영상처리 기초

Computer Vision and Pattern Recognition

Byeongjoon Noh

powernoh@sch.ac.kr

SCH SOON CHUN HYANG UNIVERSITY

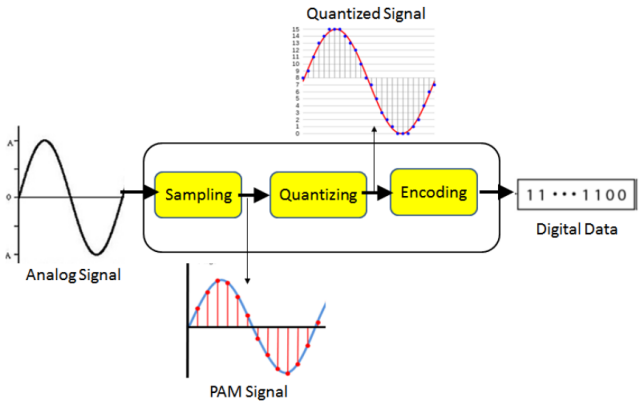
Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

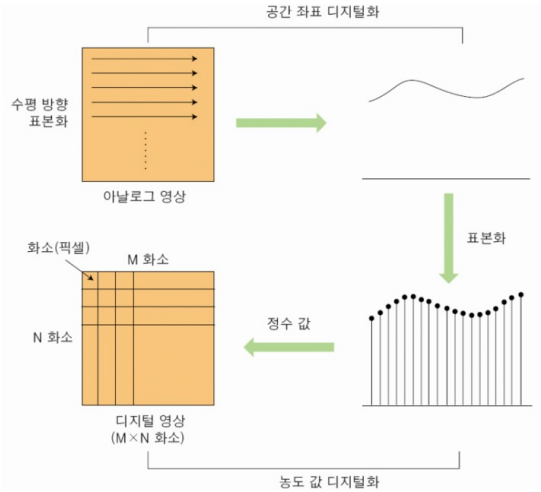
Contents

1. 디지털 영상 기초

디지털 영상 획득과 표현

- 영상 획득 과정은 매우 복잡
  - 아날로그 영상 ➔ 디지털 영상





- 디지털 (영상) 변환
  - 1) M*N 영상으로 샘플링(sampling)
  - 2) L 단계로 양자화(quantization)
  - 3) 변환/압축을 통한 부호화(encoding)

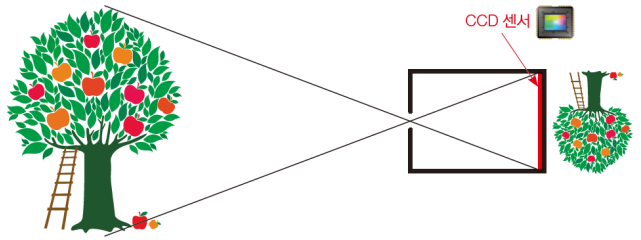


그림 3-2 핀홀 카메라 모델과 CCD 센서

- (참고)해상도 (resolution) = 화소의 개수
  - 물리적 단위 공간에서 식별 가능한 점의 개수
  - dpi (dot per inch) = 인치 당 점의 수

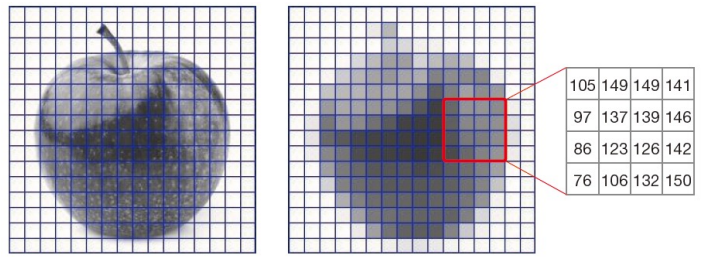
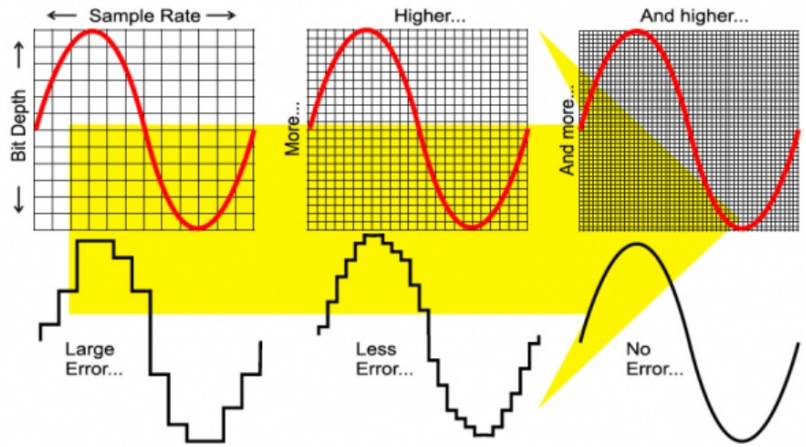


그림 3-3 피사체가 반사하는 빛 신호를 샘플링과 양자화를 통해 디지털 영상으로 변환

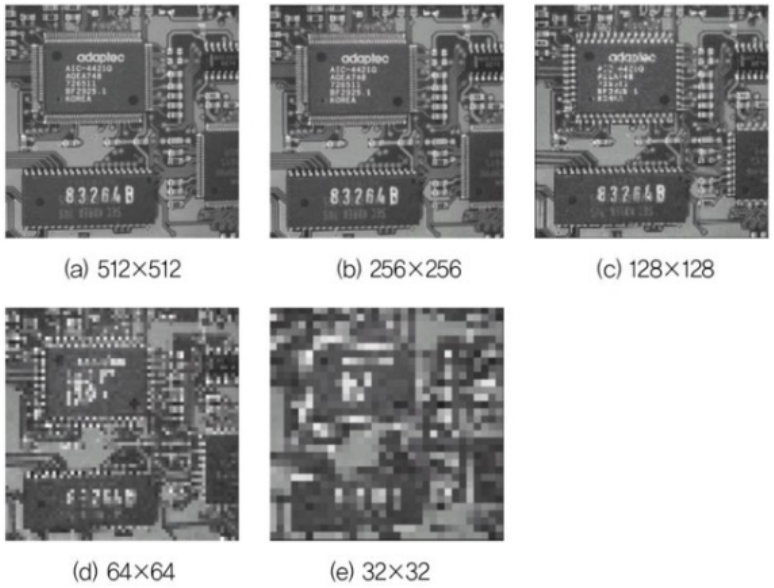
디지털 영상 변환

- 샘플링
  - 아날로그 신호를 일정 간격으로 표본화 하여
    PAM (Pulse Amplitude Modulation) 신호를 획득함. (데이터의 크기 결정)
  - Sampling rate ↑ ➔ Data size ↑
  - 일반적인 sampling rate = 44.1 KHz
  - Nyquist Theorem:아날로그 신호의 최고 주파수 2배 크기로 sampling 진행 ➔ 디지털 신호를 원래의 아날로그 신호의 손실 없이 복원 할 수 있음



디지털 영상 변환

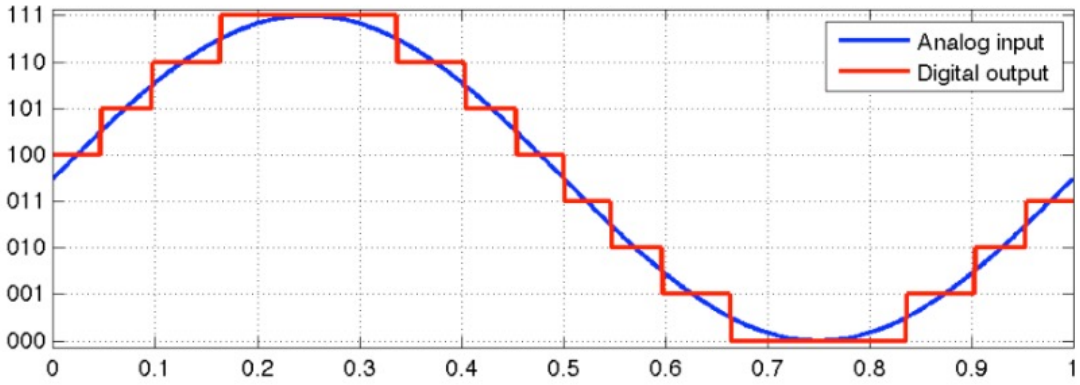
- 디지털 영상에서의 샘플링
  - 아날로그 영상에서 공간적, 시간적으로 연속되는 밝기 강도 (Intensity)에 따라 이산적인 점 (Pixel) 을 추출하는 과정
  - Sampling rate ↑ ➔ High resolution (Huge size…)



디지털 영상 변환

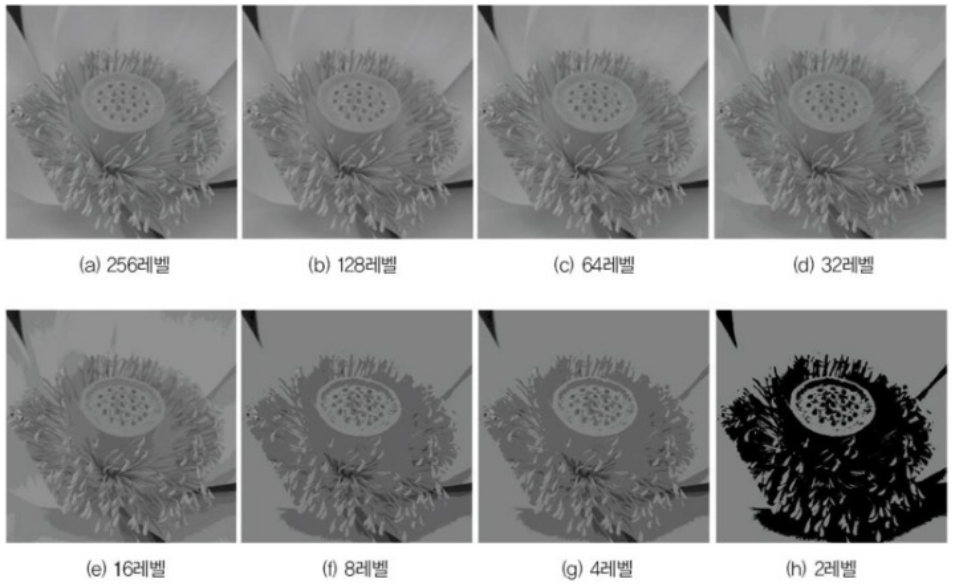
- 양자화
  - Sampling된 값도 결국 아날로그의 형태

    ➔ 디지털 신호로 완전한 변환이 필요함
  - 각 아날로그 값들을 맵핑 (Mapping)하여 변환
    - Quantization noise 발생



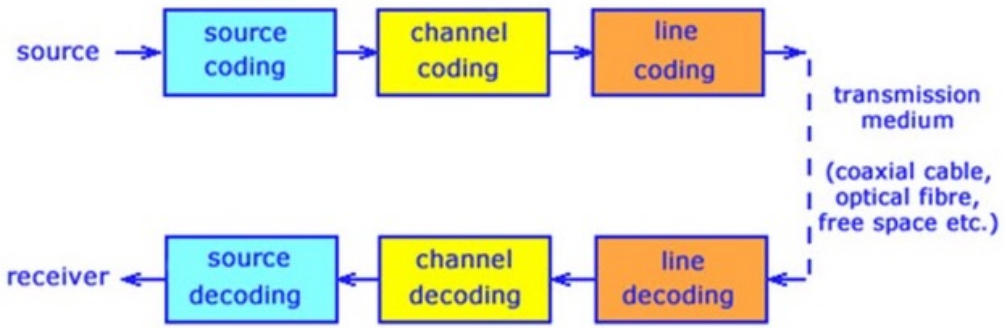
디지털 영상 변환

- 디지털 영상에서의 양자화
  - Sampling된 영상의 각 화소의 밝기나 색을 정해진 몇 단계의 값으로 근사화
  - ➔ 각 화소의 밝기나 색이 숫자로 표현
  - ➔ 각 화소에 양자화된 표본 값을 획득
  - Gray level 해상도(진폭)를 결정
    - Quantization ↑ ➔ 표현할 수 있는 색상 ↑



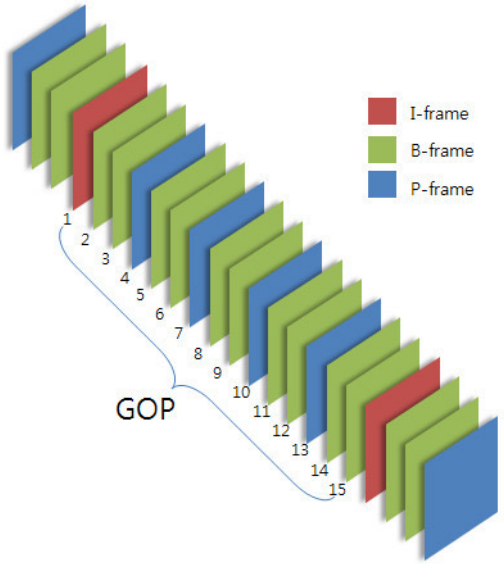
디지털 영상 변환

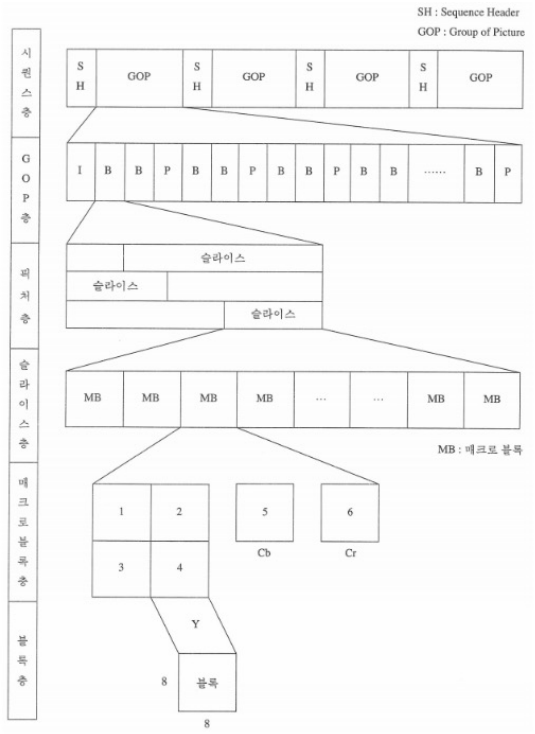
- 부호화
  - 양자화된 값을 이진 데이터로 변환하는 과정
    - 단위 시간당 가능한 많은 데이터를 처리할 수 있는 방법을 찾고자 고안됨
  - 신호처리: 소스 부호화, 채널 부호화, 라인 부호화 등
  - 복호화 (Decoding) 함께 수행됨



