

Edge and Region

Byeongjoon Noh

Dept. of AI and Bigdata, SCH Univ.

powernoh@sch.ac.kr

Contents

1. Edge
2. Region

1. Edge

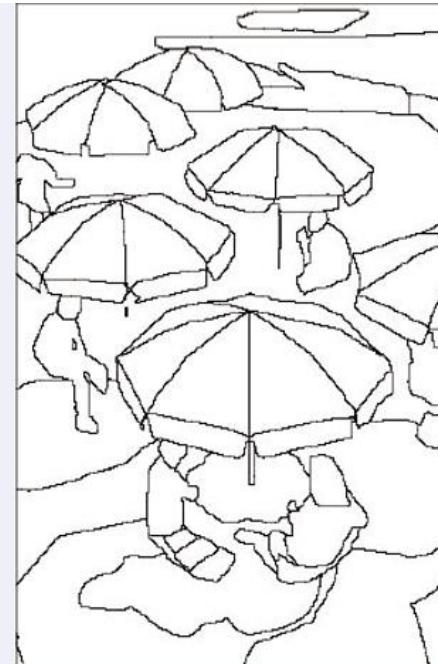
Preview

Edge 검출과 영역 (Region)분할

- → 컴퓨터비전 초창기부터 중요한 연구 주제
- Q. 컴퓨터비전 알고리즘이 사람 수준으로 분할할 수 있을까?



(a) 원래 영상



(b) 영역 영상(사람이 분할)

Preview

Edge와 region은 서로 다른 접근 방법 사용

- Edge → 특성이 다른 곳 검출
- Region → 유사한 화소를 묶음

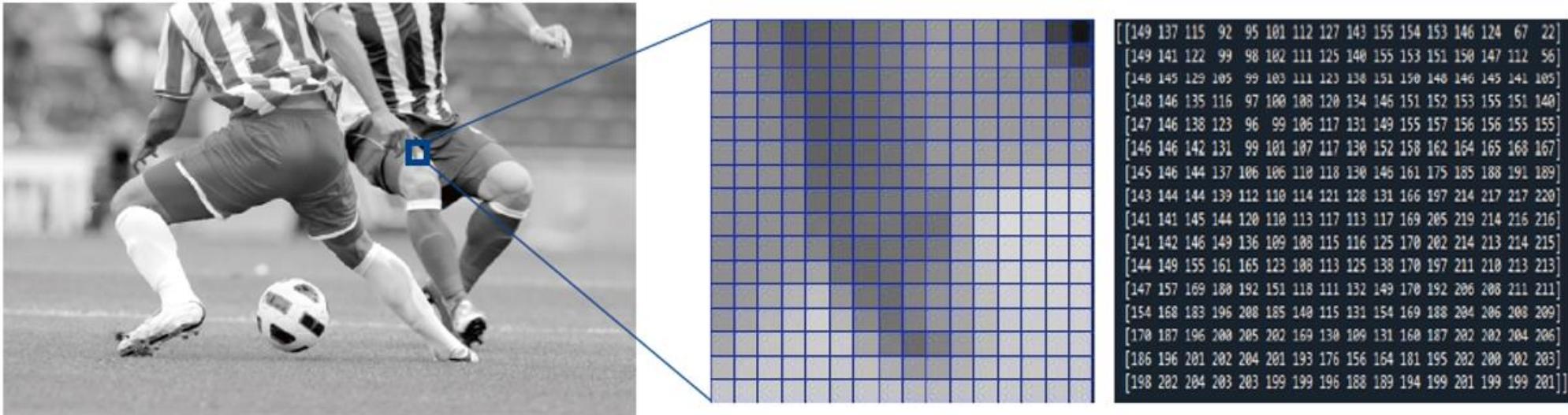
Human → Semantic segmentation (의미 분할)에 능숙

- 사람은 머리 속에 기억된 물체 모델을 활용하여 의미 분할
- 본 장에서 학습하는 고전적인 방법으로는 의미 분할 불가능
- → Deep learning은 의미 분발 가능! (Chapter 6?)

