1주차 chap03

Chapter3. 영상 처리

3.1 디지털 영상 기초

디지털 카메라는 실제 세상에 존재하는 피사체 일정한 간격 샘플링하고 명암을 일정한 간격으로 양자화하는 과정을 통해 디지털 영상 획득

핀홀 카메라와 디지털 변환

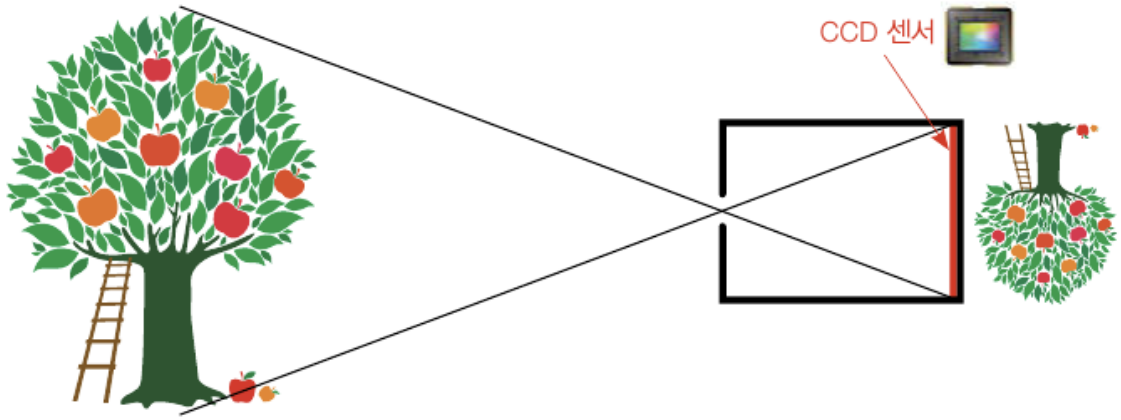
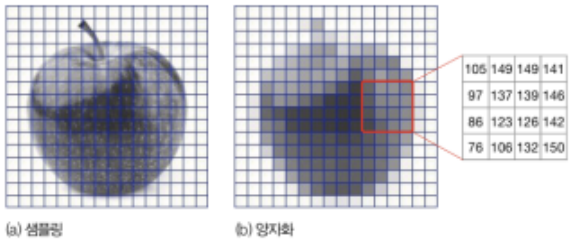


그림 3-2 핀홀 카메라 모델과 CCD 센서

카메라가 영상 획득하는 과정이 핀홀 카메라 모델은 핵심 설명

- 물체에 반사된 빛은 작은 구멍을 통해 안으로 들어감
- 이 빛은 영상 평면에 맺힘
- 빛이라는 아날로그 신호를 받은 영상 평면(OCC센서) 는 **디지털 신호**로 변환한 영상을 메모리에 저장
    - 디지털로 변환하는 과정에서 샘플링과 양자화 수행
    - 샘플링 : 2차원 영상 공간을 가로 방향 N개 세로방향 M개 구간으로 나눔