디지털 영상 변환

- 디지털 영상에서의 부호화
  - 양자화된 화소의 밝기나 색 데이터를 2진수로 표현하는 과정
  - 압축 부호화를 수행함
    - MPEG, JPEG, …





영상 좌표계

- 디지털 영상의 좌표계
  - 왼쪽 구석이 원점
  - (y, x) 표기 ➔ $X(j,i), X(y,x), f(j,i)$
  - 함수에 따라 (x, y)표기를 사용하니 주의 필요
    - ex) cv2.line()

```
cv.line(img,(10,20),(100,20),….)
```

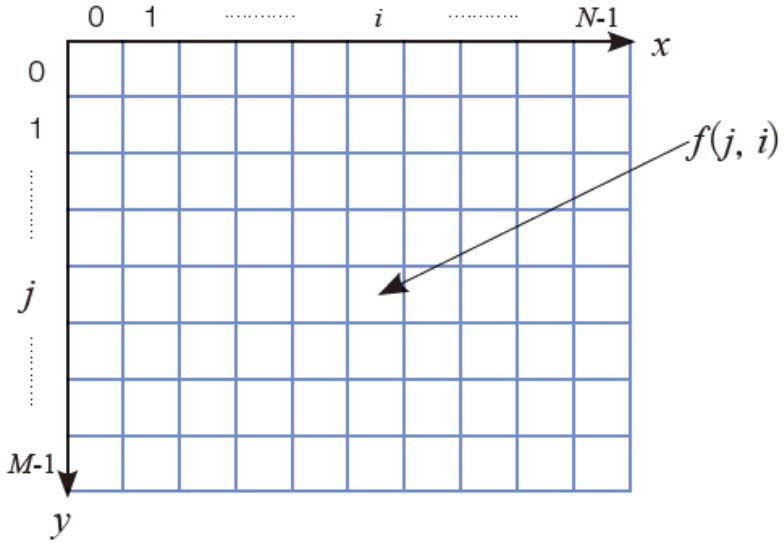
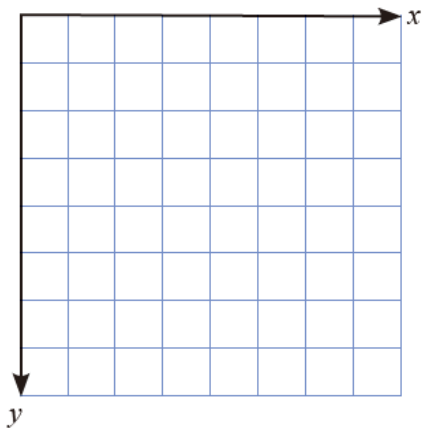


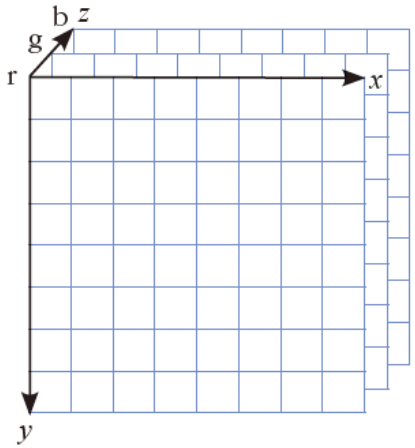
그림 3-4 디지털 영상의 좌표계

- OpenCV 는 numpy.ndarray로 영상 표현
  - numpy.ndarray가 지원하는 다양한 함수를 사용할 수 있다는 큰 장점
  - ex) min, max, argmin, argmax, mean, sort, reshape, transpose, etc.

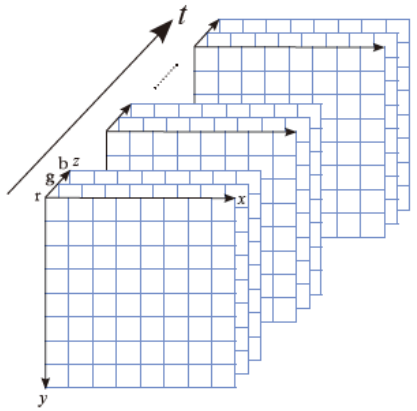
다양한 종류의 영상



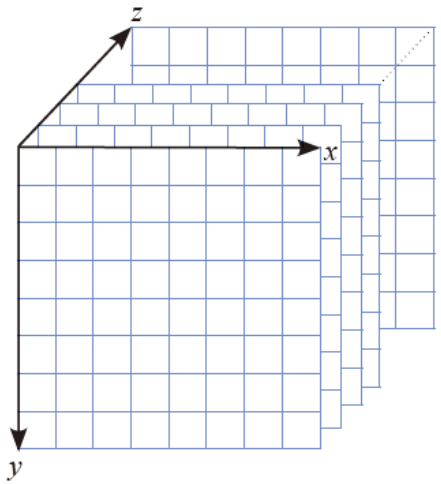
(a) 명암 영상



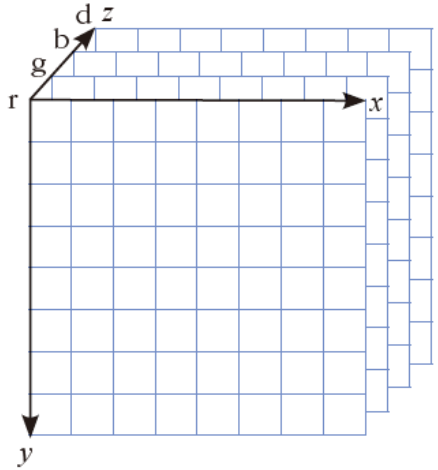
(b) 컬러 영상



(c) 컬러 동영상



(d) 다분광/초분광/MR/CT 영상



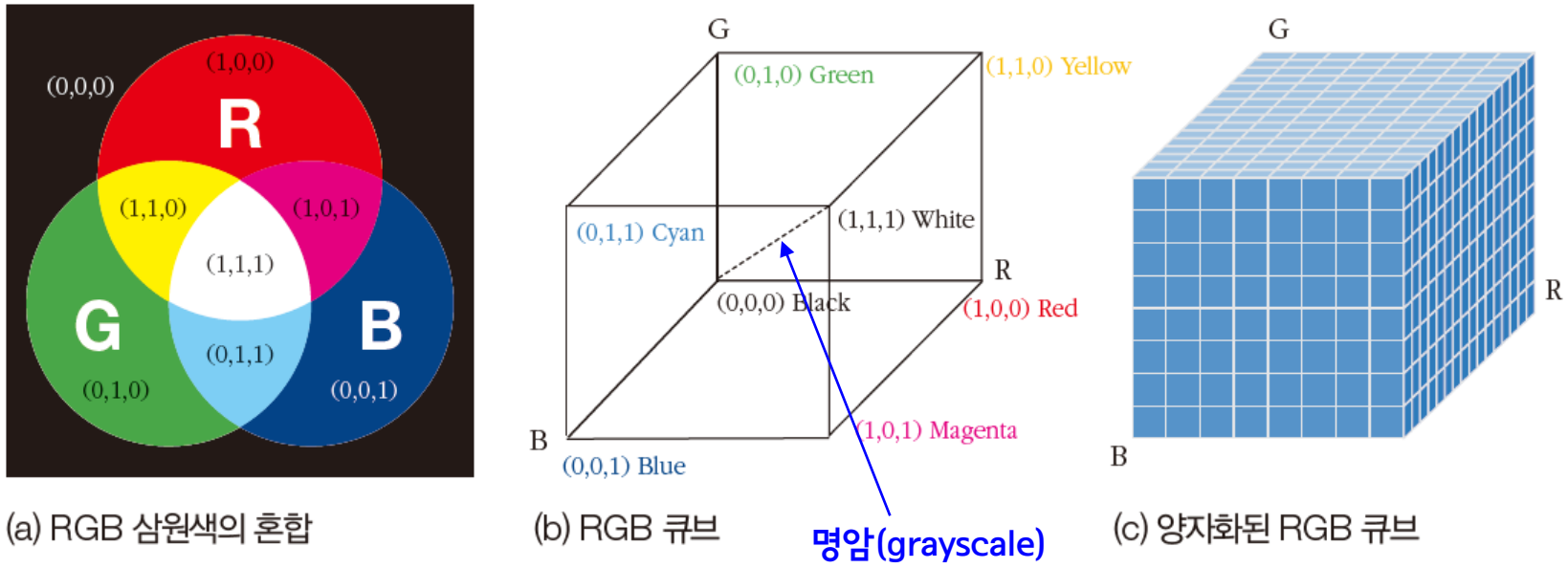
(e) RGB-D 영상

(f) 점 구름 영상

그림 3-5 다양한 형태의 디지털 영상

컬러 모델

- RGB 컬러 모델
  - $f_c(x,y) = f_r(x,y), f_g(x,y), f_b(x,y)$



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

그림 3-6 RGB 컬러 공간

컬러 모델

- HSV 컬러 모델
  - 빛의 밝기를 고려한 컬러 모델
    - ➔ RGB 보다 빛 변환에 강건

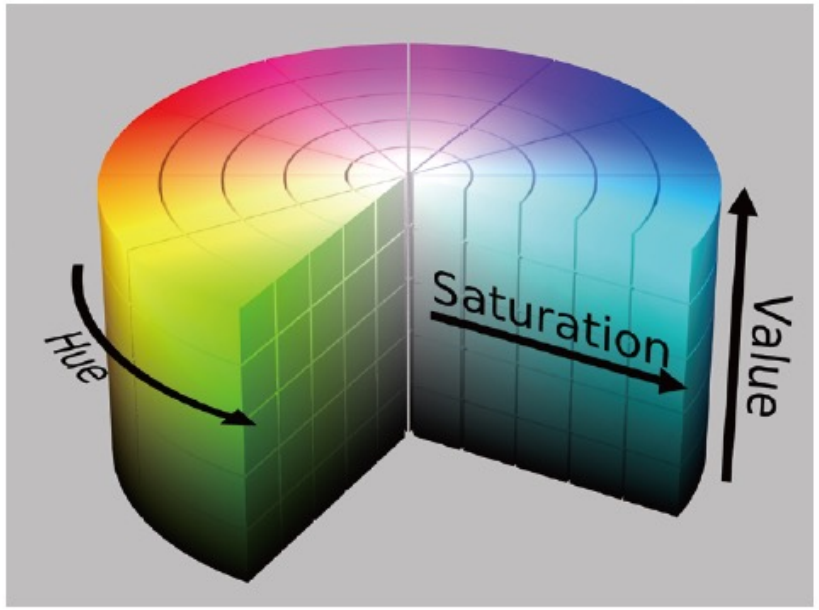
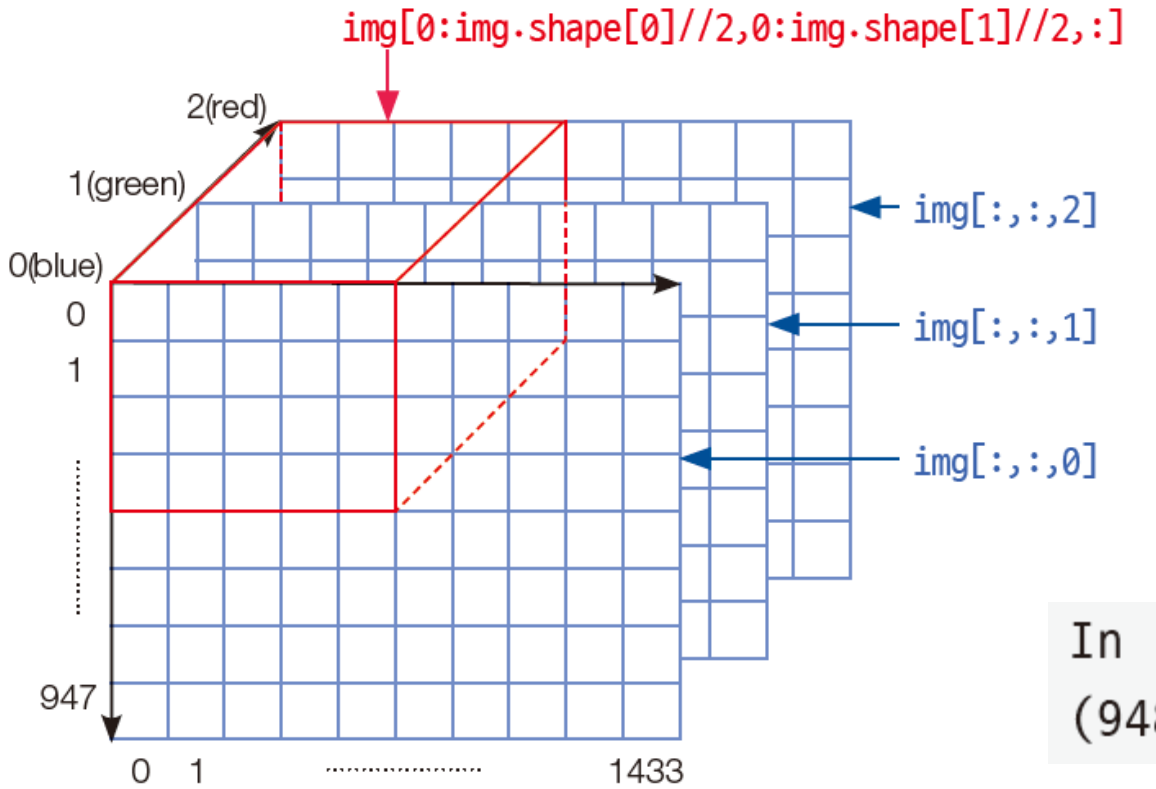


그림 3-7 HSV 컬러 모델

컬러 모델

- RGB 채널별 디스플레이
  - numpy의 슬라이싱 기능을 이용하여 RGB 채널별 정보 추출 가능




```
In [0]: img.shape
(948,1434,3)
```