Edge detection

Edge detection 알고리즘

- Idea: 물체 내부는 명암이 서서히 변하고 경계 (Edge)는 급격히 변하는 특성을 활용



- 어떻게 변화를 탐지할 수 있을까? → 미분

Edge detection

미분

- 변수 x 가 미세하게 증가할 때 함수 변화량을 측정하는 수학적 기법

$$f'(x) = \lim_{\delta x \rightarrow 0} \frac{f(x + \delta x) - f(x)}{\delta x}$$

- 디지털 영상에서의 미분

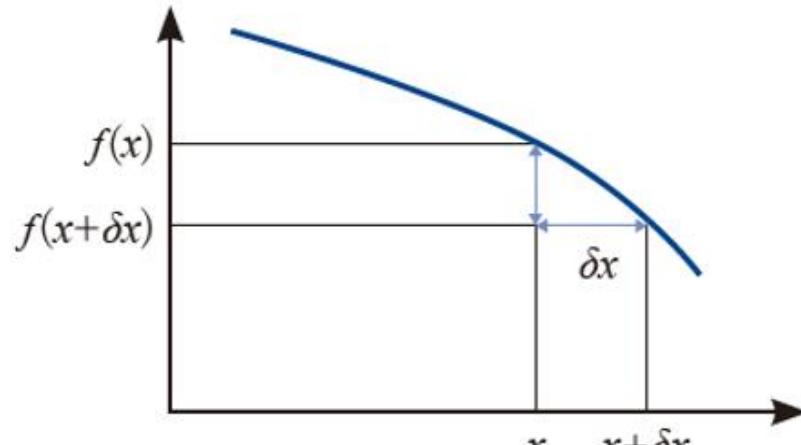
$$f'(x) = \frac{f(x + \delta x) - f(x)}{\delta x} = f(x + 1) - f(x)$$

- 의미: 주변 화소와의 값 차이

Edge detection

미분

- 실제 구현은 필터 μ 로 convolution 연산 수행
 - μ = Edge operator



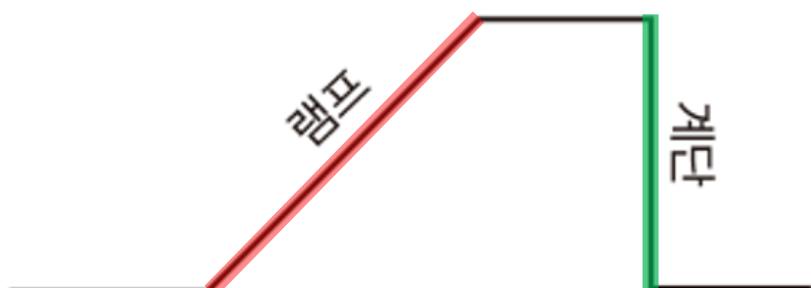
(a) 연속 함수의 미분



Edge detection

현실 세계의 램프 에지 문제

- 현실 세계에서는 명암이 몇 화소에 걸쳐서 변함



두꺼운 에지로 인해 위치찾기 문제 발생



Edge detection

2차 미분

$$\begin{aligned}f''(x) &= \frac{f'(x) - f'(x - \delta)}{\delta} = f'(x) - f'(x - 1) \\&= (f(x + 1) - f(x)) - (f(x) - f(x - 1)) = f(x + 1) - 2f(x) + f(x - 1)\end{aligned}$$

