, y)$$

31

### 7.2.4 2차 미분 마스크

◆ 최종 수식

$$\nabla^2 f(x, y) = f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4 \cdot f(x, y)$$

◆ 수식에 따른 마스크 예시

0	-1	0
-1	4	-1
0	-1	0

(a) 4방향 마스크

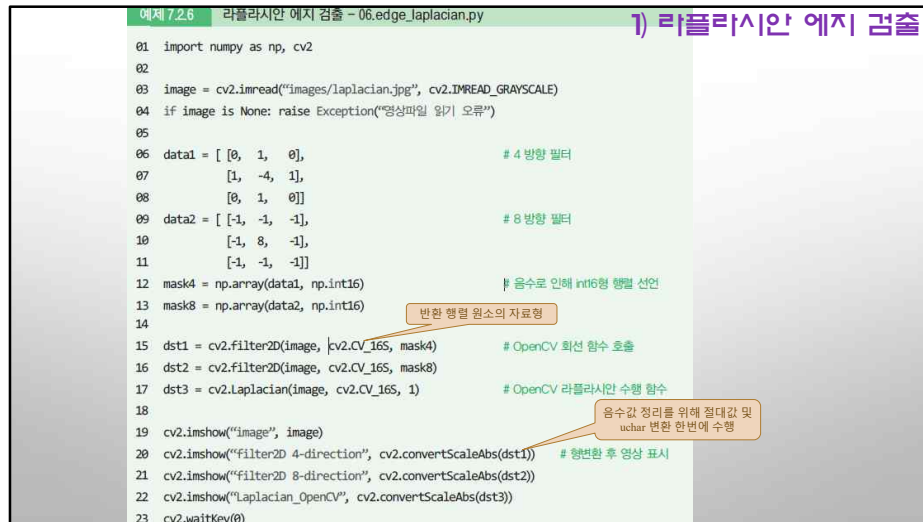
0	1	0
1	-4	1
0	1	0

(b) 8방향 마스크

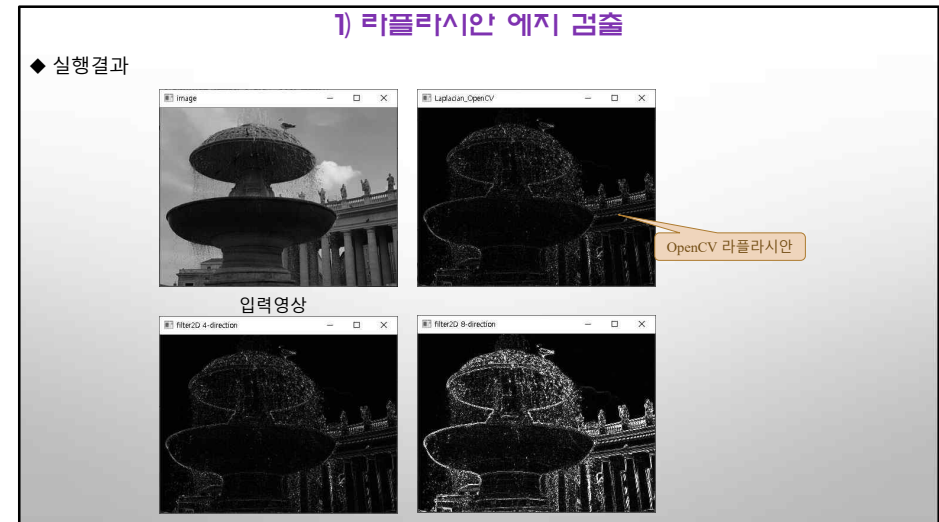
《그림 7.2.6》 라플라시안 마스크의 예

32





33



34

## 2) LoG와 DoG

### ◆ LoG(Laplacian of Gaussian)

- ❖ 라플라시안은 잡음에 민감한 단점
- ❖ 먼저 잡음 제거 후 라플라시안 수행 → 잡음에 강한 에지 검출 가능
- ❖ 다양한 잡음 제거 방법 있음
  - 비선형 필터링은 계산에서 속도 저하문제 발생
  - 선형 필터링으로 단일 마스크 생성

$$\Delta[G_\sigma(x, y) * f(x, y)] = [\Delta G_\sigma(x, y)] * f(x, y) = LoG * f(x, y)$$

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

35

## 2) LoG와 DoG

### ◆ DoG(Difference of Gaussian)

- ❖ 단순한 방법으로 2차 미분 계산
- ❖ 가우시안 스무딩 필터링의 차이를 이용해서 에지를 검출하는 방법

$$DoG(x, y) = \left( \frac{1}{2\pi\sigma_1^2} \cdot e^{-\frac{x^2 + y^2}{2\sigma_1^2}} \right) - \left( \frac{1}{2\pi\sigma_2^2} \cdot e^{-\frac{x^2 + y^2}{2\sigma_2^2}} \right) \quad (\text{단, } \sigma_1 < \sigma_2)$$