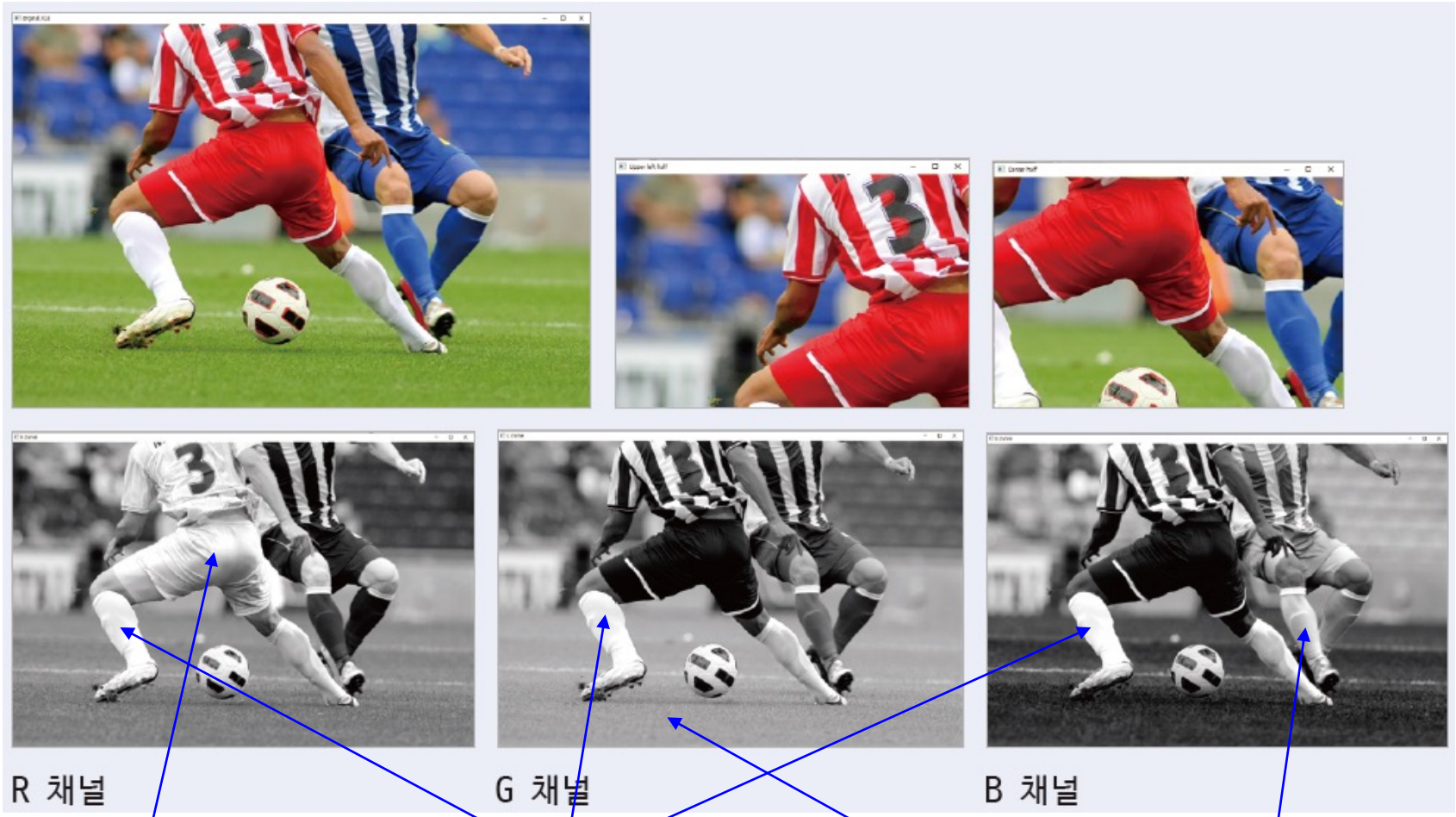
그림 3-8 numpy.ndarray의 슬라이싱을 이용한 영상 일부분 자르기([프로그램 3-1]의 10행)

프로그래밍 실습: RGB 채널별 디스플레이

프로그램 3-1 RGB 컬러 영상을 채널별로 구분해 디스플레이하기

```
01  import cv2 as cv
02  import sys
03
04  img=cv.imread('soccer.jpg')
05
06  if img is None:
07      sys.exit('파일을 찾을 수 없습니다.')
08
09  cv.imshow('original_RGB',img)
10  cv.imshow('Upper left half',img[0:img.shape[0]//2,0:img.shape[1]//2,:])
11  cv.imshow('Center half',img[img.shape[0]//4:3*img.shape[0]//4,img.
              shape[1]//4:3*img.shape[1]//4,:])
12
13  cv.imshow('R channel',img[:,:,2])
14  cv.imshow('G channel',img[:,:,1])
15  cv.imshow('B channel',img[:,:,0])
16
17  cv.waitKey()
18  cv.destroyAllWindows()
```

프로그래밍 실습: RGB 채널별 디스플레이



빨간 유니폼 영역이 밝음

흰색 양말 영역은 모두 밝음

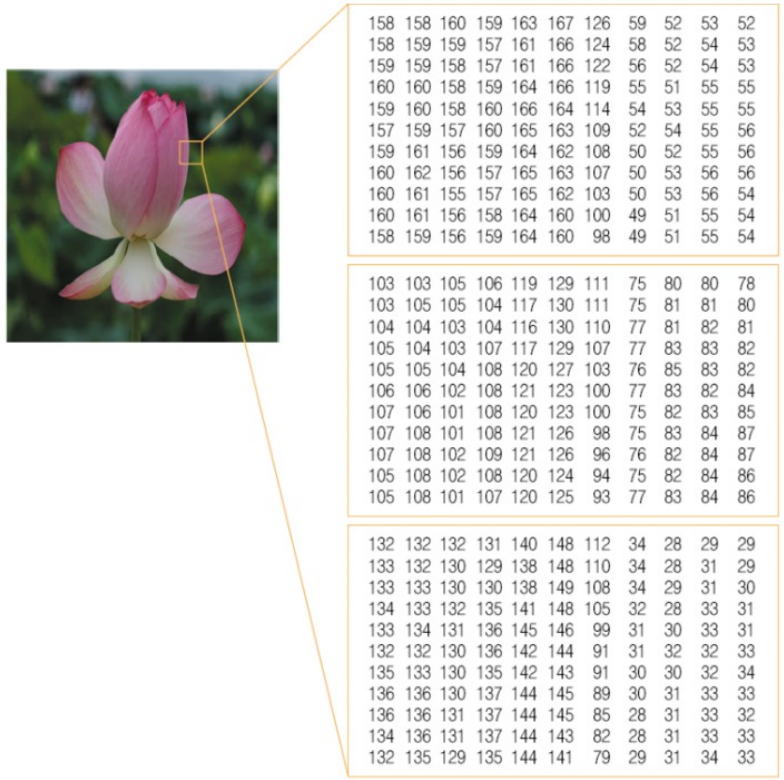
녹색 잔디 영역이 밝음

파란 양말 영역이 밝음

2. 이진 영상

디지털 영상의 종류

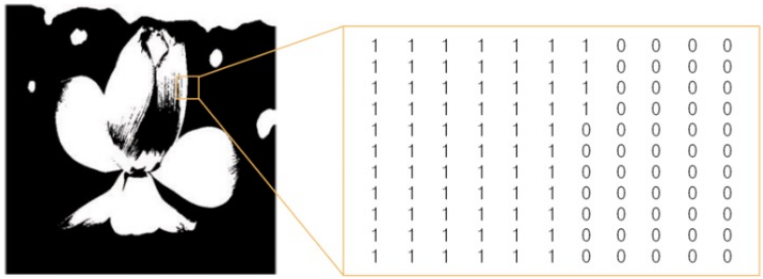
- 컬러 영상
  - 실제로 눈에 보이는 모습과 유사한 밝기와 색상을 가진 영상
  - 빛의 3원색을 이용
  - 각 색을 독립적인 형태로 처리하고 합침
  - 각 색의 상호작용이 큼 ➔ 처리 어려움



디지털 영상의 종류

- Gray-level 영상
  - Binary image보다 조금 더 밝게 표현됨 (bit 수 ↑)
  - 화소의 밝기가 여러 단계 (흑백 사진)
  - 밝기의 단계: 검정색~회색~흰색 (단계 수는 quantization bit 수가 결정)

디지털 영상의 종류



- 이진 영상
  - 화소가 0(흑) 또는 1(백)인 영상
    - 흑/백으로만 구성 ➔ 처리 속도 빠름
  - Quantization 진행 시 양자화 bit 수를 1비트로 지정
    - 편의상 1바이트 사용하는 경우 많음
  - 지문, 팩스, 문자 영상, 에지 검출 결과를 표시 등 물체와 배경을 구분하여 표시하는 응용 등에 사용

- 이진화 전략
  - 임계값 T 보다 큰 화소는 1, 그렇지 않은 화소는 0으로 변환 ➔ 임계값 결정이 중요

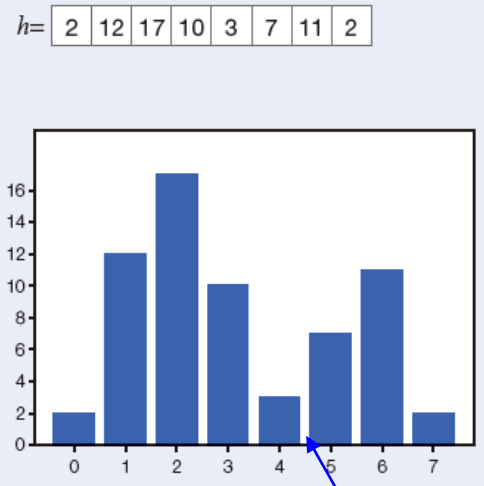
$$b(j,i)=\begin{cases}1, f(j,i)\geq T\\ 0, f(j,i)<T\end{cases} \qquad (3.1)$$

이진화 알고리즘

- **히스토그램 기반 이진화 알고리즘**

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

(a) 입력 영상



(b) 히스토그램

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

(c) 이진 영상

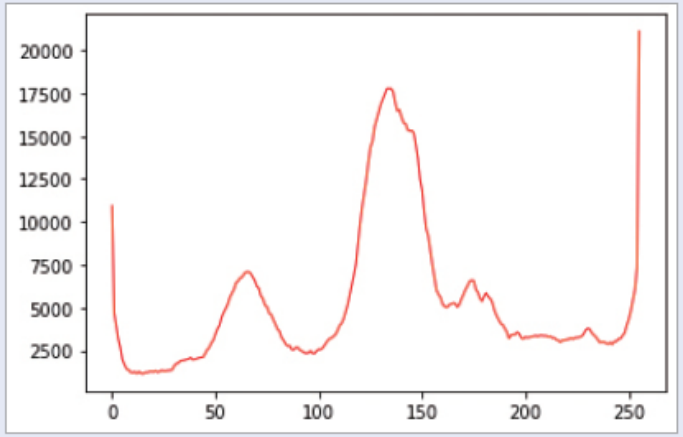
그림 3-9 히스토그램을 이용한 이진화

이진화 알고리즘

- 히스토그램 기반 이진화 알고리즘
  - 실제 영상에서는 계곡이 아주 많이 나타나서 구현이 쉽지 않음

프로그램 3-2 실제 영상에서 히스토그램 구하기

```
01  import cv2 as cv
02  import matplotlib.pyplot as plt
03
04  img=cv.imread('soccer.jpg')
05  h=cv.calcHist([img],[2],None,[256],[0,256]) # 2번 채널인 R 채널에서 히스토그램 구함
06  plt.plot(h,color='r',linewidth=1)
```



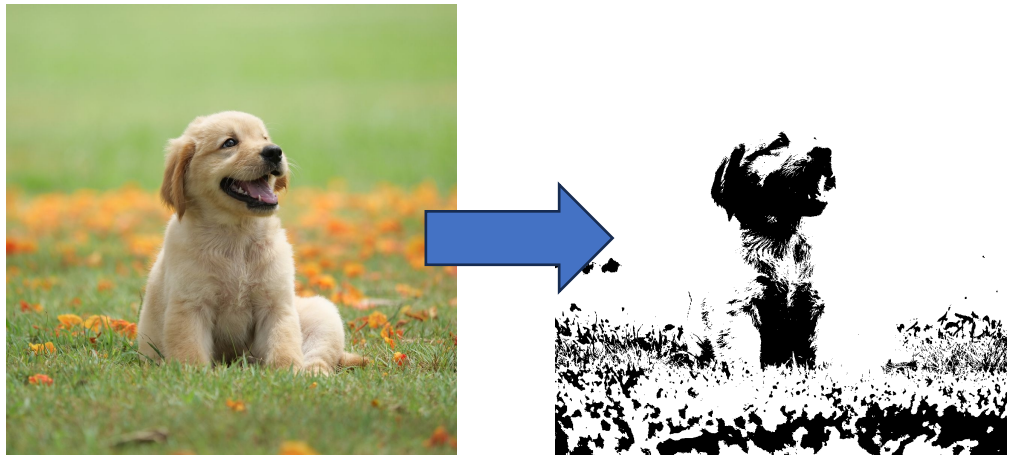
이진화 알고리즘

- Otsu 알고리즘
  - idea: 이진화 시 흑/백 그룹이 각각 균일할 수록 좋다
    - 균일성 ➔ 분산으로 측정 ➔ 즉, 분산이 작을수록 균일성이 높음

  - 이미지 내 전체 픽셀 값을 두 그룹으로 분할하여 가장 작은 분산을 갖는 임계값을 찾는 과정
    - 이미지 내 어떤 픽셀 값(T)을 기준으로 이진화했을 때 가장 좋을까?