→ 이러한 점을 화소(픽셀), M x N을 영상 크기 또는 해상도



영상 좌표계

영상은 2차원 좌표계

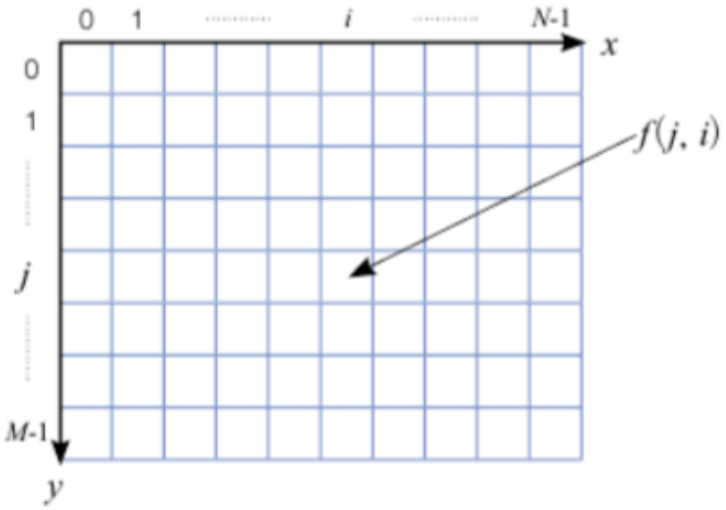


그림 3-4 디지털 영상의 좌표계

1. 원점*(0,0)*이 좌측 상단

   x축 *0, 1, … , N-1*로 정수 좌표를 가짐

   y축 *0, 1, … , M-1*의 정수 좌표를 가짐

2. *(y,x)* 표기

   행 좌표 먼저, 열 좌표 그 다음

   → 디지털 영상은 2차원 배열에 저장되기 때문에 이 관행 따름

   영상을 저장하는 배열에서 화소의 위치 지정할 때 사용하나 그 외엔 주로 *(x,y)*

3.1.2 다양한 종류의 영상

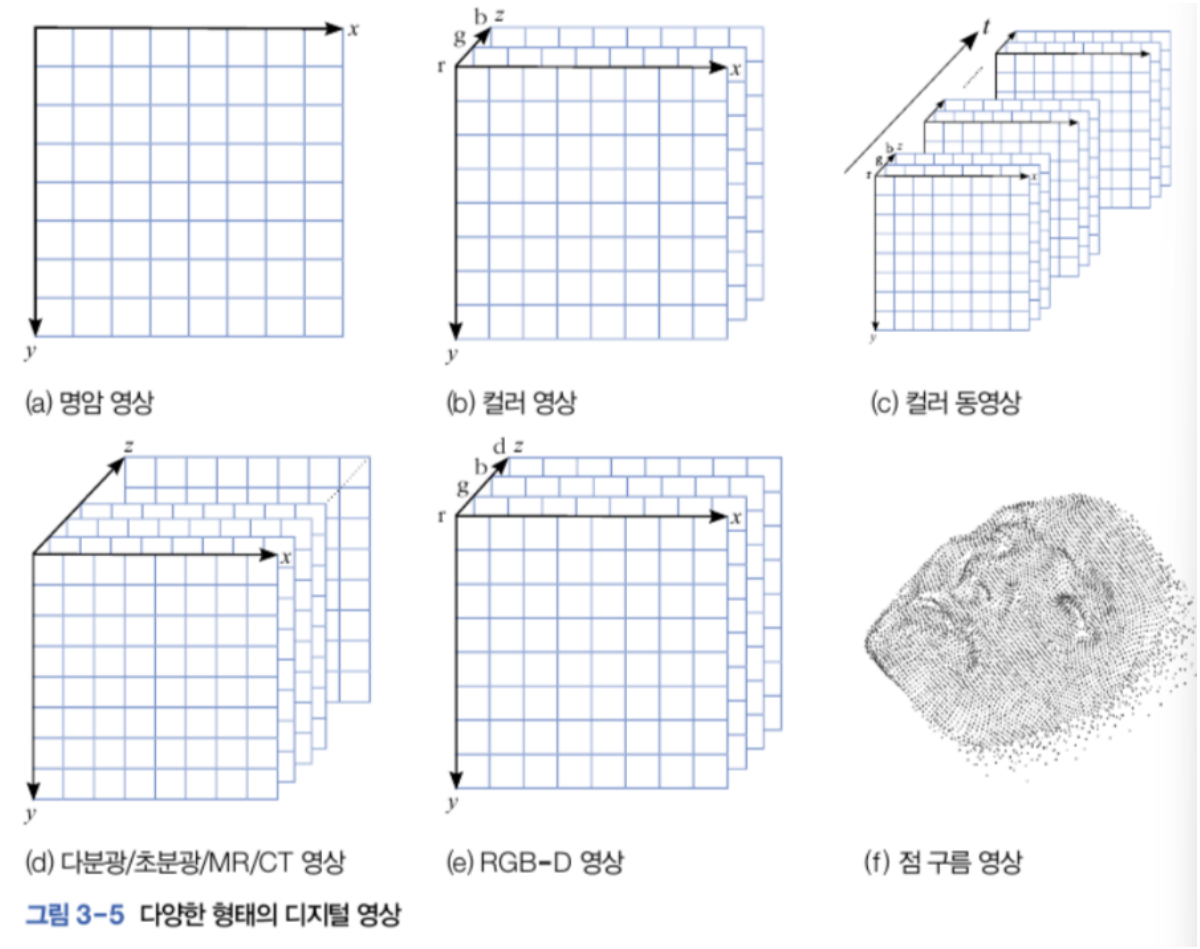


그림 3-5 다양한 형태의 디지털 영상

물체가 반사한 빛을 측정한 디지털 영상

a. **명암영상**은 채널이 하나이며 2차원 배열 구조

b. RGB는 **3개 채널 컬러 영상**으로 3차원 구조의 배열 표현

c. **웹캠 컬러 동영상**은 4차원 구조 배열

d. 3차원 구조의 배열과 다채널

다분광 영상으로 3-10개의 채널 존재하며 자외선과 적외선 영역까지 확장한 형태

초분광영상은 조밀하게 획득한 수백 수천개의 채널 존재

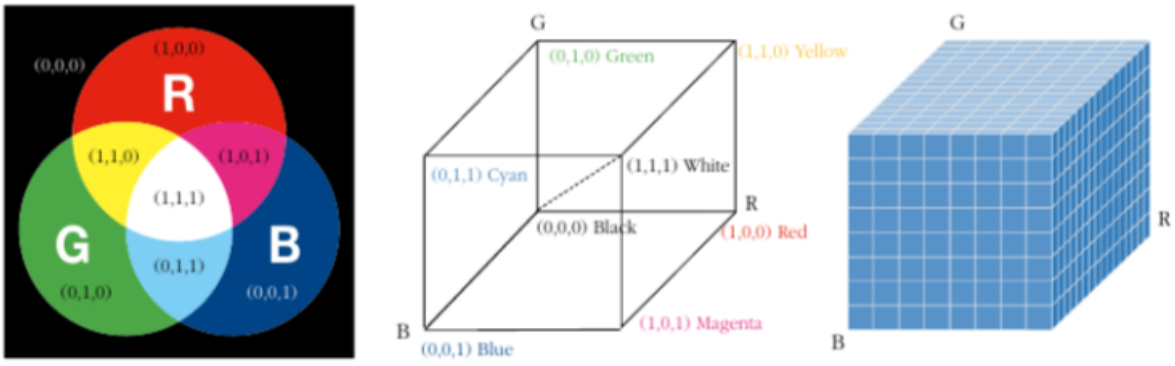
의료에서 사용되는 MR, CT도 동일

물체까지 거리를 측정한 영상

e. **RGB-D영상**(RGB 컬러 센서와 깊이 센서가 통합된 카메라로 획득)

f. 라이다라는 깊이센서로 확인했을 시, 어떤 조건을 만족하는 점의 거리만 측정하기 때문에, 완벽한 격자 구조가 아니기에 획득한 거리 데이터를 보여주는 **점구름영상**