36

## 2) LoG와 DoG

예제 7.27 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("영상파일이 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')) # 행변환 후 영상 표시
15 cv2.imshow("dst2- DoG", dst2)
16 cv2.waitKey(0)
```

37

## 2) LoG와 DoG

### ◆ 실행결과



입력영상

38

## 7.2.5 캐니 에지 검출

- ◆ 잡음은 다른 부분과 경계를 이루는 경우 많음
  - ❖ 대부분의 에지 검출 방법이 이 잡음들을 에지로 검출
  - ❖ 이런 문제를 보완하는 방법 중의 하나가 캐니 에지(Canny Edge) 검출 기법

- ◆ 캐니 에지 검출 - 여러 단계의 알고리즘으로 구성된 에지 검출 방법

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

39

## 7.2.5 캐니 에지 검출

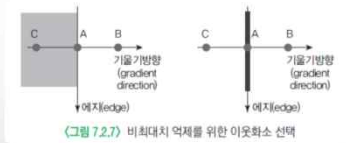
- ◆ 1) 블러링 - 5x5 크기의 가우시안 필터 적용
  - ❖ 불필요한 잡음 제거 및 필터 크기는 변경가능
- ◆ 2) 화소 기울기(gradient) 검출
  - ❖ 가로 방향과 세로 방향의 소벨 마스크로 회선 적용
  - ❖ 회선 완료 행렬로 화소 기울기의 크기(magnitude)와 방향(direction) 계산
  - ❖ 기울기 방향은 4개 방향(0, 45, 90, 135)으로 근사하여 단순화

40

## 7.2.5 캐니 에지 검출

## ◆ 3) 비최대치 억제(non-maximum suppression)

❖ 기울기의 방향과 에지의 방향은 수직



❖ 기울기 방향에 따른 이웃 화소 선택

0	1	2	0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8	6	7	8

기울기 방향: 0      기울기 방향: 45      기울기 방향: 90      기울기 방향: 135

〈그림 7.2.8〉 비최대치 억제를 위한 이웃 화소 선택

41

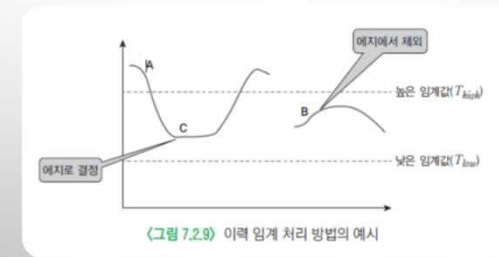
## 7.2.5 캐니 에지 검출

## ◆ 4) 이력 임계값 방법(hysteresis thresholding)

❖ 두 개의 임계값(Thigh, Tlow) 사용해 에지 이력 추적으로 에지 결정

❖ 각 화소에서 높은 임계값보다 크면 에지 추적 시작

• →추적하면 추적하지 않은 이웃 화소로 낮은 임계값보다 큰 화소를 에지로 결정



42

예제 7.2.8 캐니 에지 검출 - 08.edge\_canny.py

## 7.2.5 캐니 에지 검출

```

01 import numpy as np, cv2
02
03 def nonmax_suppression(sobel, direct):
04     rows, cols = sobel.shape[2]
05     dst = np.zeros((rows, cols), np.float32)
06     for i in range(1, rows - 1):
07         for j in range(1, cols - 1):
08             # 관심 영역 탐색 통해 이웃 화소 가져오기
09             values = sobel[i-1:i+2, j-1:j+2].flatten()
10             first = [3, 0, 1, 2]
11             id = first[direct[i, j]]
12             v1, v2 = values[id], values[8-id]
13
14             # if 문으로 이웃 화소 가져오기
15             # if direct[i, j] == 0: # 기울기 방향 0도
16             #     v1, v2 = sobel[i, j-1], sobel[i, j+1]
17             # if direct[i, j] == 1: # 기울기 방향 45도
18             #     v1, v2 = sobel[i-1, j-1], sobel[i+1, j+1]
19             # if direct[i, j] == 2: # 기울기 방향 90도
20             #     v1, v2 = sobel[i-1, j], sobel[i+1, j]
21             # if direct[i, j] == 3: # 기울기 방향 135도
22             #     v1, v2 = sobel[i+1, j-1], sobel[i-1, j+1]
23
24             dst[i, j] = sobel[i, j] if (v1 < sobel[i, j] < v2) else 0 # 최대치 억제
25     return dst
  
```

이웃 화소 3x3 범위 1차원 전개

45도 방향  
0도 방향  
90도 방향  
135도 방향

중심화소가  
두 이웃 화소보다 작으면 억제

두 이웃 화소 합 = 8

기울기 방향 따라  
이웃 화소 선택

if문 사용

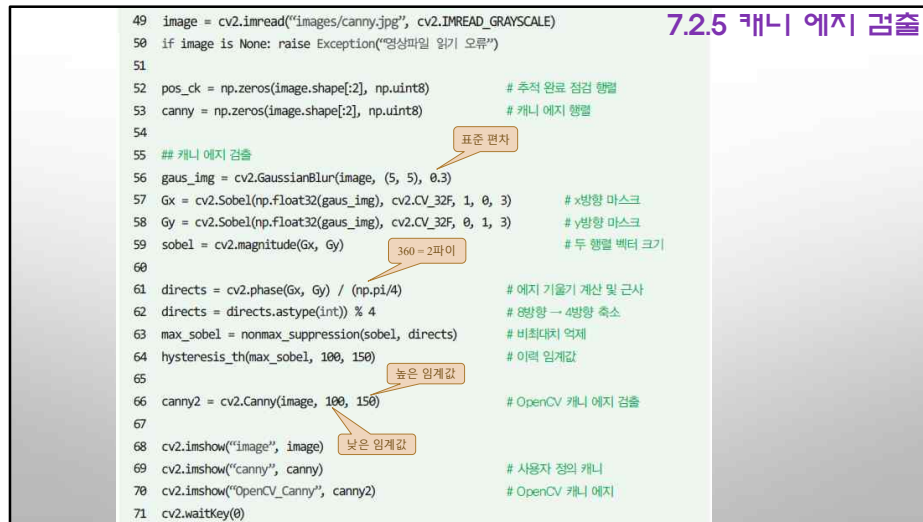
43