Recall

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



이진화 알고리즘

- Otsu 알고리즘
  - 이진화를 최적화 문제로 바라봄
  - 최적값 $\hat{t}$을 임계값 T로 활용

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

  - 목적함수 $J(t)$는 임계값 t의 좋은 정도를 측정함 (작을수록 좋음)

$$J(t) = n_0(t)v_0(t) + n_1(t)v_1(t) \qquad (3.3)$$

    - $v_0(t)$: t로 이진화했을 때 0이 되는 화소들의 분산
    - $v_1(t)$: t로 이진화했을 때 1이 되는 화소들의 분산
    - $J(t)$: $v_0(t)$과 $v_1(t)$의 가중치 ($n_0(t)$, $n_1(t)$)합

이진화 알고리즘

- Otsu 알고리즘 예제 – 최적 임계값 $\hat{t}(=T)$를 찾는 과정

화소 값	0	1	2	3	4	5	6	7
빈도	1	2	1	3	2	2	2	2

  - $t = 2$라고 가정해보자
    - 그룹 C_0, C_1에 속하는 픽셀 값: $C_0 = \{0, 1, 2\},\ C_1 = \{3, 4, 5, 6, 7\}$
    - 가중치 $n_0 = 4(= 1 + 2 + 1),\ n_1 = 11(= 3 + 2 + 2 + 2 + 2)$
    - $v_0(t), v_1(t)$ (분산) 계산
      - $v_i(t) = \frac{1}{n_i(t)} \sum_{x \in C_i} \left(x - \mu_i(t)\right)^2$
      - $v_0(t), = 0.5, v_1(t) = 2.15$
    - 목적함수 $J(2) = 4 * 0.5 + 11 * 2.15 = 25.65$
- $t = 0, \ldots, 7$까지 반복하여 $J(t)$가 가장 낮은 값을 갖는 t를 임계값 $\hat{t}(=T)$로 설정하여 이진화 수행

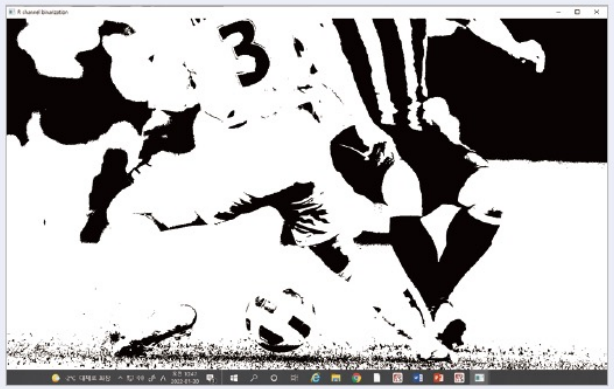
프로그래밍 실습: 이진화 알고리즘

- Otsu 알고리즘 실습

프로그램 3-3 오츄 알고리즘으로 이진화하기

```
01  import cv2 as cv
02  import sys
03
04  img=cv.imread('soccer.jpg')
05
06  t,bin_img=cv.threshold(img[:,:,2],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
07  print('오츄 알고리즘이 찾은 최적 임곗값=',t) ①
08
09  cv.imshow('R channel',img[:,:,2])                # R 채널 영상
10  cv.imshow('R channel binarization',bin_img)      # R 채널 이진화 영상
11
12  cv.waitKey()
13  cv.destroyAllWindows()
```

```
오츄 알고리즘이 찾은 최적 임곗값= 113.0 ①
```



프로그래밍 실습: 이진화 알고리즘

- OpenCV에서 제공하는 다양한 이진화 알고리즘

cv2.THRESH_BINARY	임계값 이상 = 최댓값, 임계값 이하 = 0
cv2.THRESH_BINARY_INV	위의 반전, 임계값 이상 = 0, 임계값 이하 = 최댓값
cv2.THRESH_TOZERO	임계값 이상 = 원본값, 임계값 이하 = 0
cv2.THRESH_TOZERO_INV	위의 반전, 임계값 이상 = 0, 임계값 이하 = 원본값
cv2.THRESH_TRUNC	임계값 이상 = 임계값, 임계값 이하 = 원본값
cv2.THRESH_MASK	흑색 이미지로
cv2.THRESH_OTSU	otsu 알고리즘
cv2.THRESH_TRIANGLE	triangle 알고리즘

모폴로지

- 연결 요소 (connected component)의 개념
  - 동일한 픽셀 값 (이진 영상에서는 0 또는 1)을 가지고 있으며 서로 연결되어 있는 픽셀 그룹
  - 4-connectivity와 8-connectivty 방식

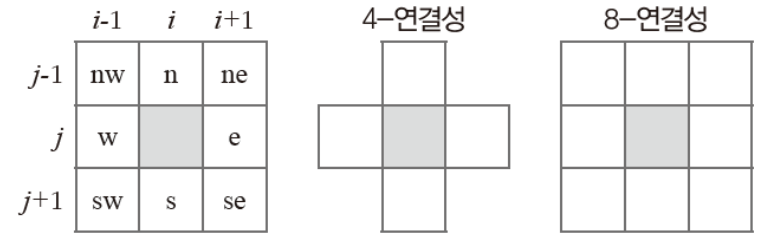


그림 3-10 화소의 연결성

- 연결 요소

[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

모폴로지

- 모폴로지 (morphology)
  - 구조 요소 (structuring element)를 이용하여 영역의 모양을 조작

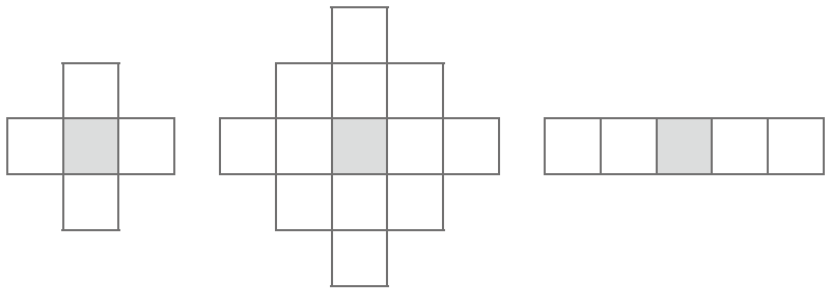
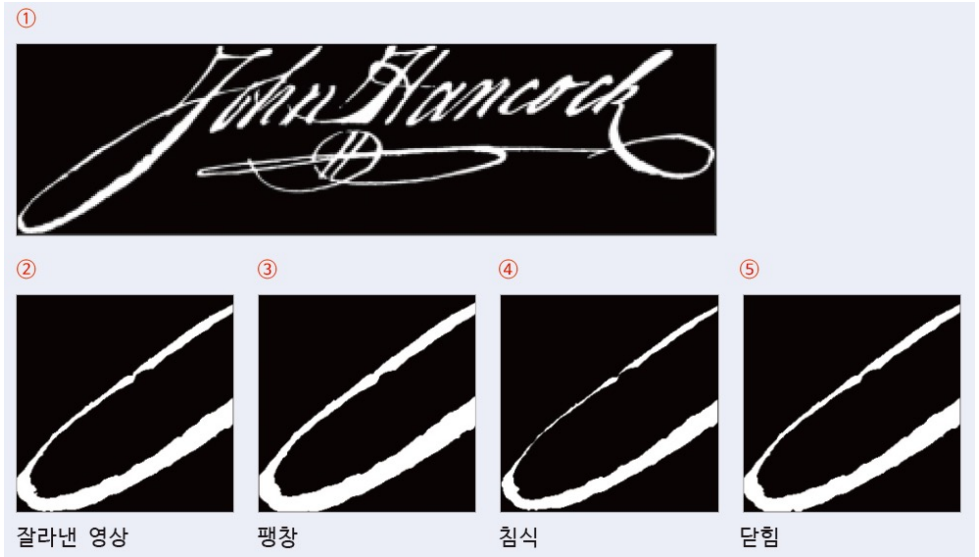


그림 3-12 모폴로지가 사용하는 구조 요소

- 모폴로지 연산
  - 팽창(dilation): 작은 홈을 메우거나 끊어진 영역을 연결하는 효과 ➔ 영역을 키움
  - 침식(erosion): 경계에 솟은 돌출 부분을 깎은 효과 ➔ 영역이 작아짐
  - 열림(opening): 침식한 결과에 팽창 적용 ➔ 원래 영역 크기 유지
  - 닫힘(closing): 팽창한 결과에 침식을 적용 ➔ 원래 영역 크기 유지

모폴로지

- 모폴로지 연산 – 팽창과 침식 연산 (예시)
  - 팽창: 구조 요소를 기준으로 양 옆을 1로 변환
  - 침식: 구조 요소를 기준으로 자기자신을 0으로 변환



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

구조 요소

(a) 입력 영상과 구조 요소

0	0	0	0	0	0	0	0
0	0	0	P 1	0	0	0	0
0	1	1	1	1	1	0	0
0	1	1	0	1	1	1	0

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

(b) 팽창

0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	P 1	1	1	1	1	0	0
0	1	1	0	1	1	1	0

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

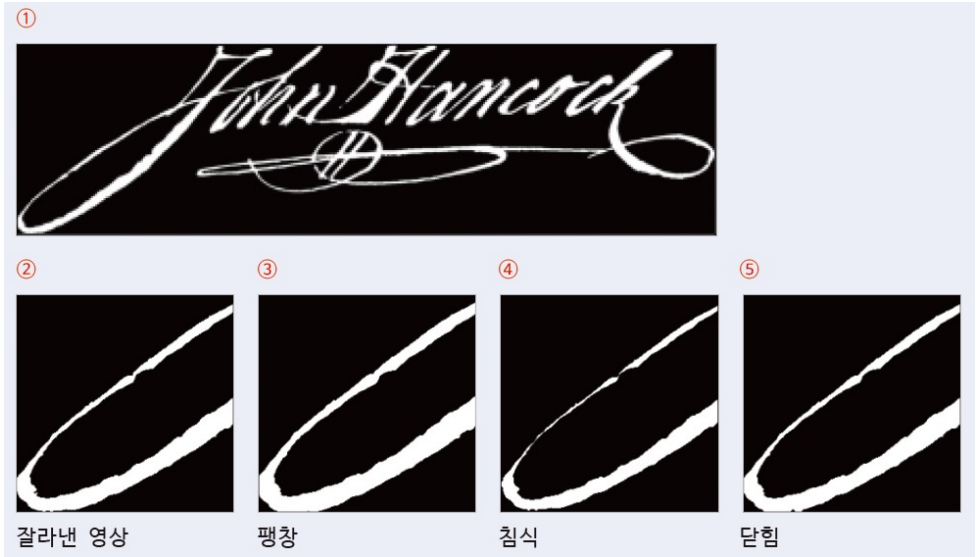
(c) 침식

그림 3-13 팽창과 침식

프로그래밍 실습: 모폴로지 연산

프로그램 3-4 모폴로지 연산 적용하기

```
01  import cv2 as cv
02  import numpy as np
03  import matplotlib.pyplot as plt
04
05  img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)
06
07  t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
08  plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([]) ①
09  plt.show()
```

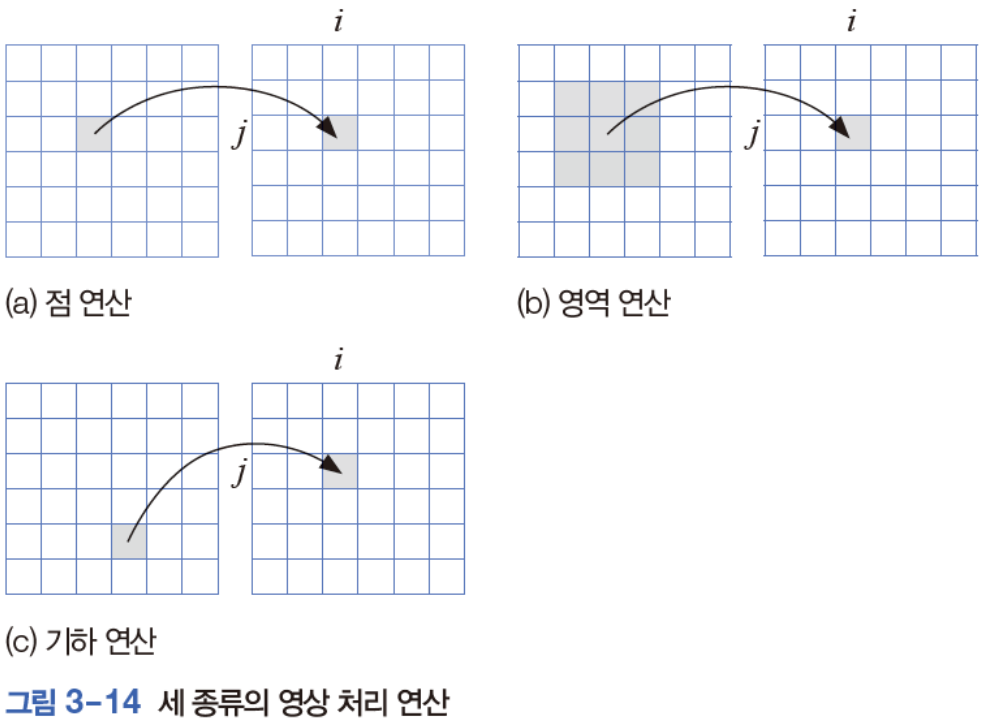



```
10
11  b=bin_img[bin_img.shape[0]//2:bin_img.shape[0],0:bin_img.shape[0]//2+1]
12  plt.imshow(b,cmap='gray'), plt.xticks([]), plt.yticks([]) ②
13  plt.show()
14
15  se=np.uint8([[0,0,1,0,0],                                    # 구조 요소
16               [0,1,1,1,0],
17               [1,1,1,1,1],
18               [0,1,1,1,0],
19               [0,0,1,0,0]])
20
21  b_dilation=cv.dilate(b,se,iterations=1)                       # 팽창
22  plt.imshow(b_dilation,cmap='gray'), plt.xticks([]), plt.yticks([]) ③
23  plt.show()
24
25  b_erosion=cv.erode(b,se,iterations=1)                         # 침식
26  plt.imshow(b_erosion,cmap='gray'), plt.xticks([]), plt.yticks([]) ④
27  plt.show()
28
29  b_closing=cv.erode(cv.dilate(b,se,iterations=1),se,iterations=1) # 닫기
30  plt.imshow(b_closing,cmap='gray'), plt.xticks([]), plt.yticks([]) ⑤
31  plt.show()
```

3. 점 연산

영상 처리의 기본 연산 종류

- 화소가 새로운 값을 어디서 받느냐에 따라 세 가지로 구분
- 점 연산 (point operation): 자기 자신으로부터 획득
  - 명암 조절, 히스토그램 평활화 등
- 영역 연산 (area operation): 이웃 화소들로부터 획득
  - convolution 연산 등
- 기하 연산 (geometry operation): 기하학적 변환이 정해주는 곳으로부터 획득
  - 기하변환 등



(a) 점 연산 (b) 영역 연산

(c) 기하 연산

그림 3-14 세 종류의 영상 처리 연산

점 연산 (Point operation)