3.1.3 컬러 모델



(a) RGB 삼원색의 혼합 (b) RGB 큐브 (c) 양자화된 RGB 큐브

그림 3-6 RGB 컬러 공간

a. RGB 삼원색을 사용하면 모든 색 표현 가능

b. 세 요소의 범위를 0-1로 설정해 세상의 모든 색을 큐브에 삽입

   R (1,0,0) G (0,1,0) B (0,0,1)

c. b인 RGB 큐브를 디지털 영상으로 표현할 때 각 요소를 양자화

3.1.4 RGB 채널별로 디스플레이

np가 지원하는 **ndarray 클래스**로 표현

RGB 영상의 일부를 잘라내고 R G B 채널로 분리하기 위해 numpy 배열 다루는 연습

```
import cv2 as cv
import sys

img = cv.imread("glory.jpg") # 영상읽기

if img is None:
  sys.exit("파일을 찾을 수 없습니다.")

# 세 채널을 가진 원래 이미지 img 디스플레이
cv.imshow("original_RGB",img)
# img 왼쪽 위 컷컷
cv.imshow("Upper left half",img[0:img.shape[0]//2,0:img.shape[1]//2,:])
# img 첫번째와 두번째 축을 1/4~3/4 지정해 영상의 중간 부분 컷
cv.imshow("Center half",img[img.shape[0]//4,3*img.shape[0]//4,img.shape[1]//4:3*img.shape[1]//4,:])

cv.imshow("R channel",img[:,:,2])
cv.imshow("R channel",img[:,:,1])
cv.imshow("R channel",img[:,:,0])

cv.waitKey()
cv.destroyAllWindows()
```