```

27 def trace(max_sobel, i, j, low): # 에지 추적 함수
28     h, w = max_sobel.shape
29     if (0 <= i < h and 0 <= j < w) == False: return # 추적 화소 범위 확인
30     if pos_ck[i, j] == 0 and max_sobel[i, j] > low: w: # 추적 조건 확인
31         pos_ck[i, j] = 255 # 추적 좌표 원료 표시
32         canny[i, j] = 255 # 에지 지정
33
34     trace(max_sobel, i - 1, j - 1, low) # 재귀 호출-8방향 추적
35     trace(max_sobel, i, j - 1, low)
36     trace(max_sobel, i + 1, j - 1, low)
37     trace(max_sobel, i - 1, j, low)
38     trace(max_sobel, i + 1, j, low)
39     trace(max_sobel, i - 1, j + 1, low)
40     trace(max_sobel, i, j + 1, low)
41     trace(max_sobel, i + 1, j + 1, low)
42
43 def hysteresis_th(max_sobel, low, high): # 이력 임계 처리 수행 함수
44     rows, cols = max_sobel.shape[2]
45     for i in range(1, rows - 1):
46         for j in range(1, cols - 1):
47             if max_sobel[i, j] >= high: trace(max_sobel, i, j, low) # 높은 임계값 이상시 추적
  
```

제귀 호출로  
8방향으로 추적 수행

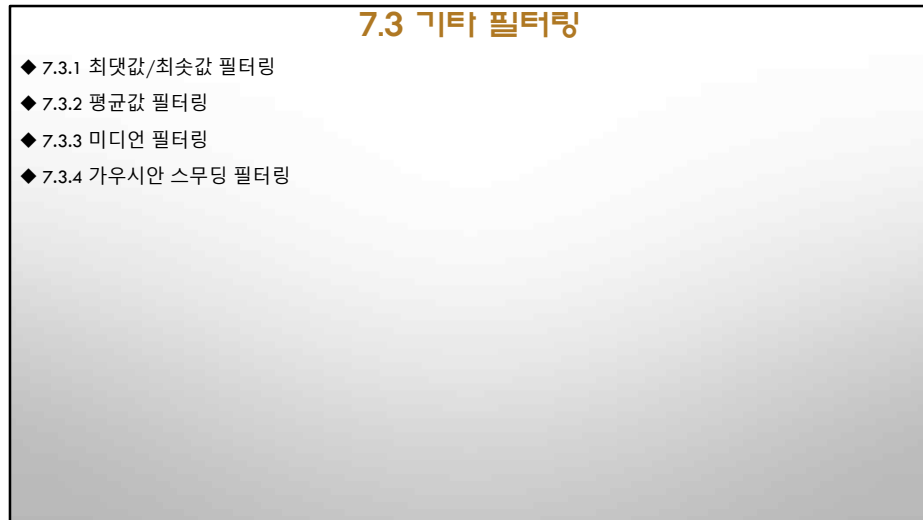
44



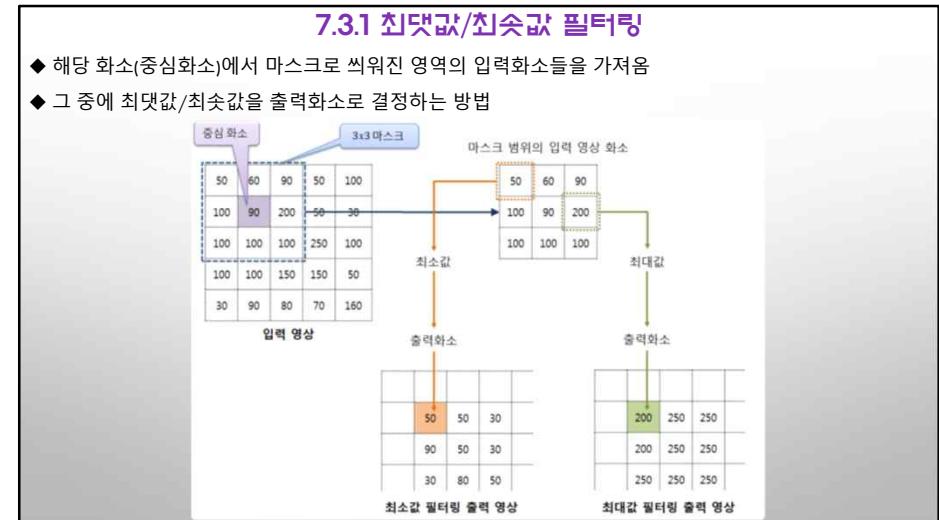
45



46



47



48

### 7.3.1 최댓값/최솟값 필터링

#### ◆ 최댓값 필터링

- ❖ 가장 큰 값인 밝은 색들로 출력화소가 구성
- ❖ 돌출되는 어두운 값이 제거 전체적으로 밝은 영상이 됨

#### ◆ 최솟값 필터링

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

49

예제 7.3.1 최솟값-최댓값 필터링 - 09.filter\_minMax.py

### 7.3.1 최댓값/최솟값 필터링

```

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

```

1이면 최댓값 필터링 수행,  
0이면 최솟값 필터링 수행

# 마스크 절반 크기

# 입력 영상 순회

입력영상의 모서리에서  
마스크 절반 크기 조회 안함

# 마스크 높이 범위

# 마스크 너비 범위

# 마스크 영역

# 최소 or 최대

# 3x3 마스크 최솟값 필터링

# 3x3 마스크 최댓값 필터링

50

### 7.3.1 최댓값/최솟값 필터링

#### ◆ 실행결과



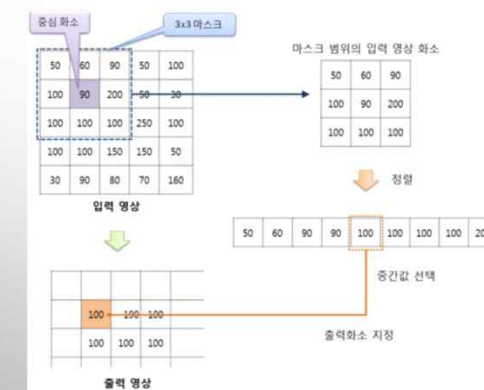
전체적으로 밝아짐

전체적으로 어두워짐

51

### 7.3.2 평균값 필터링

#### ◆ 마스크 영역 입력화소들의 평균을 구하여 출력화소로 지정하는 방법



52



## 7.3.2 평균값 필터링

예제 7.3.2 평균값 필터링 - 10.filter\_average.py