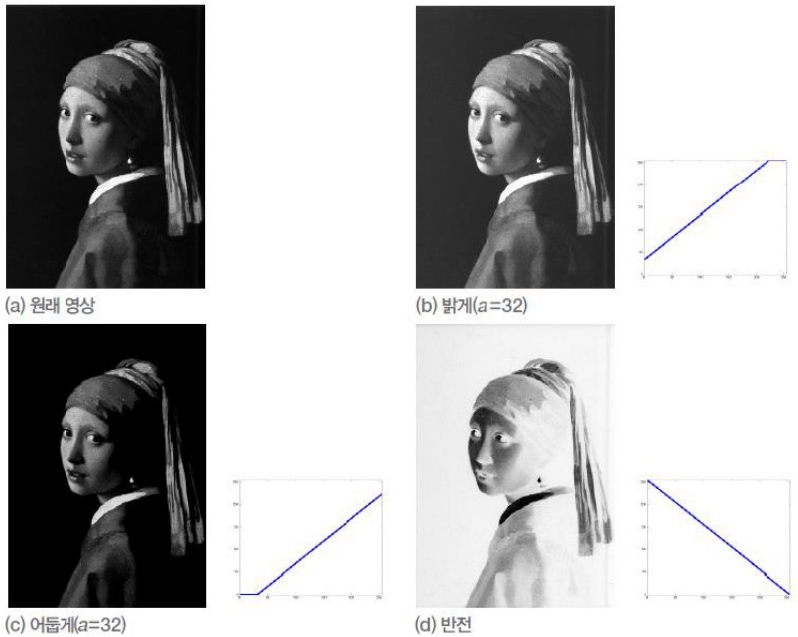
- 점 연산의 수식 표현
  - $f'(j,i) = t\big(f_1(j,i), f_2(j,i), \ldots f_k(j,i)\big)$
    - 대부분 $k = 1$ (한 장의 영상을 변환)
    - 자료에 따라 $f'(j,i)$를 $f_{out}(j,i)$로 표기하기도 함

- 점 연산의 예
  - 명암 조절 (Contrast adjustment): 영상을 밝거나 어둡게 조정
  - 히스토그램 평활화 (Histogram equalization): 영상의 대비를 개선하는 작업
    - 영상의 대비 = 영상의 밝기 분포
    - ➔ 전반적인 영상의 가시성 개선
  - 디졸브 (Dissolve): 두 장면이나 이미지 사이의 전환을 부드럽게 하는 기법 ($k = 2$)

명암 조절

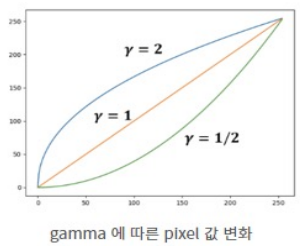
- 선형 연산 기반의 명암 조절
  - 모든 픽셀 값을 동일한 비율 만큼 증가 또는 감소

$$f'(j,i)=\begin{cases}\min\left(f(j,i)+a, L-1\right) & \text{밝게}\\ \max\left(f(j,i)-a, 0\right) & \text{어둡게}\\ (L-1)-f(j,i) & \text{반전}\end{cases} \qquad (3.4)$$

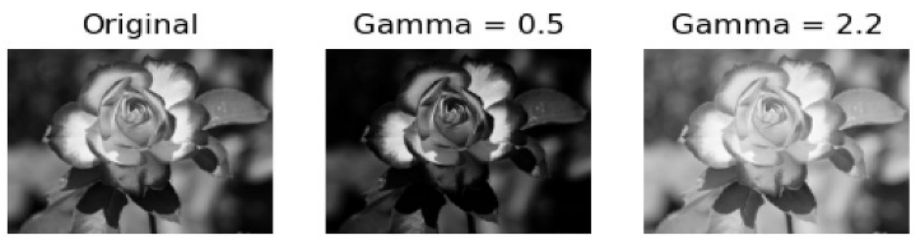


- 비선형 연산을 통한 명암 조절
  - 예시) 감마 조정 (감마 수정)
    - 모니터나 프린터 색상 조절에 사용

$$f'(j,i)=(L-1)\times \dot{f}(j,i)^{\gamma} \qquad (3.5)$$



gamma 에 따른 pixel 값 변화



프로그래밍 실습: 명암 조절 (감마 보정)

프로그램 3-5 감마 보정 실험하기

```
01  import cv2 as cv
02  import numpy as np
03
04  img=cv.imread('soccer.jpg')
05  img=cv.resize(img,dsize=(0,0),fx=0.25,fy=0.25)
06
07  def gamma(f,gamma=1.0):
08      f1=f/255.0                          # L=256이라고 가정
09      return np.uint8(255*(f1**gamma))
10
11  gc=np.hstack((gamma(img,0.5),gamma(img,0.75),gamma(img,1.0),gamma(img,2.0),gamma
            (img,3.0)))
12  cv.imshow('gamma',gc)
13
14  cv.waitKey()
15  cv.destroyAllWindows()
```

numpy.float64 형

numpy.uint8 형으로 변환

numpy.hstack 함수로 이어붙이기

히스토그램 평활화

- 영상의 대비를 개선하기 위하여 활용
- 히스토그램이 평평하게 되도록 (밝기 분포를 고르게) 영상을 조작해 영상의 대비를 높이는 기법

$$l' = \text{round}\left(\ddot{h}(l) \times (L-1)\right) \qquad (3.6)$$

- $\dot{h}$: 해당 픽셀의 비율
- $\ddot{h}$: 누적 비율

l	h	$\dot{h}$	$\ddot{h}$	$\ddot{h}\times 7$	l'
0	2	0.03125	0.03125	0.21875	0
1	12	0.1875	0.21875	1.53125	2
2	17	0.265625	0.484375	3.390625	3
3	10	0.15625	0.640625	4.484375	4
4	3	0.046875	0.6875	4.8125	5
5	7	0.109375	0.796875	5.578125	6
6	11	0.171875	0.96875	6.78125	7
7	2	0.03125	1.0	7.0	7

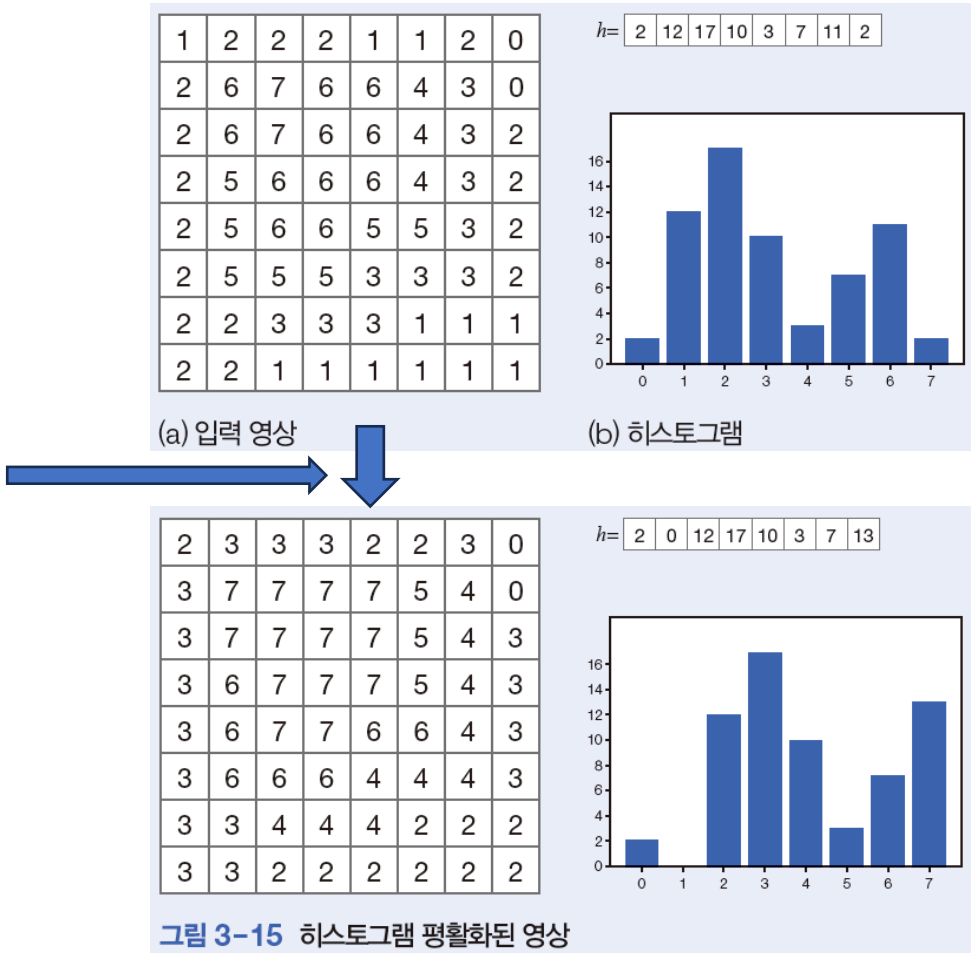
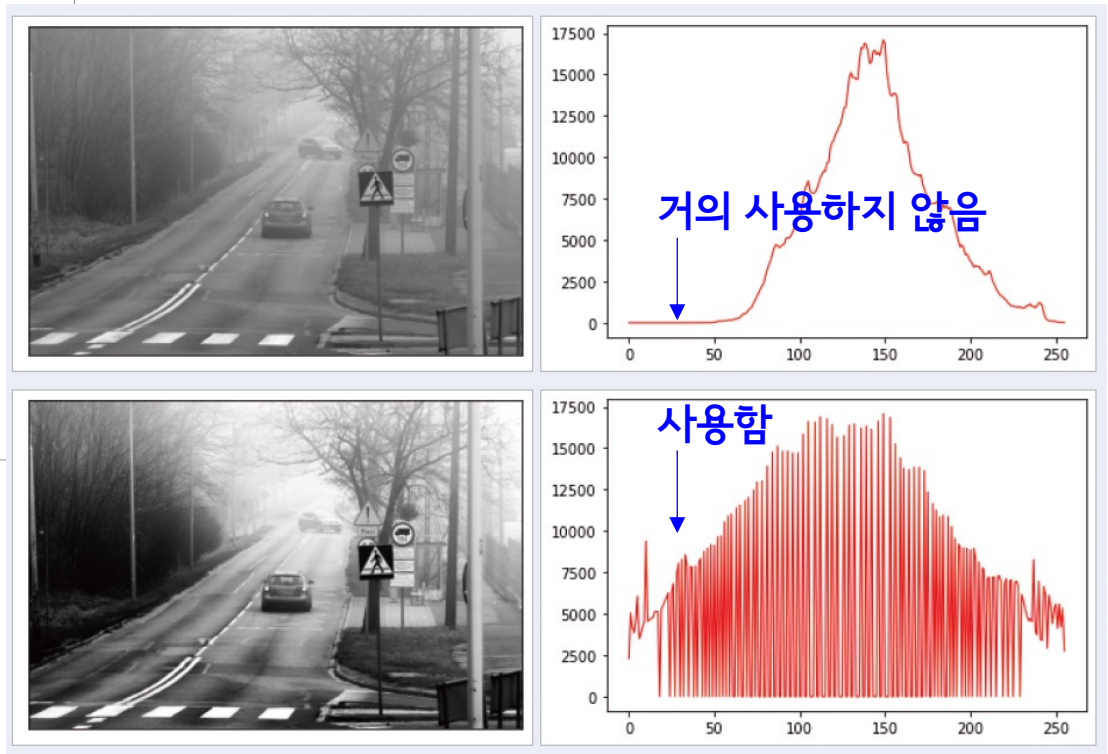


그림 3-15 히스토그램 평활화된 영상

프로그래밍 실습: 히스토그램 평활화

프로그램 3-6 히스토그램 평활화하기

```
01  import cv2 as cv
02  import matplotlib.pyplot as plt
03
04  img=cv.imread('mistyroad.jpg')
05
06  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)          # 명암 영상으로 변환하고 출력
07  plt.imshow(gray,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()
08
09  h=cv.calcHist([gray],[0],None,[256],[0,256])     # 히스토그램을 구해 출력
10  plt.plot(h,color='r',linewidth=1), plt.show()
11
12  equal=cv.equalizeHist(gray)                      # 히스토그램을 평활화하고 출력
13  plt.imshow(equal,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()
14
15  h=cv.calcHist([equal],[0],None,[256],[0,256])    # 히스토그램을 구해 출력
16  plt.plot(h,color='r',linewidth=1), plt.show()
```



디졸브

- 영상에서 앞 영상의 끝~그 다음 영상의 시작 간의 fade in/out 효과
  - $f'(j,i) = \alpha * f_1(j,i) + (1-\alpha) * f_2(j,i)$
    - f_1: 직전 영상
    - f_2: 직후 영상
  - OpenCV에서 별도의 함수는 없으며 cv2.addWeighted() 함수를 사용하여 두 이미지의 가중치를 조절하여 동일한 효과를 발생

프로그래밍 실습: 디졸브

- 디졸브 연산 시 두 이미지가 반드시 같은 크기로 설정되어야 함

```
import cv2
import numpy as np

def resize_image_to_match(image1, image2):
  height, width = image1.shape[:2]
  resized_image2 = cv2.resize(image2, (width, height))
  return resized_image2

def dissolve_effect(image1, image2, steps=30):
  image2 = resize_image_to_match(image1, image2)
  for alpha in np.linspace(0, 1, steps):
    beta = 1.0 - alpha
    dst = cv2.addWeighted(image1, alpha, image2, beta, 0)
    cv2.imshow('Dissolve Transition', dst)
    cv2.waitKey(100)
    cv2.destroyAllWindows()

'''
image1 = cv2.imread('path_to_your_first_image.jpg')
image2 = cv2.imread('path_to_your_second_image.jpg')
dissolve_effect(image1, image2, steps=30)
```

4. 영역 연산

영역 연산 (Area operation)