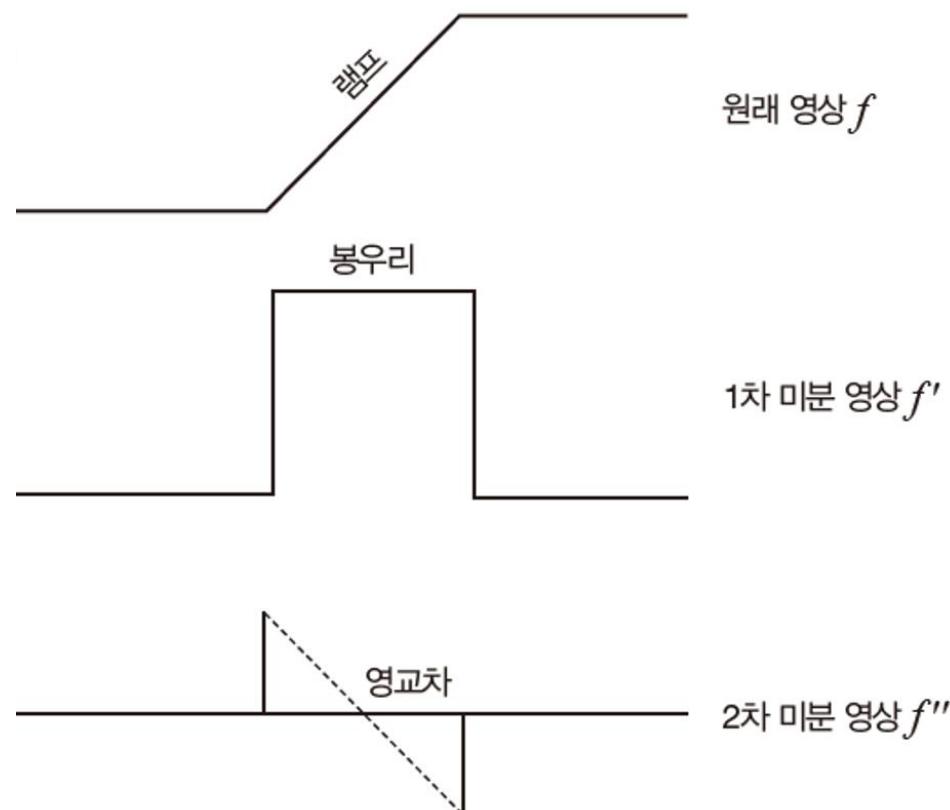
- 이 식을 구현 하는 필터

1	-2	1
---	----	---

Edge operator

Edge 검출 방법

- 1차 미분에서 봉우리 찾기
- 2차 미분에서 영교차(zero-crossing) 찾기



Edge operator

1차 미분에 기반한 edge operator

- 실제 영상에 있는 잡음을 흡수하기 위해 크기가 2인 필터를 3으로 확장

$$f'_x(y, x) = f(y, x + 1) - f(y, x - 1)$$

$$f'_y(y, x) = f(y + 1, x) - f(y - 1, x)$$

- 위 식을 구현하는 필터 $\mu_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ $\mu_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

- 2차원 확장

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

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

(a) 프레윗(Prewitt) 연산자

$$u_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$u_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

(b) 소벨(Sobel) 연산자

총합 = zero

Edge operator

Edge 강도와 방향

- Gradient

$$\nabla f = \left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x} \right) = (d_y, d_x)$$

- Edge 강도

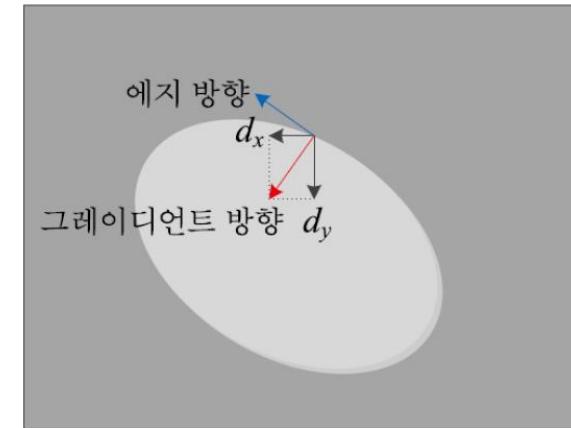
$$s(y, x) = \sqrt{f'_x(y, x)^2 + f'_y(y, x)^2} = \sqrt{d_y^2 + d_x^2}$$

Edge 일 가능성 또는 신뢰도 (Confidence)

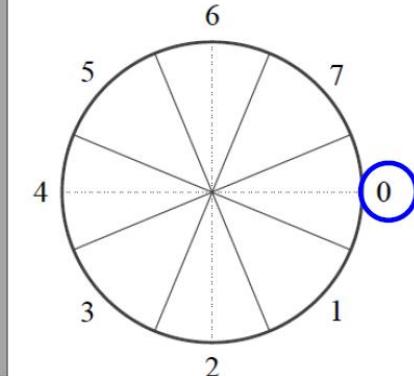
- Gradient 방향

- Edge 방향과 수직

$$d(y, x) = \arctan \left(\frac{f'_y(y, x)}{f'_x(y, x)} \right) = \arctan \left(\frac{d_y}{d_x} \right)$$