```

01 import numpy as np, cv2
02
03 def average_filter(image, ksize):          # 평균값 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(rows):                  # 입력 영상 순회
09         for j in range(cols):
10             y1, y2 = i - center, i + center + 1 # 마스크 높이 범위
11             x1, x2 = j - center, j + center + 1 # 마스크 너비 범위
12             if y1 < 0 or y2 > rows or x1 < 0 or x2 > cols: # 입력 영상 벗어남
13                 dst[i, j] = image[i, j]          # cv2.BORDER_CONSTANT 방식
14             else:
15                 mask = image[y1:y2, x1:x2]        # 범위 지정
16                 dst[i, j] = cv2.mean(mask)[0]    # cv2.mean(mask) 사용 가능
17
18 return dst

```

시작위치와 종료위치로 마스크 사각형 선언

## 7.3.2 평균값 필터링

```

19 image = cv2.imread("images/avg_filter.jpg", cv2.IMREAD_GRAYSCALE)
20 if image is None: raise Exception("영상파일이 읽기 오류")
21
22 avg_img = average_filter(image, 5)          # 사용자 정의 평균값 필터 함수
23 blur_img = cv2.blur(image, (5, 5), (-1,-1), cv2.BORDER_REFLECT) # OpenCV의 블러링
24 box_img = cv2.boxFilter(image, ddepth=-1, ksize=(5, 5)) # OpenCV의 박스 필터 함수
25
26 cv2.imshow("image", image)
27 cv2.imshow("avg_img", avg_img)
28 cv2.imshow("blur_img", blur_img)
29 cv2.imshow("box_img", box_img)
30 cv2.waitKey(0)

```

같은 기능을 수행하는 함수들

53

54

## 7.3.2 평균값 필터링

## ◆ 실행결과



55

## OpenCV에서 회선 수행시 경계 설정 방법

## ◆ cv::filter2D(), cv::blur(), cv::boxFilter(), cv::sepFilter2D() 등의 함수에서 적용

〈표 7.3.1〉 경계타입 결정 옵션상수

옵션 상수	값	설명
cv2.BORDER_CONSTANT	0	특정 상수값으로 대체 70 70 70 20 25 35 45 50 60 70 70 70
cv2.BORDER_REPLICATE	1	계산 가능한 경계의 출력 화소 하나만으로 대체 대체 화소 20 20 20 20 25 35 45 50 60 60 60 60
cv2.BORDER_REFLECT	2	계산 가능한 경계의 출력 화소로부터 대칭되게 한 화소씩 지정 대체 화소 35 25 20 20 25 35 45 50 60 60 50 45
cv2.BORDER_WRAP	3	영상의 왼쪽 끝과 오른쪽 끝이 연결되어 있다고 가정하여 한 화소씩 가져와서 지정 대체 화소 45 50 60 20 25 35 45 50 60 20 25 35

56



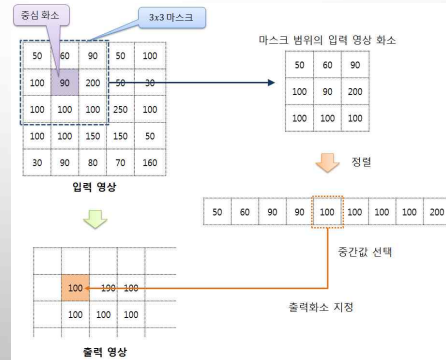
### 7.3.3 미디언 필터링

◆ 마스크 범위 원소 중 중간값 취하여 출력화소로 결정하는 방식

- ❖ 마스크 범위내에 있는 화소값 정렬 필요
- ❖ 임펄스 잡음, 소금-후추 잡음 제거

❖ RGB 컬러

- 3개 채널 간의 상호 의존도 큼
- →잡음이 많아 질 수 있음



57

### 7.3.3 미디언 필터링

예제 7.3.3 미디언 필터링 - 11.filter\_median.py

```
01 import numpy as np, cv2
02
03 def median_filter(image, size): # 미디언 필터링 함수
04     rows, cols = image.shape[:2]
05     dst = np.zeros((rows, cols), np.uint8)
06     center = size // 2 # 마스크 절반 크기
07
08     for i in range(center, rows - center): # 입력 영상 순회
09         for j in range(center, cols - center): # 마스크 높이 범위
10             y1, y2 = i - center, i + center + 1 # 마스크 너비 범위
11             x1, x2 = j - center, j + center + 1 # 관심 영역 지정 및 벡터 변환
12             mask = image[y1:y2, x1:x2].flatten()
13
14             sort_mask = cv2.sort(mask, cv2.SORT_EVERY_COLUMN) # 정렬 수행
15             dst[i, j] = sort_mask[sort_mask.size//2] # 출력 화소로 지정
16
17     return dst
```

58

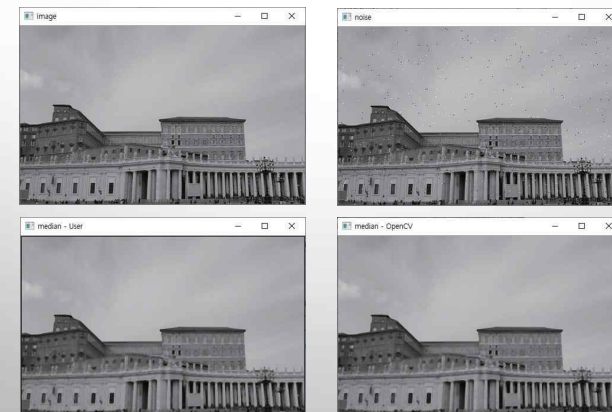
### 7.3.3 미디언 필터링