- 영역 연산은 이웃 화소를 고려해서 새로운 값을 결정함
- 주로 convolution 연산을 통해 이루어짐

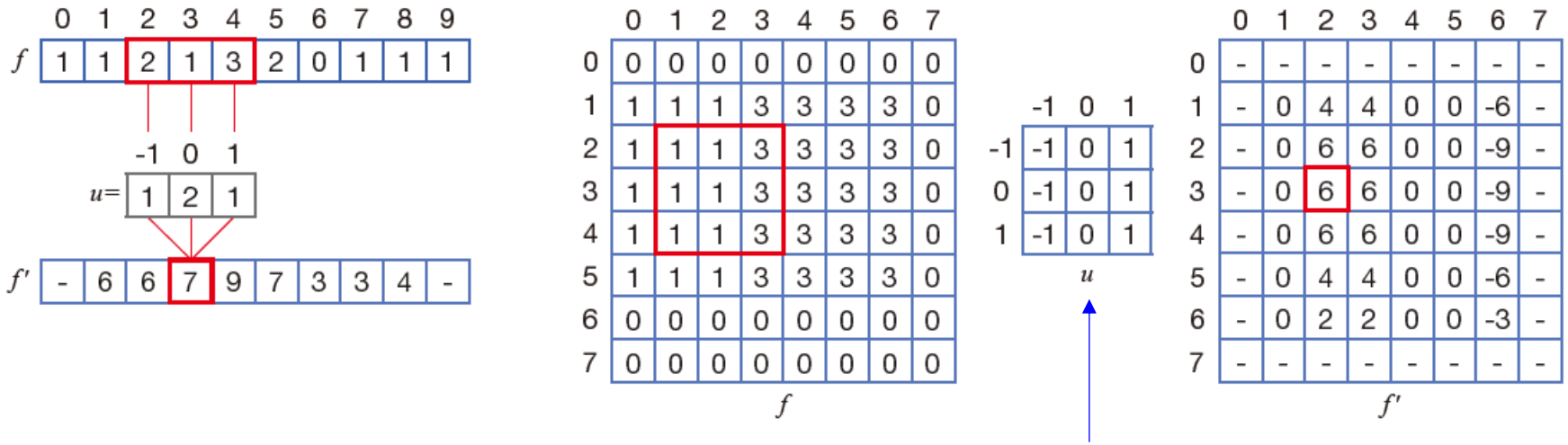
- 영역 연산의 예
  - 블러링(blurring) 또는 스무딩(smoothing)
    - 영상에서 노이즈를 줄이거나 부드럽게 만들기 위해 사용됨
  - 샤프닝(sharpening)
    - 영상의 대비를 높여 세부 사항을 더욱 뚜렷하게 만드는 기법
  - 엣지 검출(Edge detection)
    - 영상 내에서 객체의 경계를 찾는 과정

Convolution

- 각 화소에 필터 u를 적용해 곱의 합(Sum of product)을 구하는 연산
  - 필터 u를 커널(kernel) 또는 마스크(mask)라고도 함
  - u의 값에 따라서 다양한 결과를 도출 할 수 있음

$$f'(x) = \sum_{i=-(w-1)/2}^{(w-1)/2} u(i) f(x+i) \qquad (3.7)$$

$$f'(y,x) = \sum_{j=-(h-1)/2}^{(h-1)/2} \sum_{i=-(w-1)/2}^{(w-1)/2} u(j,i) f(y+j, x+i) \qquad (3.8)$$

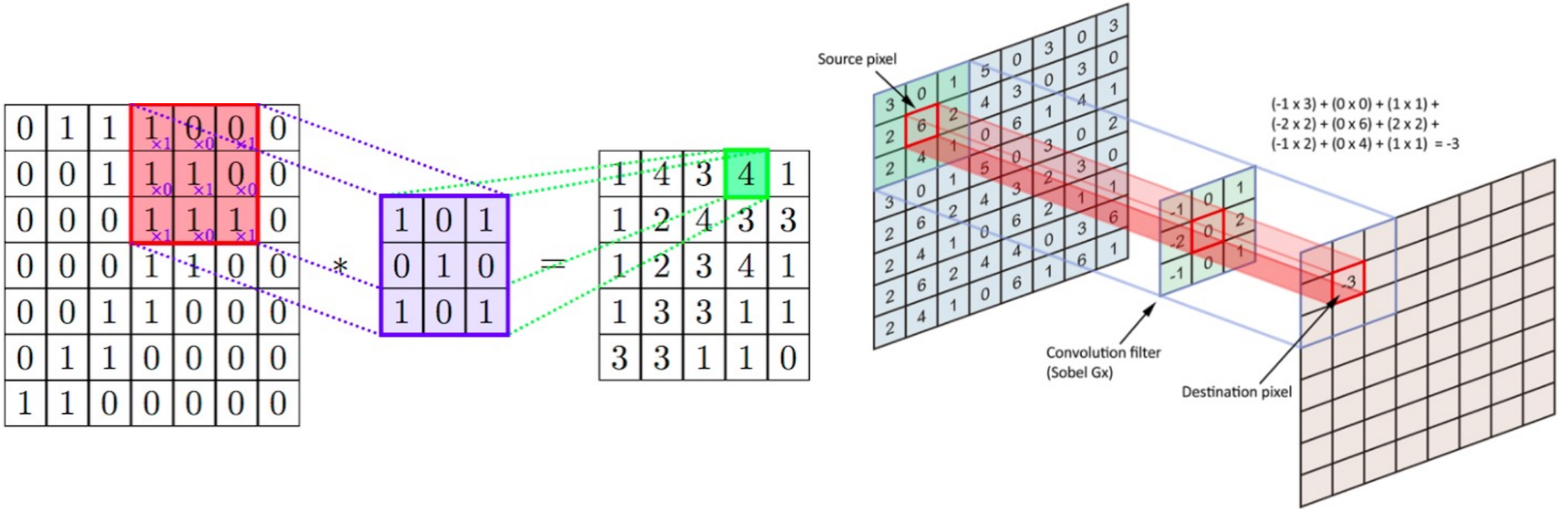


(a) 1차원 영상에 컨볼루션 적용 (b) 2차원 영상에 컨볼루션 적용

그림 3-16 컨볼루션의 원리

Convolution

- convolution 연산 (상세)



다양한 필터 (커널)

- 목적에 따라 다양한 필터 사용
  - 이미 연구를 통해 다양한 필터가 발명됨

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

0.0030	0.0133	0.0219	0.0133	0.0030
0.0133	0.0596	0.0983	0.0596	0.0133
0.0219	0.0983	0.1621	0.0983	0.0219
0.0133	0.0596	0.0983	0.0596	0.0133
0.0030	0.0133	0.0219	0.0133	0.0030

(a) 스무딩 필터

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

-1	-1	-1
-1	8	-1
-1	-1	-1

(b) 샤프닝 필터

-1	0	0
0	0	0
0	0	1

-1	-1	0
-1	0	1
0	1	1

(c) 엠보싱 필터

그림 3-17 잡음 제거와 대비 향상을 위한 필터

박스

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

가우시안

.0000	.0000	.0002	.0000	.0000
.0000	.0113	.0837	.0113	.0000
.0002	.0837	.6187	.0837	.0002
.0000	.0113	.0837	.0113	.0000
.0000	.0000	.0002	.0000	.0000

샤프닝

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

수평 에지

1	1	1
0	0	0
-1	-1	-1

수직 에지

1	0	-1
1	0	-1
1	0	-1

모션

.0304	.0501	0	0	0
.0501	.1771	.0519	0	0
0	.0519	.1771	.0519	0
0	0	.0519	.1771	.0501
0	0	0	.0501	.0304

(a) 원래 영상과 여러 가지 마스크들

> 박스

> 가우시안

> 샤프닝

> 수평 에지

> 수직 에지

> 모션

(b) 다양한 마스크로 컨볼루션한 영상들

다양한 필터 (커널)

- 가우시안(Gaussian) 필터

$$\text{1차원 가우시안: } g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

$$\text{2차원 가우시안: } g(y, x) = \frac{1}{\sigma^2 2\pi} e^{-\frac{y^2+x^2}{2\sigma^2}} \qquad (3.9)$$

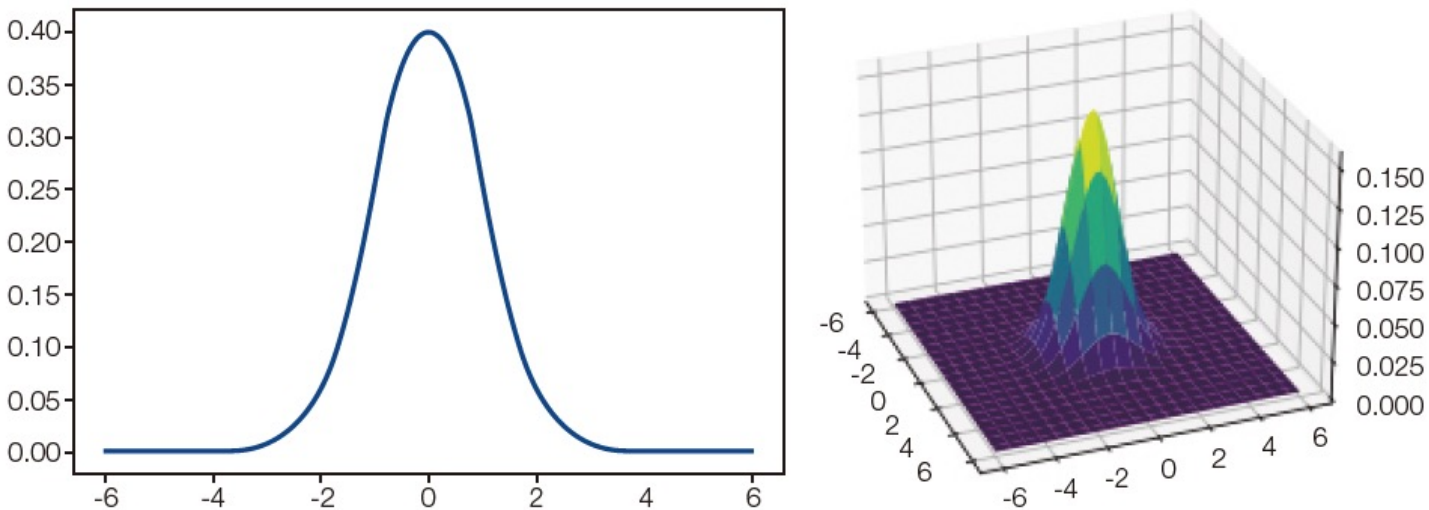


그림 3-18 1차원과 2차원 가우시안 함수

프로그래밍 실습: 필터

- 임의의 필터(커널) 생성 후 원본 이미지와 convolution 연산 수행

```
path = './data/view.png’

image = cv2.imread(path)

kernel = np.ones((5, 5))/5**2 # 5x5 평균 filter kernel 생성 생성

'''
kernel = np.array([[0.0, 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]])
'''

blurred = cv2.filter2D(image, -1, kernel)

cv2.imshow('origin', image)
cv2.imshow('avrg blur', blurred)

cv2.waitKey()
cv2.destroyAllWindows()
```

프로그래밍 실습: 가우시안 필터

- 1) 직접 가우시안 커널과 convolution 연산하는 방법
- 2) OpenCV에서 제공하는 가우시안 블러 함수를 호출하는 방법

```
path = './data/view.png'

image = cv2.imread(path)

# 1) 가우시안 커널을 활용하여 필터링하는 방법
g_kernel = cv2.getGaussianKernel(3, 0)
g_blur1 = cv2.filter2D(image, -1, g_kernel*g_kernel.T)

# 2) 가우시안 블러 OpenCV 함수를 호출하여 활용
g_blur2 = cv2.GaussianBlur(image, (3, 3), 0)

cv2.imshow('origin', image)
cv2.imshow('g_blur1', g_blur1)
cv2.imshow('g_blur2', g_blur2)
cv2.waitKey()
cv2.destroyAllWindows()
```

데이터 형과 convolution

- 연산 결과를 저장하는 변수의 유효 값 범위
  - OpenCV는 주의를 기울여 작성되어 있음
  - 때로 프로그래머가 직접 신경 써야 하는 경우 있음. ex) filter2D 함수
- 데이터 형
  - OpenCV는 영상 화소를 주로 numpy.uint8 형으로 표현 ([0, 255] 범위)

```
In [1]: print(type(img[0,0,0]))
        numpy.uint8
```

  - [0,255] 범위를 벗어나는 경우 문제 발생 (엠보싱 필터 등)

```
In [2]: a=np.array([-3,-2,-1,0,1,254,255,256,257,258],dtype=np.uint8)
In [3]: print(a)
        [253 254 255   0   1 254 255   0   1   2]
```

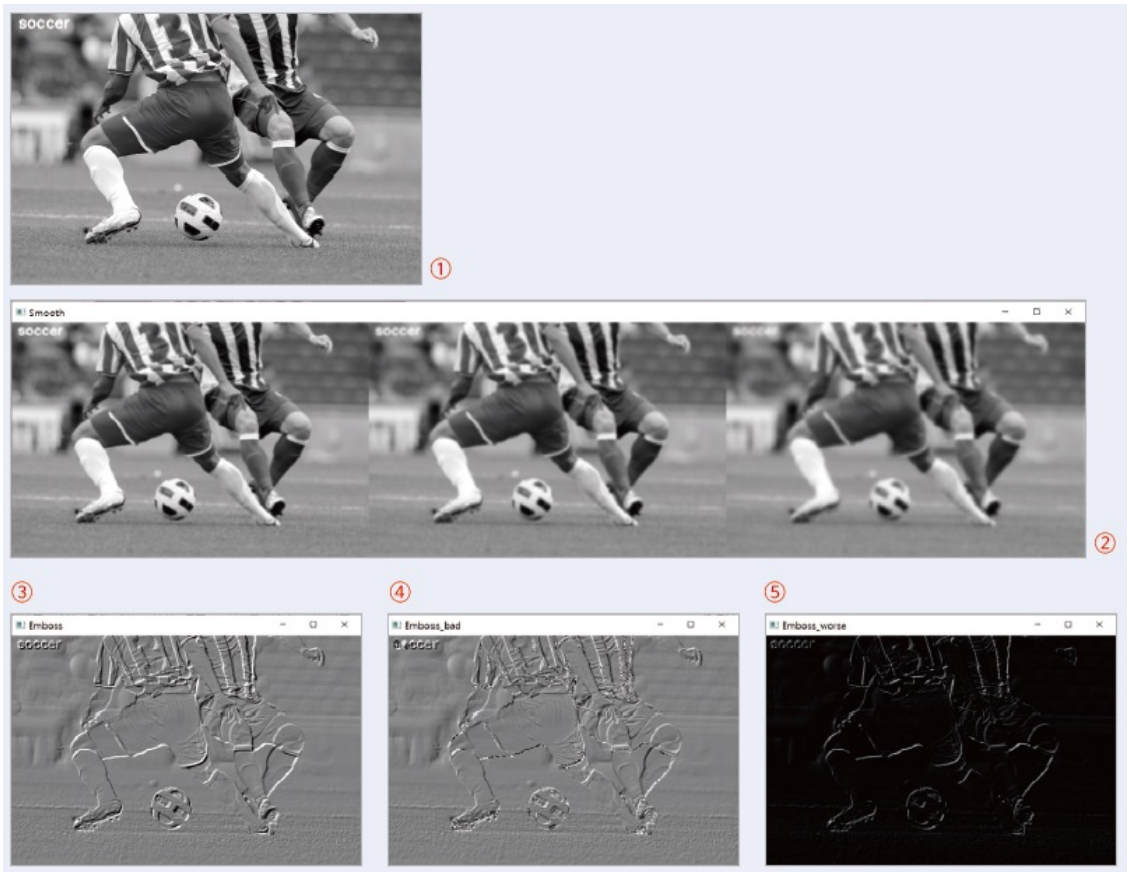
- 즉, 모든 픽셀 값의 범위가 [0, 255] 범위에 있을 수 있도록 조정해줄 필요가 있음