3.2 이진 영상

이진영상 : 0 또는 1의 흑백 영상

→ 프로그래밍 편리성 우선시

3.2.1 이진화

$$b(j,i) = \begin{cases} 1, f(j,i) \geq T \\ 0, f(j,i) < T \end{cases} \qquad (3.1)$$

f 는 원래 명암 영상 b는 이진영상으로 위 식은 이진화하는 함수

임계값 T의 결정이 중요

- 너무 낮게 또는 높게 설정 시, 대부부분 화소가 물체또는 배경에 쏠리는 문제 발생

  → 해결 : 히스토그램의 계곡 근처를 임곗값으로 결정해 쏠림현상 누구러뜨림

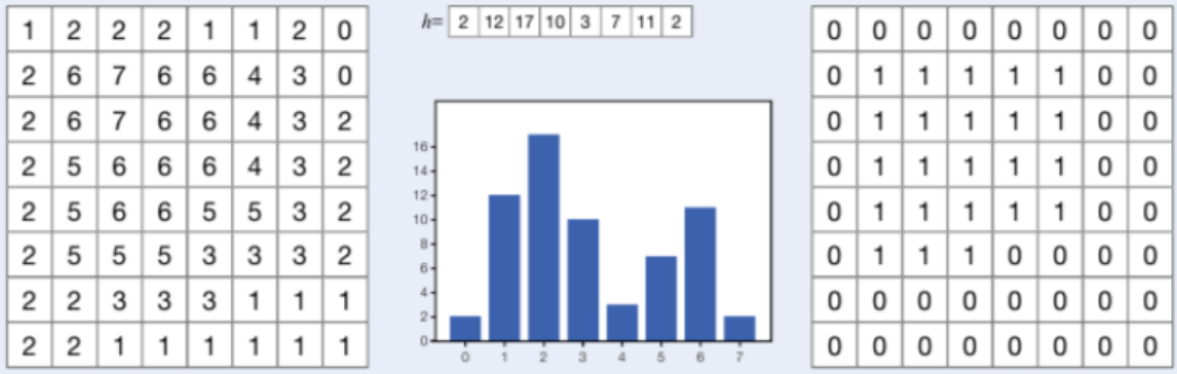


그림 3-9 히스토그램을 이용한 이진화

임계값 T = 4로 계곡이 발생해 이진화에 따라 다음과 같이 이진영상 표현

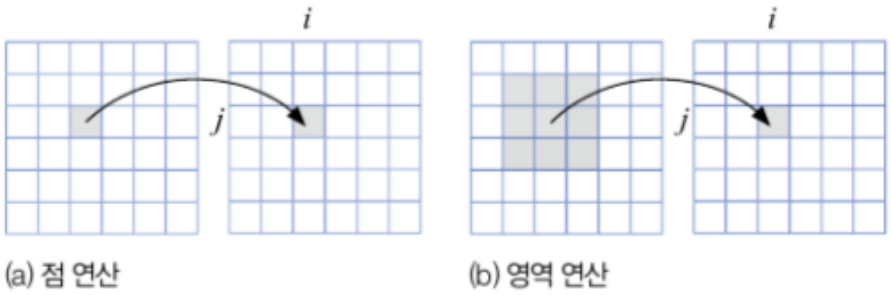
```
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread("glory.jpg") # 영상읽기
h = cv.calcHist([img],[2],None,[256],[0,256]) # 2번 채널에서 히스토그램
plt.plot(h,color='r',linewidth=1)
```

실행 결과를 확인해보면 큰 계곡3, 작은 계곡 아주 많음

→ 계곡 값으로 정하기 어려움

3.2.2 오츄 알고리즘

점연산 해당 ↔ 모폴로지 : 영역 연산



(a) 점 연산 (b) 영역 연산

오츄알고리즘은 위에서 본 계곡값으로 임곗값을 정하는 것에 어려움이 있어 **이진화 최적화**로 가능

$$\hat{t} = \underset{t \in \{0,1,2,\cdots,L-1\}}{\operatorname{argmin}} J(t) \qquad (3.2)$$

- J 목적함수
  - t의 좋은 정도를 측정하는 데 작을수록 좋음
- 매개 변수 t : 해 공간 구성
- t 햇 : J가 최소인 명암값을 최적값