```
18 def salt_pepper_noise(img, n): # 소금 후추 잡음 생성 함수
19     h, w = img.shape[:2]
20     x, y = np.random.randint(0, w, n), np.random.randint(0, h, n)
21     noise = img.copy()
22     for (x, y) in zip(x, y):
23         noise[y, x] = 0 if np.random.rand() < 0.5 else 255
24     return noise
25
26 image = cv2.imread("images/median.jpg", cv2.IMREAD_GRAYSCALE)
27 if image is None: raise Exception("영상파일 읽기 오류")
28
29 noise = salt_pepper_noise(image, 500) # 소금-후추 잡음 영상 생성
30 med_img1 = median_filter(noise, 5) # 사용자 정의 함수
31 med_img2 = cv2.medianBlur(noise, 5) # OpenCV 제공 함수
32
33 cv2.imshow("image", image),
34 cv2.imshow("noise", noise),
35 cv2.imshow("median - User", med_img1)
36 cv2.imshow("median - OpenCV", med_img2)
37 cv2.waitKey(0)
```

59

### 7.3.3 미디언 필터링

◆ 실행결과



60

### 7.3.4 가우시안 스무딩 필터링

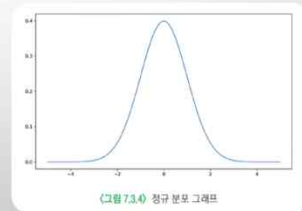
#### ◆ 스무딩

- ❖ 회선을 통해서 영상의 세세한 부분을 부드럽게 하는 기법
- ❖ 대표적인 방법 - 가우시안 필터링

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

#### ◆ 가우시안 분포(정규 분포)

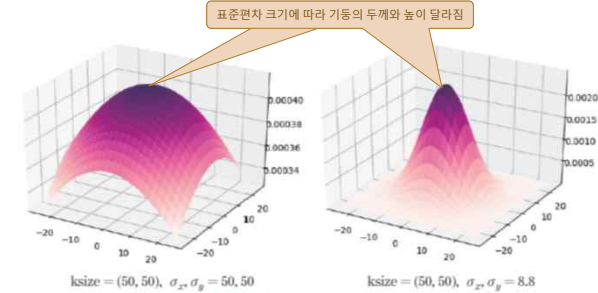
- ❖ 특정 값의 출현 비율을 그래프로 그렸을 때, 평균에서 가장 큰 수치 가짐
- ❖ 평균을 기준으로 좌우 대칭 형태
- ❖ 양끝으로 갈수록 수치가 낮아지는 종 모양
  - 평균과 표준 편차로 그래프 모양 변경



### 7.3.4 가우시안 스무딩 필터링

#### ◆ 2차원 가우시안 분포

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



61

62

### 7.3.4 가우시안 스무딩 필터링

예제 7.3.4 가우시안 필터링 - 12.filter\_gaussian.py

```
01 import numpy as np, cv2
02
03 def getGaussianMask(ksize, sigmaX, sigmaY):
04     sigma = 0.3 * ((np.array(ksize) - 1.0) * 0.5 - 1.0) + 0.8
05     if sigmaX <= 0: sigmaX = sigma[0]
06     if sigmaY <= 0: sigmaY = sigma[1]
07
08     u = np.arange(ksize)[0, :]
09     x = np.arange(-u[0], u[0]+1, 1)
10     y = np.arange(-u[1], u[1]+1, 1)
11     x, y = np.meshgrid(x, y)
12
13     ratio = 1 / (sigmaX * sigmaX * 2 * np.pi)
14     v1 = x ** 2 / (2 * sigmaX ** 2)
15     v2 = y ** 2 / (2 * sigmaY ** 2)
16     mask = ratio * np.exp(-(v1+v2))
17     return mask / np.sum(mask)
```

# 가우시안 마스크 생성 함수  
 # 표준편차 양수 입력  
 # ksize로 기본 표준  
 # 커널 크기  
 # x 방향 범위  
 # y 방향 범위  
 # 정방 행렬  
 # 가우시안 분포 공식  
 # 2차원 정규분포 공식  
 # 원소 전체 합 1 유지

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

### 7.3.4 가우시안 스무딩 필터링

```
18
19 image = cv2.imread("images/smoothing.jpg", cv2.IMREAD_GRAYSCALE)
20 if image is None: raise Exception("영상파일을 읽기 오류")
21
22 ksize = (17, 5)
23 gaussian_2d = getGaussianMask(ksize, 0, 0)
24 gaussian_1dx = cv2.getGaussianKernel(ksize[0], 0, cv2.CV_32F)
25 gaussian_1dy = cv2.getGaussianKernel(ksize[1], 0, cv2.CV_32F)
26
27 gauss_img1 = cv2.filter2D(image, -1, gaussian_2d)
28 gauss_img2 = cv2.GaussianBlur(image, size, 0)
29 gauss_img3 = cv2.sepFilter2D(image, -1, gaussian_1dx, gaussian_1dy)
30
31 titles = ['image', 'gauss_img1', 'gauss_img2', 'gauss_img3']
32 for t in titles: cv2.imshow(t, eval(t))
33 cv2.waitKey(0)
```

가로로 긴 마스크  
 2차원 가우시안 마스크 생성  
 # 커널 크기: 가로×세로  
 # 가로 방향 마스크  
 # 세로 방향 마스크  
 # 사용자 생성 마스크 적용  
 # OpenCV 제공 1-가우시안 블러링  
 # 1차원 가우시안 마스크로 cv2.sepFilter2D() 함수에 적용

다양한 가우시안 블러링 수행 방법

63

64

### 7.3.4 가우시안 스무딩 필터링

#### ◆ 실행결과



65

### 심화 예제 - 블러링 & 캐니에지

심화예제 7.3.5 블러링과 캐니 에지를 이용한 컬러 에지 검출 - 13.edge\_color\_canny.py