프로그래밍 실습: 가우시안 스무딩과 엠보싱

- 엠보싱 필터 적용 후 픽셀 값이 [0, 255] 범위를 벗어나는 것을 보정

프로그램 3-7 컨볼루션 적용(가우시안 스무딩과 엠보싱)하기

```
01  import cv2 as cv
02  import numpy as np
03
04  img=cv.imread('soccer.jpg')
05  img=cv.resize(img,dsize=(0,0),fx=0.4,fy=0.4)
06  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
07  cv.putText(gray,'soccer',(10,20),cv.FONT_HERSHEY_SIMPLEX,0.7,(255,255,255),2)
08  cv.imshow('Original',gray) ①
09
10  smooth=np.hstack((cv.GaussianBlur(gray,(5,5),0.0),cv.
                  GaussianBlur(gray,(9,9),0.0),cv.GaussianBlur(gray,(15,15),0.0)))
11  cv.imshow('Smooth',smooth) ②
12
13  femboss=np.array([[-1.0, 0.0, 0.0],
14                    [ 0.0, 0.0, 0.0],
15                    [ 0.0, 0.0, 1.0]])
16
17  gray16=np.int16(gray)
18  emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
19  emboss_bad=np.uint8(cv.filter2D(gray16,-1,femboss)+128)
20  emboss_worse=cv.filter2D(gray,-1,femboss)
21
22  cv.imshow('Emboss',emboss) ③
23  cv.imshow('Emboss_bad',emboss_bad) ④
24  cv.imshow('Emboss_worse',emboss_worse) ⑤
25
26  cv.waitKey()
27  cv.destroyAllWindows()
```



5. 기하 연산

기하 연산 (Geometry operation)

- 기하 연산
  - 기하학적 변환이 정해준 위치의 화소에서 값을 가져옴
  - 주로 물체의 이동, 크기, 회전에 따른 기하 변환을 수행

- 기하학적 변환
  - 이동(translation), 회전(rotation), 축적(scaling), 반사(reflection),
    원근/투시 변환(perspective transform), 비선형 변환(non-linear transform), 등고선 맵핑(contour warping),
    스플라인 보간(spline interpolation), 케글린 변환(keystone transform) 등

동차 좌표와 동차 행렬

- 동차 좌표(homogeneous coordinate)
  - 컴퓨터 비전, 컴퓨터 그래픽스, 로봇공학 등에서 널리 사용되는 좌표 시스템
  - 기존의 2차원 (또는 3차원) 좌표에 추가적인 차원을 더하여 확장한 것
    - ➔ 기하 변환(translation, scaling, rotation 등)에 용이
  - 2차원 좌표를 3차원 벡터로 표현
    - $p = (x, y) \rightarrow \bar{p} = (x, y, 1)$ ← 2차원 좌표에 1을 추가해 3차원 벡터로 표현
    - ex) (-2, 4, 1)과 (-4, 8, 2)는 (-2, 4)에 해당하는 같은 좌표

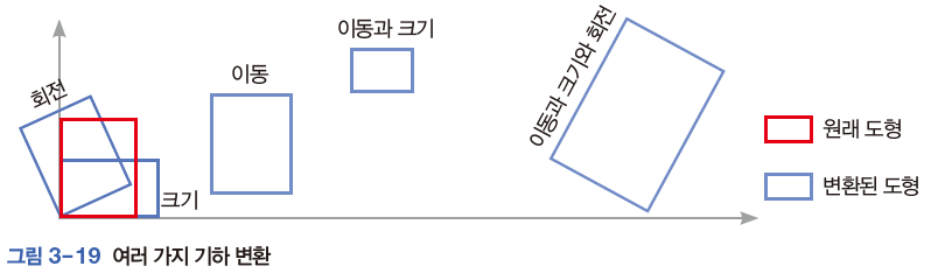


그림 3-19 여러 가지 기하 변환

  - 3차원 좌표를 4차원 벡터로 표현
    - $p = (x, y, z) \rightarrow \bar{p} = (x, y, z, 1)$ ← 3차원 좌표에 1을 추가해 4차원 벡터로 표현

동차 좌표와 동차 행렬

- 기하 연산을 동차 행렬(homogeneous matrix, $\dot{H}$)로 표현
  - 동차 행렬은 동차 변환 행렬(homogeneous transformation matrix)라고도 함
  - 여러가지 기하 변환
    - 이동(translation)
    - 축적(scaling)
    - 회전(rotation)
    - 반사(reflection)

(참고) 이동, 축적, 회전, 반사는 모두 affine transform의 한 종류임

affine transform: 2차원 (또는 3차원) 공간의 점, 직선, 평면을 보존하는 선형 변환

변환 전과 변환 후에 평행성이 유지된다는 특징

표 3-1 3가지 기하 변환

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_x만큼 크기 조정(1보다 크면 확대, 1보다 작으면 축소)

기하변환 - 이동

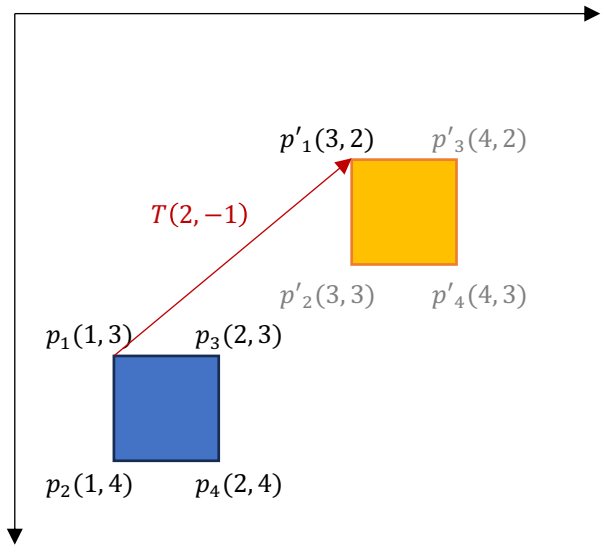
- 이동 (translation)
  - 이미지 내 한 점 $p(x, y)$를 각각 t_x, t_y 만큼 이동하는 경우

$$x' = x + t_x$$
$$y' = y + t_y$$

  - 행렬 표현

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 이동 예시



$$T(2,-1) = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\bar{p}'_1 = T(2,-1)\bar{p}_1 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \bar{p}_1 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

기하변환 - 축적

- 축적 (scaling)
  - 이미지 내 한 점 $p(x, y)$를 각각 s_x, s_y 만큼 축소 또는 확대하는 경우

원점 기준 scaling

$$x' = s_x x$$
$$y' = s_y y$$

- 행렬 표현

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

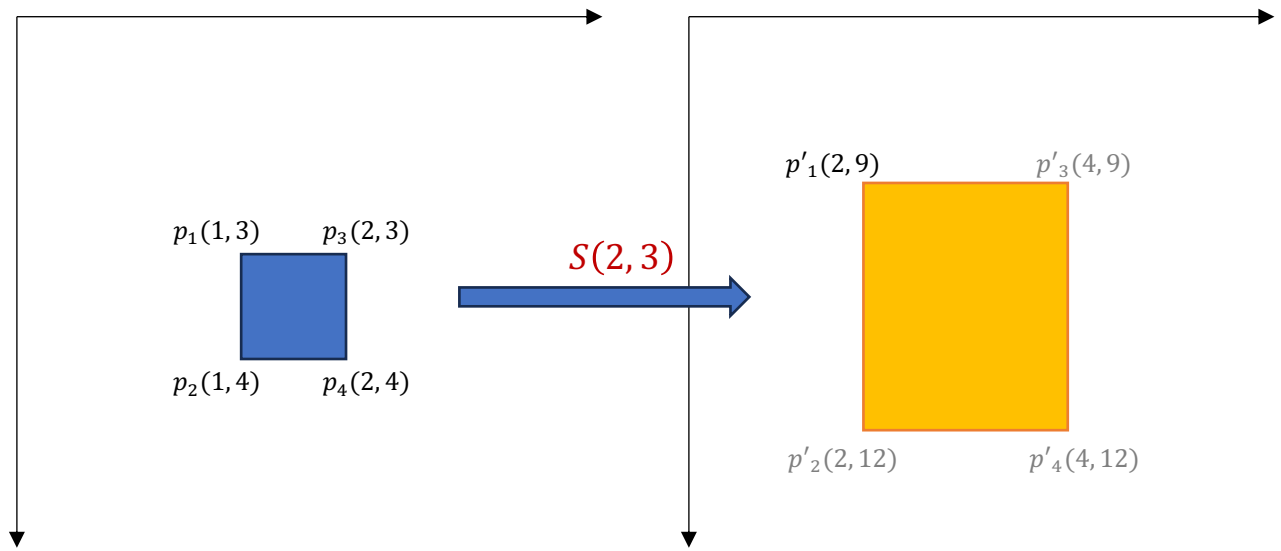
임의의 점 기준 scaling

$$x' = s_x(x - p_x) + p_x = s_x x + p_x(1 - s_x)$$
$$y' = s_y(x - p_y) + p_y = s_y y + p_y(1 - s_y)$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & p_x(1 - s_x) \\ 0 & s_y & p_y(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

기하변환 - 축적

- 축적 예시 – x축으로 2배, y축으로 3배 확대 한 경우



$$S(2,3) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\bar{p}'_1 = S(2,3)\bar{p}_1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \bar{p}_1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 9 \\ 1 \end{bmatrix}$$

기하변환 - 회전

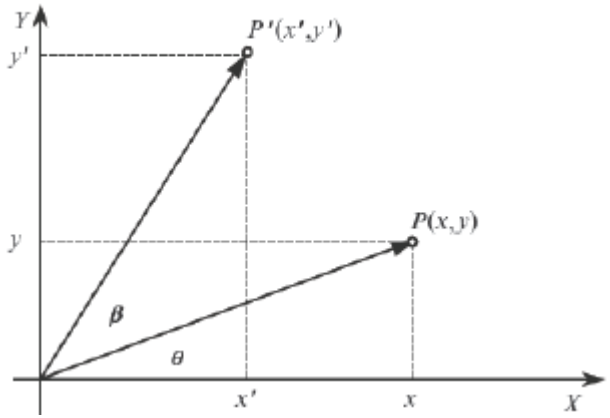
- 회전 (rotation)
  - 이미지 내 한 점 $p(x, y)$를 각도 β만큼 (반시계방향) 회전하는 경우
    - $R(p, \beta)$: $p(x, y) \rightarrow p'(x', y')$ ← **변환**의 수식적 표현

$$x' = \cos(\theta + \beta) = \cos\theta \cos\beta - \sin\theta \sin\beta = x\cos\beta - y\sin\beta$$

$$y' = \sin(\theta + \beta) = \sin\theta \cos\beta + \cos\theta \sin\beta = y\cos\beta + x\sin\beta$$

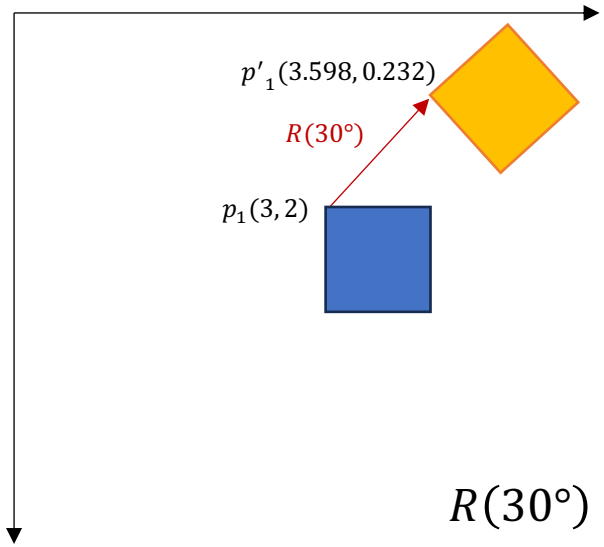
- 행렬 표현

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



기하변환 – 회전

- 회전 예시

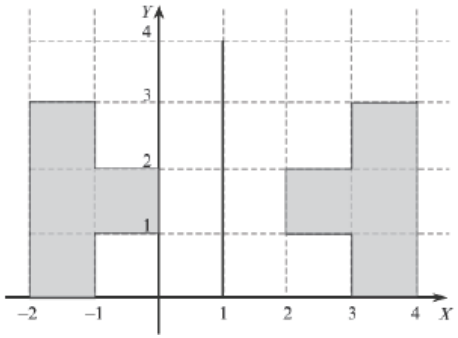


$$R(30°) = \begin{bmatrix} 0.8660 & 0.5 & 0 \\ -0.5 & 0.8660 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\bar{p}'_1 = R(30°)\bar{p}_1 = \begin{bmatrix} 0.8660 & 0.5 & 0 \\ -0.5 & 0.8660 & 0 \\ 0 & 0 & 1 \end{bmatrix} \bar{p}_1 = \begin{bmatrix} 0.8660 & 0.5 & 0 \\ -0.5 & 0.8660 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.598 \\ 0.232 \\ 1 \end{bmatrix}$$

기하변환 - 반사

- 반사 (reflection)
  - 이미지 내 한 점 $p(x, y)$를 특정 축에 대해서 반사하는 경우



y축 반사

$$x' = -x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

임의의 y축 반사 $(x = a)$

$$x' = -(x - a) + a = -x + 2a$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 2a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

x축 반사

$$x' = x$$
$$y' = -y$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

임의의 x축 반사 $(y = b)$

$$x' = x$$
$$y' = -(y - b) + b = -y + 2b$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 2b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

기하변환 - 반사

- 반사 (reflection)
  - 임의의 y축 $(x = a)$ 반사에 대한 해석
    - 1) 축을 원점으로 이동
      - $x = a$를 $x = 0$으로 이동