→ t 햇을 임계값으로 사용해 이진화

```
import cv2 as cv
import sys

img = cv.imread("glory.jpg") # 영상읽기

t,bin_img = cv.threshold(img[:,:,2],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
print("오츄 알고리즘이 찾은 최적 임계값=", t)

cv.imshow("R channel",img[:,:,2]) # R 채널 영상
cv.imshow("R channel binarization",bin_img) # R 채널 이진화 영상

cv.waitKey()
cv.destroyAllWindows()
```

cv.THRESH_BINARY+cv.THRESH_OTSU : 오츄 알고리즘으로 이진화 수행

threshold 함수는 알고리즘이 찾은 최적의 임계값과 이진화된 영상 반환

[:, :, 2] = [B, G, R] 이기 때문에 위의 결과는 R채널로 이진화

+) 오츄알고리즘 말고도 낱낱 탐색 알고리즘(모든 해 검사), 탐욕 알고리즘, 역전파 알고리즘 존재

3.2.3 연결 요소

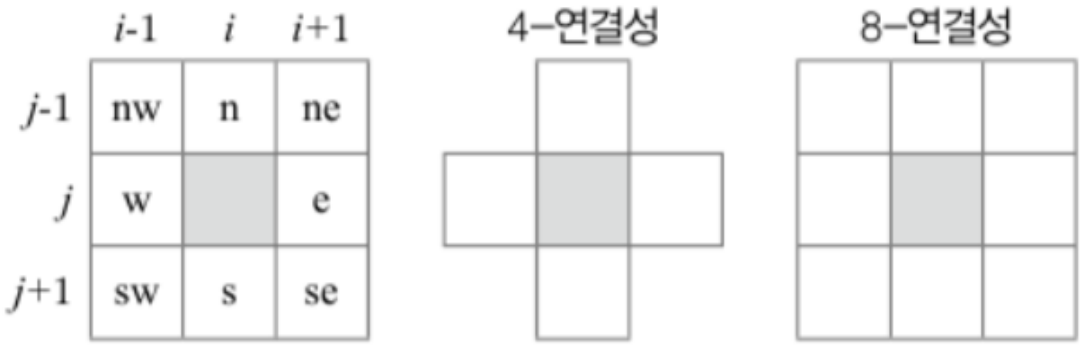


그림 3-10 화소의 연결성

연결 요소 : 1의 값을 진 연결된 화소의 집합 → opencv에서 connectedComponents 함수로 탐색

- 고유한 정수 번호 구분

[예시 3-2] 연결 요소

[그림 3-11]에서 (a)는 입력 이진 영상이고, (b)와 (c)는 각각 4-연결성과 8-연결성으로 찾은 연결 요소다. 연결 요소는 고유한 정수 번호로 구분한다.

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

(a) 입력 이진 영상

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	0	2	2	0	0
0	1	1	0	2	2	0	0
0	0	0	0	2	2	0	0
0	0	0	0	0	0	0	0
0	0	3	3	3	0	4	0
0	0	0	3	3	0	4	0

(b) 4-연결성으로 찾은 연결 요소

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	0	1	1	0	0
0	1	1	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0
0	0	2	2	2	0	3	0
0	0	0	2	2	0	3	0

(c) 8-연결성으로 찾은 연결 요소

그림 3-11 연결 요소 찾기

이해가..

3.2.4 모폴로지

목적 : 영상을 변환하는 과정에서 하나의 물체가 **여러 영역으로 분리**되거나 다른 물체가 **한 영역으로 붙는 현상** 발생을 최소화하기 위함