```
01 import cv2
02
03 def onTrackbar(th):
04     edge = cv2.GaussianBlur(gray, (5,5),0)
05     edge = cv2.Canny(edge,th,th*2,5)
06
07     color_edge = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
08     dst = cv2.cvtColor(color_edge, cv2.COLOR_RGB2BGR)
09     cv2.imshow("color edge", dst)
10
11 image = cv2.imread("images/color_edge.jpg", cv2.IMREAD_COLOR)
12 if image is None: raise Exception("영상파일이 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)
```

에지만 검출 위해 영상에서 부드러운 부분 제거

# 트랙바 콜백 함수

# 가우시안 블러링

# 캐니 에지 검출

원래값

복사시 에지 영상을 마스크로 사용 - 컬러 영상의 에지 부분만 복사

# 가로 반복 복사

# 명암도 영상 변환

# 윈도우 생성

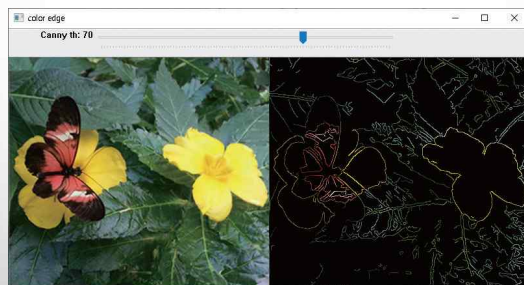
# 콜백 함수 등록

# 콜백 함수 첫 실행

66

### 심화예제

#### ◆ 실행결과



67

## 7.4 모폴로지(morphology)

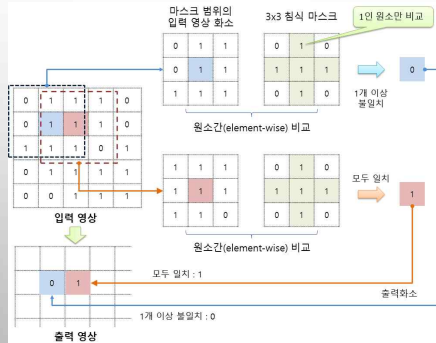
- ◆ 7.4.1 침식 연산
- ◆ 7.4.2 팽창 연산
- ◆ 7.4.3 열림 연산과 닫힘 연산
- ◆ 모폴로지- 형태학
  - ❖ 생물학
    - 생물형태의 기술과 그 법칙성의 탐구를 목적으로 일반적으로 해부학과 발생학을 합쳐서 형태학 이라 부름
  - ❖ 의학
    - 인체 형태학이란 의미로 사용
  - ❖ 체육학
    - 스포츠 운동 형태학이란 개념으로 사용
- ◆ 영상 처리에서 형태학
  - ❖ 영상의 객체들의 형태(shape)를 분석하고 처리하는 기법
  - ❖ 영상의 경계, 골격, 블록 등의 형태를 표현하는데 필요한 요소 추출
  - ❖ 영상 내에 존재하는 객체의 형태를 조금씩 변형시킴으로써 영상 내에서 불필요한 잡음 제거하거나 객체를 뚜렷하게 함

68

## 7.4.1 침식 연산

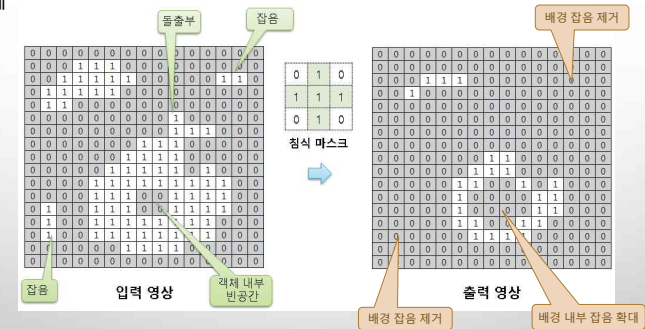
## ◆ 객체를 침식시키는 연산

- ❖ 객체의 크기 축소, 배경 확장
- ❖ 영상 내에 존재하는 잡음 같은 작은 크기의 객체 제거 가능
- ❖ 소금-후추 잡음과 같은 임펄스 잡음 제거



## 7.4.1 침식 연산

## ◆ 침식 연산의 예



## ◆ OpenCV 함수

- ❖ cv2.morphologyEx() - 옵션 : MORPH\_ERODE
- ❖ cv2.erode()

69

70

## 7.4.1 침식 연산

## 예제 7.4.1 모폴로지 침식 연산 - t4.erode.py

```
01 import numpy as np, cv2
02
03 def erode(img, mask=None): # 침식 연산 함수
04     dst = np.zeros(img.shape, np.uint8)
05     if mask is None: mask = np.ones((3, 3), np.uint8) # 마스크 없으면 생성
06     ycenter, xcenter = np.divmod(mask.shape[2], 2)[0] # 마스크 중심 좌표
07
08     mcnt = cv2.countNonZero(mask) # 마스크 1인 원소 개수
09     for i in range(ycenter, img.shape[0] - ycenter): # 입력 행렬 반복 순회
10         for j in range(xcenter, img.shape[1] - xcenter):
11             y1, y2 = i - ycenter, i + ycenter + 1 # 마스크 높이 범위
12             x1, x2 = j - xcenter, j + xcenter + 1 # 마스크 너비 범위
13             roi = img[y1:y2, x1:x2] # 마스크 영역
14             temp = cv2.bitwise_and(roi, mask) # 논리곱으로 일치 원소 지정
15             cnt = cv2.countNonZero(temp) # 일치 원소 개수 계산
16             dst[i, j] = 255 if (cnt == mcnt) else 0 # 출력 화소에 저장
17
18     return dst
```

모두 일치하면

하나라도 일치하지 않으면

71

## 7.4.1 침식 연산

```
19 image = cv2.imread("images/morph.jpg", cv2.IMREAD_GRAYSCALE)
20 if image is None: raise Exception("영상파일 읽기 오류")
21
22 data = [ 0, 1, 0,
23         1, 1, 1,
24         0, 1, 0] # 침식 마스크 원소
25 mask = np.array(data, np.uint8).reshape(3, 3) # 마스크 원소 지정
26 th_img = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)[1] # 마스크 행렬 생성
27 #th_img[image > 128] = 255 # 영상 이진화
28 dst1 = erode(th_img, mask) # 사용자 정의 침식 함수
29 dst2 = cv2.erode(th_img, mask) # OpenCV의 침식 함수
30 # dst2 = cv2.morphologyEx(th_img, cv2.MORPH_ERODE, mask) # OpenCV의 침식 함수2
31
32 cv2.imshow("image", image)
33 cv2.imshow("binary image", th_img)
34 cv2.imshow("User erode", dst1)
35 cv2.imshow("OpenCV erode", dst2)
36 cv2.waitKey(0)
```

128보다 큰 화소는 255, 작으면 0으로 이진화

72



## 7.4.1 침식 연산

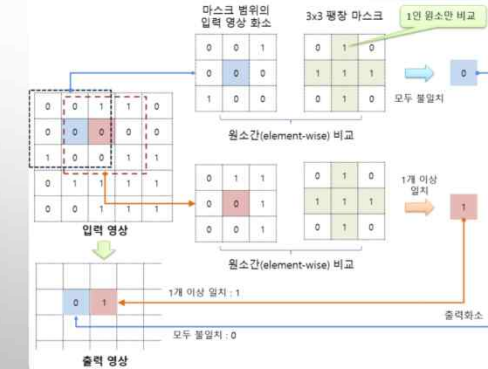
## ◆ 실행 결과



## 7.4.2 팽창 연산

## ❖ 객체를 팽창시키는 연산

- 객체의 최외곽 화소를 확장시키는 기능 → 객체 크기 확대, 배경 축소
- 객체 팽창으로 객체 내부의 빈 공간도 채워짐 → 객체 내부 잡음 제거



73

74

## 7.4.2 팽창 연산

## ◆ 팽창 연산의 예



〈그림 7.4.4〉 팽창 연산의 예

75

## 7.4.2 팽창 연산

## 예제 7.4.2 모폴로지 팽창 연산 - 15.dilate.py