        ➔ 모든 x좌표에서 a를 뺌

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

    - 2) y축에 대해 반사
      - $x = 0$를 기준으로 반사하는 변환 적용

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

    - 3) 다시 원래 대로 이동

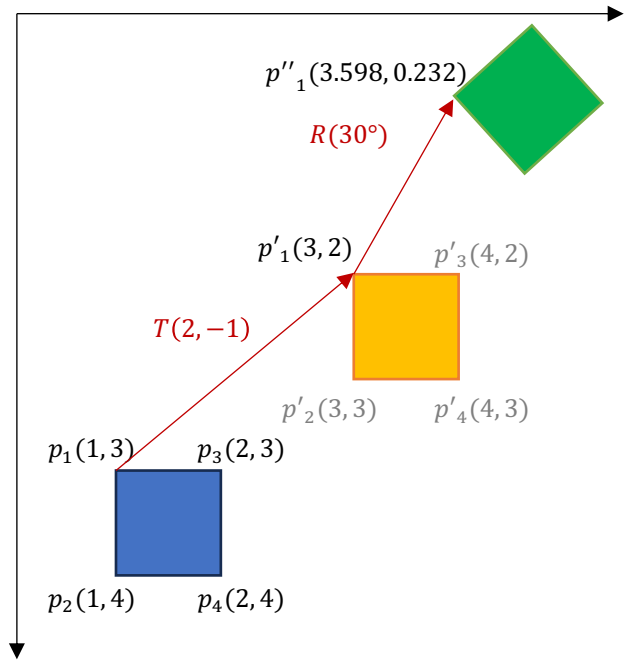
$$T_a = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = T_{-a} R_y T_a = \begin{bmatrix} -1 & 0 & 2a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

👆 동차행렬을 사용하는 이유

동차 행렬을 이용한 기하변환 예시

- 동차 행렬의 사용 ➔ 계산 효율화



$$A = R(30°)T(2,-1)$$

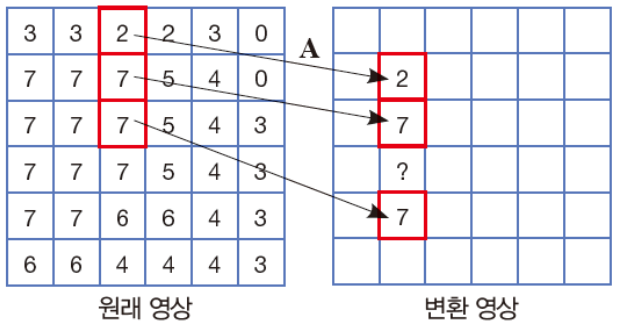
$$= \begin{bmatrix} 0.8660 & 0.5 & 0 \\ -0.5 & 0.8660 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 & 1.232 \\ -0.5 & 0.866 & -1.866 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\bar{p}''_1 = \begin{bmatrix} 0.866 & 0.5 & 1.232 \\ -0.5 & 0.866 & -1.866 \\ 0 & 0 & 1 \end{bmatrix} \bar{p}_1$$

$$= \begin{bmatrix} 0.866 & 0.5 & 1.232 \\ -0.5 & 0.866 & -1.866 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.598 \\ 0.232 \\ 1 \end{bmatrix}$$

기하변환의 문제점

- 화소 좌표에 동차 행렬을 적용하여 기하 변환
  - 전방 변환(forward transformation)
    - 원본 영상의 픽셀 위치를 직접 변환 함수에 적용하여 새로운 위치를 획득하는 방법
  - 값을 받지 못하는 화소가 발생할 수 있음
    - ex) $p_1(1,3) \Rightarrow p''_1(3.598, 0{,}232)$ ➔ 픽셀 좌표가 실수인 경우 정수형으로 변환
    - 결과 영상에서 픽셀 위치가 연속적이지 않게 되어 홀(holes), 오버랩(overlaps) 발생 ➔ 에일리어싱(aliasing)

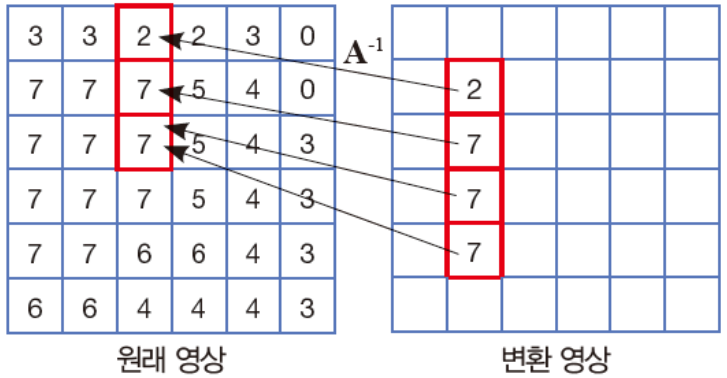


(a) 전방 변환

그림 3-21 영상의 기하 변환

에일리어싱 문제 해결 방법

- 후방 변환(inverse transformation) 활용
  - 결과 영상의 각 픽셀 위치에서 시작하여 원본 영상에 해당하는 위치를 역변환 함수를 통해 찾는 방법
  - 결과 영상의 모든 픽셀에 대해 원본 영상에서의 값을 결정



(b) 후방 변환

- 후방 변환의 문제점
  - 원본 영상의 특정 위치가 실수 좌표에 해당할 수 있음

    ➔ 정수로 변환하는 과정이 필요 (영상 보간, interpolation)

- (참고) 영상 보간은 후방 변환 과정에만 적용되는 것은 아님 (전방 변환에도 적용될 수 있음)
  - 전방 변환에서의 보간: missing value를 채우는 과정 [2, 7, ?, 7]

보간법 (Interpolation)

- 후방 변환 과정에서 실수 좌표에 해당하는 픽셀 값의 추정치를 계산하는 방법
  - 최근접 이웃 방법(k-nearest neighbor)
    - 변환된 픽셀 위치에서 가장 가까운 원본 픽셀의 값을 그대로 사용
  - 양선형 보간법(bilinear interpolation): 네 개의 가장 가까운 픽셀 값의 가중 평균을 계산하여 픽셀 값 추정

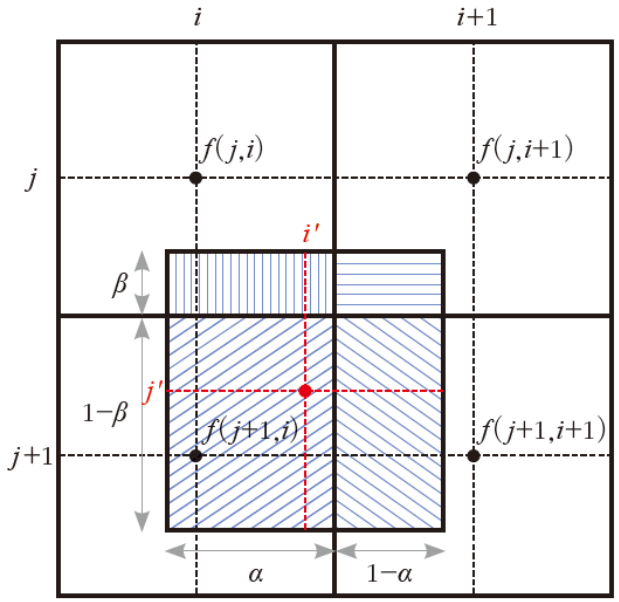


그림 3-22 실수 좌표의 화솟값을 보간하는 과정

$$f(j',i') = \alpha\beta f(j,i) + (1-\alpha)\beta f(j,i+1) + \alpha(1-\beta) f(j+1,i) + (1-\alpha)(1-\beta) f(j+1,i+1) \quad (3.11)$$

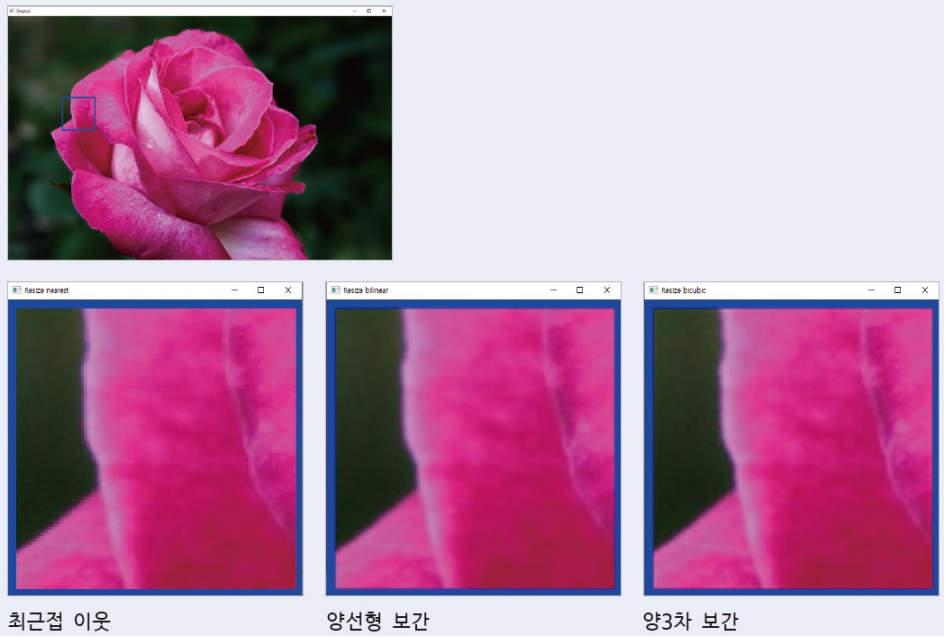
(참고) 삼선형 보간(trilinear interpolation), 바이큐빅 보간(bicubic interpolation) 등의 보간 알고리즘 존재

프로그래밍 실습: 영상 보간

- 보간을 이용해 영상의 기하 변환

프로그램 3-8 보간을 이용해 영상의 기하 변환하기

```
01  import cv2 as cv
02
03  img=cv.imread('rose.png')
04  patch=img[250:350,170:270,:]
05
06  img=cv.rectangle(img,(170,250),(270,350),(255,0,0),3)
07  patch1=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_NEAREST)
08  patch2=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_LINEAR)
09  patch3=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_CUBIC)
10
11  cv.imshow('Original',img)
12  cv.imshow('Resize nearest',patch1)
13  cv.imshow('Resize bilinear',patch2)
14  cv.imshow('Resize bicubic',patch3)
15
16  cv.waitKey()
17  cv.destroyAllWindows()
```



다양한 기하 변환의 구현

- cv2.warpPerspective(): 이미지의 기하학적 변환을 위한 함수
  - 원본 이미지와 동차행렬, 이미지의 크기 등을 파라미터로 가짐

- 이동

```
T = np.float32([[1, 0, 100], [0, 1, 50], [0, 0, 1]])
translated_image = cv2.warpPerspective(image, T, (image.shape[1], image.shape[0]))
```

- 회전 (x, y축으로 각각 1.5배 확대)

```
S = np.float32([[1.5, 0, 0], [0, 1.5, 0], [0, 0, 1]])
scaled_image = cv2.warpPerspective(image, S, (image.shape[1], image.shape[0]))
```

다양한 기하 변환의 구현

- 반사 (y축 반사)

```
(h, w) = image.shape[:2]
Ry = np.float32([[-1, 0, w], [0, 1, 0], [0, 0, 1]])
flipped_image = cv2.warpPerspective(image, Ry, (image.shape[1], image.shape[0]))
```

- 회전 (중심에서 45도 회전)

```
angle = np.radians(45) # 각도 -> 라디안
cos = np.cos(angle)
sin = np.sin(angle)
cx, cy = w / 2, h / 2 # 이미지의 중심

R = np.float32([[cos, -sin, cx*(1-cos)+cy*sin],
                [sin, cos, cy*(1-cos)-cx*sin],
                [0, 0, 1]])

rotated_image = cv2.warpPerspective(image, R, (image.shape[1], image.shape[0]))
```

다양한 기하 변환의 구현 – OpenCV 함수 활용

- OpenCV에서 다양한 기하 연산(기하학적 변환)을 위한 함수 제공
- 이동: cv2.warpAffine, cv2.translate
- 회전: cv2.getRotationMatrix2D, cv2.warpAffine
- 축적: cv2.resize
  - 영상 보간 알고리즘을 파라미터로 선택
- 반사: cv2.flip
  - 대칭(반사)축을 파라미터로 선택

6. OpenCV의 계산 효율

OpenCV의 시간 효율

- 컴퓨터 비전의 task는 인식 정확도 뿐만 아니라 시간 효율도 중요
  - 특히 실시간 처리가 요구되는 응용이 대부분임

- OpenCV (함수)는 효율적으로 구현되어 있기 때문에 직접 구현하는 것보다 OpenCV 함수를 사용하는 것이 유리
  - OpenCV이 내부는 C/C++로 구현되어져있고, intel microprocessor에 최적화되어 있음

- 직접 구현하는 경우 Python의 배열 연산을 사용하는 것이 유리

프로그래밍 실습: OpenCV의 시간 효율 비교

프로그램 3-9 직접 작성한 함수와 OpenCV가 제공하는 함수의 시간 비교하기

```
01  import cv2 as cv
02  import numpy as np
03  import time
04
05  def my_cvtGray1(bgr_img):
06      g=np.zeros([bgr_img.shape[0],bgr_img.shape[1]])
07      for r in range(bgr_img.shape[0]):
08          for c in range(bgr_img.shape[1]):
09              g[r,c]=0.114*bgr_img[r,c,0]+0.587*bgr_img[r,c,1]+0.299*bgr_img[r,c,2
10      return np.uint8(g)
11
12  def my_cvtGray2(bgr_img):
13      g=np.zeros([bgr_img.shape[0],bgr_img.shape[1]])
14      g=0.114*bgr_img[:,:,0]+0.587*bgr_img[:,:,1]+0.299*bgr_img[:,:,2]
15      return np.uint8(g)
16
17  img=cv.imread('girl_laughing.png')
18
19  start=time.time()
20  my_cvtGray1(img)
21  print('My time1:',time.time()-start) ①
22
23  start=time.time()
24  my_cvtGray2(img)
25  print('My time2:',time.time()-start) ②
26
27  start=time.time()
28  cv.cvtColor(img,cv.COLOR_BGR2GRAY)
29  print('OpenCV time:',time.time()-start) ③
```

```
My time1: 4.798288106918335 ①
My time2: 0.015836000442504883 ②
OpenCV time: 0.013601541519165039 ③
```

End of slide