모폴로지 : 구조요소를 이용해 영역의 모양 조작

모폴로지는 이진과 명암으로 나뉜다

- 회색이 중심화

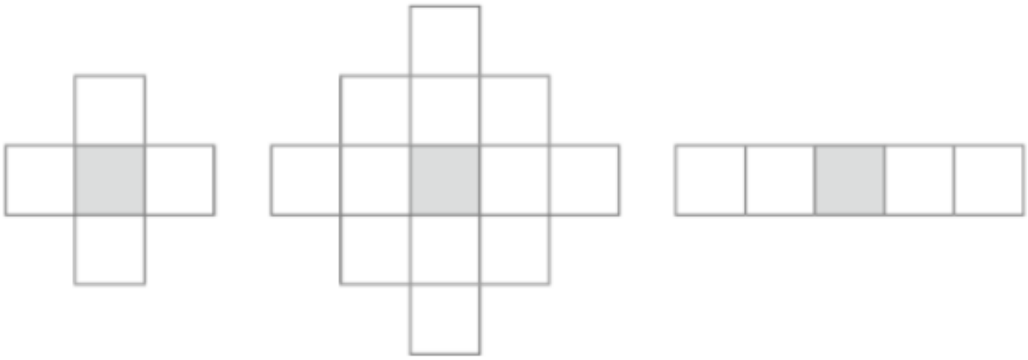
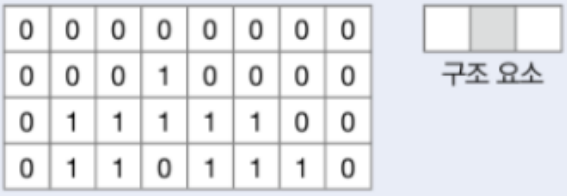


그림 3-12 모폴로지가 사용하는 구조 요소

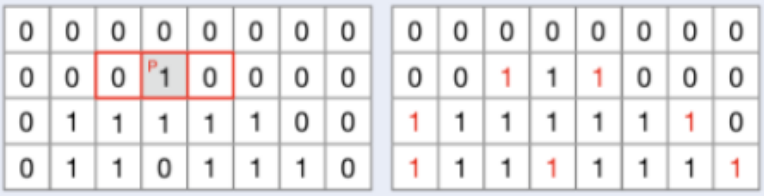
구조 요소 모양에 달려 있는 팽창과 침식

- 팽창 : 구조 요소의 중심을 1인 화소에 씌운 다음 구조 요소에 해당하는 모든 화소 1로 변경
- 침식 : 구조 요소의 중심을 1인 화소 p에 씌운 다음
  - 구조 요소에 해당하는 모든 화소가 1에 해당 O : p를 1로 유지
  - 구조 요소에 해당하는 모든 화소가 1에 해당 X : p = 0
- 닫힘 : 침식 결과 + 팽창 적용
- 닫힘 : 팽창 결과 + 침식 적용

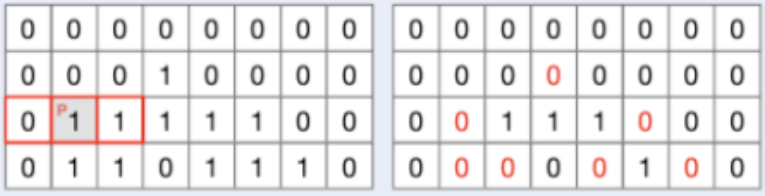
[그림 3-13]에서 (a)는 입력 이진 영상과 구조 요소를 보여준다. (b)는 팽창을 설명하는데, 왼쪽은 p라고 표시된 (1,3) 위치에 구조 요소를 씌운 상황이다. 구조 요소에 해당하는 왼쪽과 오른쪽 이웃 화소가 1로 바뀐다. 오른쪽은 팽창을 마친 결과 영상이다. 1로 바뀐 화소를 빨간색으로 표시해 이해를 돕는다. (c)는 침식을 설명하는데, 왼쪽은 p라고 표시된 (2,1) 위치에 구조 요소를 씌운 상황이다. 구조 요소에 해당하는 점에 1이 아닌 화소가 있어 p는 0으로 바뀐다. 오른쪽은 침식을 마친 결과 영상이다. 0으로 바뀐 화소를 빨간색으로 표시해 이해를 돕는다.



(a) 입력 영상과 구조 요소



(b) 팽창



(c) 침식

모폴로지 연산

히스토그램부터 모폴로지까지 - NiklasJang's Blog

- 점 연산 : 화소가 자신의 값만을 보고 새로운 값을 정한다.
- 영역 연산 : 화소가 이웃에 있는 몇 개의 화소들을 보고 새로운 값을 정한다.
- 기하 연산 : 일정한 기하하적 규칙에 따라 다른 곳에 있는 값을 취한다.