```

01 import numpy as np, cv2
02
03 def dilate(img, mask):
04     dst = np.zeros(img.shape, np.uint8)
05     if mask is None: mask = np.ones((3, 3), np.uint8)
06     ycenter, xcenter = np.divmod(mask.shape[:2], 2)[0]
07
08     for i in range(ycenter, img.shape[0] - ycenter):
09         for j in range(xcenter, img.shape[1] - xcenter):
10             y1, y2 = i - ycenter, i + ycenter + 1
11             x1, x2 = j - xcenter, j + xcenter + 1
12             roi = img[y1:y2, x1:x2]
13             temp = cv2.bitwise_and(roi, mask)
14             cnt = cv2.countNonZero(temp)
15             dst[i, j] = 0 if (cnt == 0) else 255
16     return dst

```

# 팽창 수행 함수

# 마스크 중심 좌표

# 마스크

# 영상

# 사후

# Op

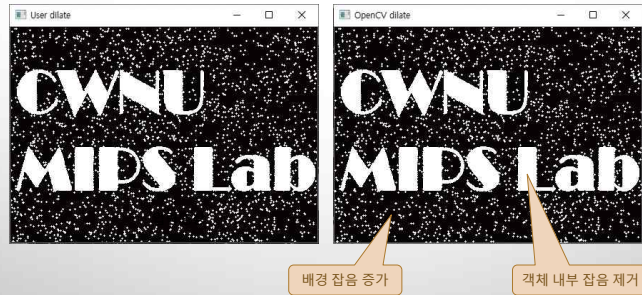
모두 불일치

하나라도 일치

76

## 7.4.2 팽창 연산

## ◆ 실행결과



77

## 7.4.3 열림 연산과 닫힘 연산

## ◆ 침식 연산과 팽창 연산의 순서를 조합하여 수행

## ❖ 열림 연산 - 침식 연산 → 팽창 연산

- 침식 연산으로 인해서 객체는 축소되고, 배경 부분의 미세한 잡음 제거
- 팽창 연산으로 인해서 축소되었던 객체들이 다시 원래 크기로



78

## 7.4.3 열림 연산과 닫힘 연산

## ❖ 닫힘 연산: 팽창 연산 → 침식 연산

- ❖ 팽창 연산으로 객체가 확장되어서 객체 내부의 빈 공간이 메워짐
- ❖ 침식 연산으로 다음으로 확장되었던 객체의 크기가 원래대로 축소



79

## 7.4.3 열림 연산과 닫힘 연산

예제 7.4.3 모폴로지 닫힘 &amp; 열림 연산 - 16.close\_open.py

```

01 import numpy as np, cv2
02 from Common.filters import erode, dilate # filters 모듈의 자차 구현 함수 임포트
03
04 def opening(img, mask): # 열림 연산 함수
05     tmp = erode(img, mask) # 침식
06     dst = dilate(tmp, mask) # 팽창
07     return dst
08
09 def closing(img, mask): # 닫힘 연산 함수
10     tmp = dilate(img, mask) # 팽창
11     dst = erode(tmp, mask) # 침식
12     return dst
13
14 image = cv2.imread("images/morph.jpg", cv2.THREAD_GRAYSCALE)
15 if image is None: raise Exception("영상파일 읽기 오류")
16
17 mask = np.array([[0, 1, 0],
18                 [1, 1, 1],
19                 [0, 1, 0]], dtype='uint8') # 마스크 생성 및 초기화
20 th_img = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)[1] # 영상 이진화
21
22 dst1 = opening(th_img, mask) # 열림 연산 함수
23 dst2 = closing(th_img, mask) # 닫힘 연산 함수
24 dst3 = cv2.morphologyEx(th_img, cv2.MORPH_OPEN, mask) # OpenCV의 열림 함수
25 dst4 = cv2.morphologyEx(th_img, cv2.MORPH_CLOSE, mask, iterations=1) # 닫힘 함수
26
27 cv2.imshow("User_opening", dst1); cv2.imshow("User_closing", dst2)
28 cv2.imshow("OpenCV_opening", dst3); cv2.imshow("OpenCV_closing", dst4)

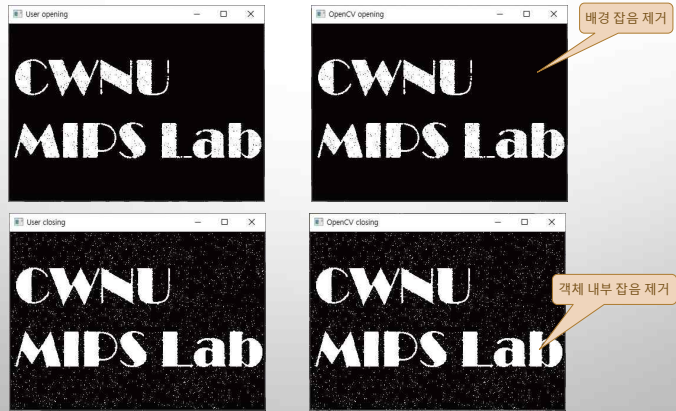
```

80



## 7.4.3 열림 연산과 닫힘 연산

## ◆ 실행결과



81

## 모폴로지 심화예제

## 심화예제 7.4.4 번호판 후보 객체 검출 - 17.detect\_plate.py