3.3 점 연산

오츄이진화를 제외한 다른 점연산 소개

3.3.1 명암 조절

수식을 이용해 영상을 밝게 어둡게 조정 가능하게 하는 선형연산

$$f'(j,i)=\begin{cases}\min\left(f(j,i)+a, L-1\right) & \text{밝게}\\ \max\left(f(j,i)-a, 0\right) & \text{어둡게}\\ (L-1)-f(j,i) & \text{반전}\end{cases}$$

1. 원래 영상에 양수를 더해 밝게
2. 원래 영상에 양수를 더해 어둡게
3. 원래 명암값을 빼서 반전

감마조정

명암 10 → 20 과 100 → 110 같은 만큼 향상했지만 인간이 느끼는 정도가 다름

위에 대해 **비선형적인 시각 반응을 수학적으로 표현**

$$f'(j,i)=(L-1)\times \dot{f}(j,i)^{\gamma}$$

정규화환 영상

감마는 사용자가 조정하는 값

1이면 원래 영상 유지, 1보다 작으면 밝음, 크면 어두움

```
def gamma(f, gamma=1.0): # (감마 보정 영상, 감마=기본값)
  f11 = f/255.0 # L = 256 가정 -> 정규화 과정
  return np.unit8(255*(f1**gamma))

# 영상 종류별로 확인
gc = np.hstack((gamma(img,0.5),gamma(img,0.75),gamma(img,1.0),gamma(img,2.0),gamma(img,3.0)))
cv.imshow('gamma',gc)
```

3.3.2 히스토그램 평활화

히스토그램이 평평하게 되도록 영상을 조작해 영상의 명암 대비를 높이는 기법
→ 명암대비 높아지면 물체 더 잘 식별

$$l' = \text{round}\left(\ddot{h}(l) \times (L-1)\right)$$

- 정규화 히스토그램 h 점 :모든 칸의 값을 합해 1.0이 되는 정규화 히스토그램
- 누적 정규화 히스토그램 h 점점 : i번째 칸은 0부터 i까지 합한 값을 가진 것
- I : 원래 명암값
- I' : 평활화로 얻은 새로운 명암값

3.4 영역 연산

3.4.1 컨볼루션 (convolution)

컨볼루션 : 입력 영상 f의 각 화소에 필터를 적용해 곱의 합 구하는 연산

입력영상 f 와 필터 u 의 컨볼류션의 결과 출력영상 f'

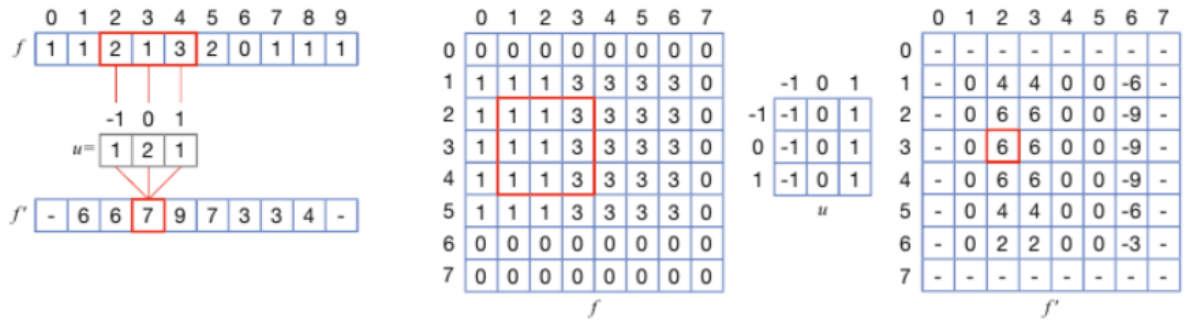
→ 필터의 픽셀 value의 index는 -1 0 1

$$f'(x) = \sum_{i=-(w-1)/2}^{(w-1)/2} u(i) f(x+i)$$

1차원 필터 적용

$$f'(y,x) = \sum_{j=-(h-1)/2}^{(h-1)/2} \sum_{i=-(w-1)/2}^{(w-1)/2} u(j,i) f(y+j, x+i)$$

2차원 필터 적용



(a) 1차원 영상에 컨볼루션 적용 (b) 2차원 영상에 컨볼루션 적용

그림 3-16 컨볼루션의 원리

- 적용결과, 필터를 가장자리 화소에 적용하면 일부가 밖으로 나가, 적용 어려움

  → f'의 가장자리를 -로 표시

- 결과는 별도의 출력 영상 f' 에 기록

3.4.2 다양한 필터

특정한 목적없는 일반 연산이며 **필터 결정시 목적 결정**

스무딩 필터

영상에 존재하는 다양한 잡음은 밝은 영역에 어두운 작은 반점이 여기 저기 나타남

스무딩 필터로 컨볼루션하면 이를 저하할 수 있음

↔ 단점

부작용으로 물체의 경계를 흐릿하게 만드는 **블러링**

샤프닝 필터

스무딩과 반대로, 엣지를 선명하게 해 물체의 식별 도움

↔ 단점

잡음 확대

3.5 기하 연산

앞에서 다룬 연산들은 자기 자신 혹은 이웃을 보고 값을 정했으나, 이번엔 머리있는 화소에서 값을 가져올 수 있는 연산

→ 영상의 크기를 조절하거나 영상회전 가능

3.5.1 동차 좌표와 동차 행렬

점 (x,y)는 동차 점(xt, yt, t)가 존재하는 데 t는 임의의 숫자

- 2차원 점의 위치에 1을 임의로 추가해 3차원 벡터로 표현한 것 = (x,y,1)

  → 변환이 자유로워 사용하는 듯?
- 동차 좌표에서는 3개 요소에 같은 값을 곱하면 같은 좌표를 나타낸다

  (-3, 6,1), (-2,4,1) (-1,2,5,0) 모두 점 (-2,4)에 해당

  → 따라서 네개의 점이 모두 점 (-2,4)의 가능한 동차 좌표들

기하 변환	동차 행렬	설명
이동	$\mathrm{T}(t_x,t_y)=\begin{pmatrix}1&0&t_x\\0&1&t_y\\0&0&1\end{pmatrix}$	x 방향으로 t_x, y 방향으로 t_y만큼 이동
회전	$\mathrm{R}(\theta)=\begin{pmatrix}\cos\theta&\sin\theta&0\\-\sin\theta&\cos\theta&0\\0&0&1\end{pmatrix}$	원점을 중심으로 반시계 방향으로 θ만큼 회전
크기	$\mathrm{S}(s_x,s_y)=\begin{pmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{pmatrix}$	x 방향으로 s_x, y 방향으로 s_y만큼 크기 조정(1보다 크면 확대, 1보다 작으면 축소)

수학표기가 깔끔하며 계산 효율이 좋음

→ 행렬 계산 을 할때 큰 이점

3.5.2 영상의 기하 변환

앞서 배운 동차 변환을 적용 가능 ; 회전하거나 크기를 조정할 수 있음

원래 영상을 동차 행렬을 이용해 새로운 영상으로 변환하는 과정에서 에러 발생

에일리어싱(aliasing) :

실수 좌표를 정수로 반올림할 시, 결측값으로 표시된 화소처럼 값을 받지 못하는 경우 발생

방지하기 위해, 원래 영상의 해당화소를 찾는 후방변환 사용(동차행렬 A의 역행렬)

↔ 안티 에일리어싱 : 에일리어싱 저하

3.5.3영상 보간

실수 좌표를 정수로 변환하는 과정 필요

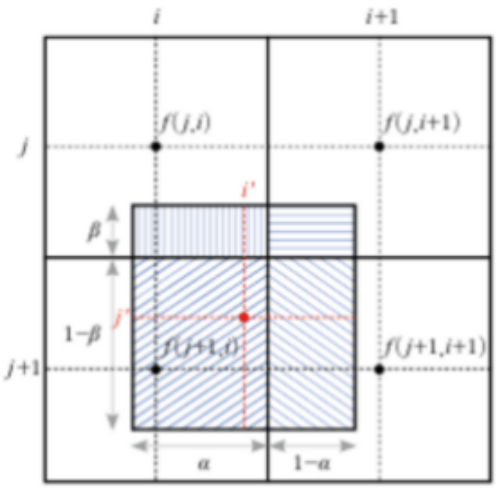


그림 3-22 실수 좌표의 화솟값을 보간하는 과정

최근접 이웃방법(=영상보간) : 반올림을 사용해 가장 가까운 화소에 배정하는 기법

- 겹치는 비율을 곱하기 때문에 선형 보간법에 해당

3.6 OpenCV의 시간 효율