```

01 import numpy as np, cv2
02
03 while True:
04     no = int(input("차량 영상 번호( 0:종료 ) : "))
05     if no == 0: break
06
07     fname = "images/test_car/{0:02d}.jpg".format(no)
08     image = cv2.imread(fname, cv2.IMREAD_COLOR)
09     if image is None:
10         print(str(no) + "번 영상파일이 없습니다.")
11         continue
12
13     mask = np.ones((5, 17), np.uint8)
14     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
15     gray = cv2.blur(gray, (5, 5))
16     gray = cv2.Sobel(gray, cv2.CV_8U, 1, 0, 5)
17
18     # 이진화 및 닫힘 연산 수행
19     th_img = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)[1]
20     morph = cv2.morphologyEx(th_img, cv2.MORPH_CLOSE, mask, iterations=3)
21
22     cv2.imshow("image", image)
23     cv2.imshow("binary image", th_img)
24     cv2.imshow("opening", morph)

```

82

## 모폴로지 심화예제

## ◆ 실행결과



83

## 단원 요약

## ◆ 회선(convolution)

- ❖ 마스크 내의 원소값과 입력 영상의 화소값들을 곱하여 출력 화소값을 계산하는 것
- ❖ 이때, 입력 영상에 곱해지는 이 마스크를 커널(kernel), 윈도우(window), 필터(filter) 등

## ◆ 블러링(bluring)

- ❖ 회선 마스크의 원소를 모두 같은 값으로 지정해 수행하며, 전체 합이 1이 되어야한다.

## ◆ 샤프닝(sharpening)

- ❖ 중심계수와 주변계수의 차이를 크게 함으로서 출력화소가 도드라지게 함으로서 선명하고 날카로운 영상 생성

## ◆ 영상 처리에서 에지

- ❖ "화소값이 급격하게 변화하는 부분"으로 정의
- ❖ 객체에서 크기, 위치, 모양을 인지, 방향성 탐지

## ◆ 에지는

- ❖ 이웃하는 두 화소의 차분으로 구할 수 있으며, 미분 공식과 유사
- ❖ 미분 마스크로 회선을 수행하면 에지를 검출
- ❖ 1차 미분 마스크라하며, 대표적으로 소벨(Sobel), 프리윗(Prewitt), 로버츠(Roberts) 등

84

### 단원 요약

- ◆ 1차 미분 마스크 단점
  - ❖ 점진적으로 변화하는 부분까지 민감하게 에지를 검출하여 너무 많은 에지가 나타날 수 있다.
- ◆ 이를 보완하기 위한 방법으로 1차 미분에서 한 번 더 미분을 하는 방법인 2차 미분 연산
  - ❖ 라플라시안(Laplacian), LoG(Laplacian of Gaussian), DoG(Difference of Gauss-ian) 등의 방법이 있다.
- ◆ 캐니 에지 검출 방법은 John F. Canny에 의해 개발된 것으로서 다음과 같은 여러 단계의 알고리즘으로 구성된 에지 검출 방법이다.
 

1. 블러링을 통한 노이즈 제거 (가우시안 블러링)
  2. 화소 기울기(gradient)의 강도와 방향 검출 (소벨 마스크)
  3. 비최대치 억제(non-maximum suppression)
  4. 이력 임계값(hysteresis threshold)으로 에지 결정
- ◆ 비선형 공간 필터링의 방법으로 최솟값, 최댓값 필터링, 평균값 필터링, 미디언 필터링 등이 있다.
- ◆ 최대 최솟값 필터링은 마스크 범위에서 최솟값 혹은 최댓값을 출력화소로 결정한다. 최솟값 필터링은 영상이 전반적으로 어두워지며, 최댓값 필터링은 영상이 밝아진다.
- ◆ 평균값 필터링은 마스크 범위의 입력화소들을 평균하여 출력화소를 결정하기 때문에 블러링과 같은 효과가 난다.

85

### 단원 요약

- ◆ 미디언 필터링은 마스크 범위의 입력화소들을 정렬하여 중간값을 출력화소로 결정하는 방식이다. 임펄스 잡음이나 소금 후추 잡음은 마스크 범위 내에서 가장 큰 값 혹은 가장 작은 값이 되기 때문에 출력화소에서 배제된다. 따라서 이와 같은 잡음 제거에 효과적이다.
- ◆ 가우시안 블러링은 정규분포 곡선을 갖는 마스크를 가우시안 수식에 따라서 생성하고 이 마스크로 회전을 수행하는 방법이다. 평균과 표준 편차로 정규분포 마스크를 생성할 수 있다. 표준편차가 크면 클수록 많이 흐려진 영상을 생성한다.
- ◆ 모폴로지는 형태학적 방법을 영상 처리에 적용한 것으로서 침식과 팽창 연산이 있다. 침식연산은 객체의 크기가 축소되기 때문에 영상 내에 존재하는 작은 크기의 잡음을 제거하는 효과적이다. 팽창 연산은 객체의 크기가 확대되어 객체 내부의 빈 공간을 메우는 역할을 한다.
- ◆ 열림 연산은 침식 연산 수행 후에 팽창 연산을 수행한다. 침식 연산으로 객체는 축소되고, 배경의 잡음들은 제거되며, 팽창 연산으로 축소되었던 객체들이 원래 크기로 돌아간다.
- ◆ 닫힘 연산은 팽창 연산 수행 후에 침식 연산을 수행한다. 팽창 연산으로 객체가 확장되어 객체 내부의 빈 공간이 메워진다. 다음으로 침식 연산으로 확장되었던 객체의 크기가 원래대로 축소된다.

86