(a) 예지 방향과 그레이디언트 방향



(b) 예지 방향의 양자화

Edge operator

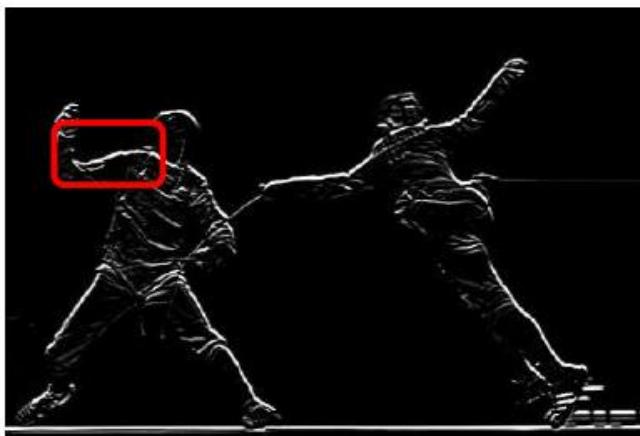
Edge detection 예시



(a) 원래 영상



(b) 에지 강도



(c) d_y

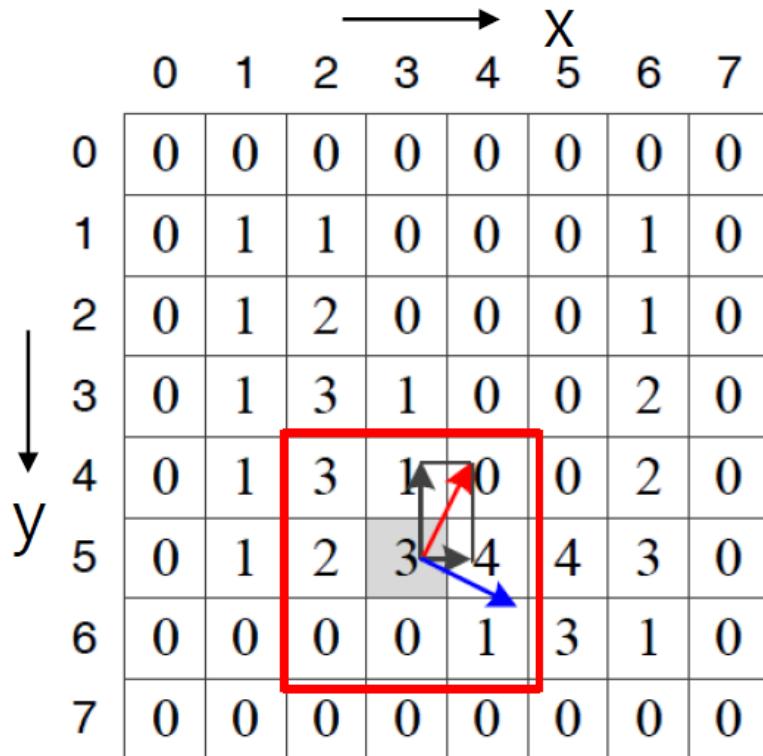


(d) d_x

Sobel edge detection

Sobel edge 검출 알고리즘

- Edge 방향은?
- 에지 방향을 양자화하면?



$$d_y = (0 \times 1 + 0 \times 2 + 1 \times 1) + (3 \times (-1) + 1 \times (-2) + 0 \times (-1)) = -4$$
$$d_x = (0 \times 1 + 4 \times 2 + 1 \times 1) + (3 \times (-1) + 2 \times (-2) + 0 \times (-1)) = 2$$

$$\rightarrow \nabla f = (d_y, d_x) = (-4, 2)$$

$$S(5,3) = ((-4)^2 + 2^2)^{\frac{1}{2}} = 4.47$$

$$D(5,3) = \arctan(-\frac{4}{2}) = -63.4^\circ$$

→ d_y 와 d_x

→ 그레이디언트 방향

→ 에지 방향

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Sobel edge detection

Sobel edge 검출 알고리즘

- 2_1_SobelEdge.py

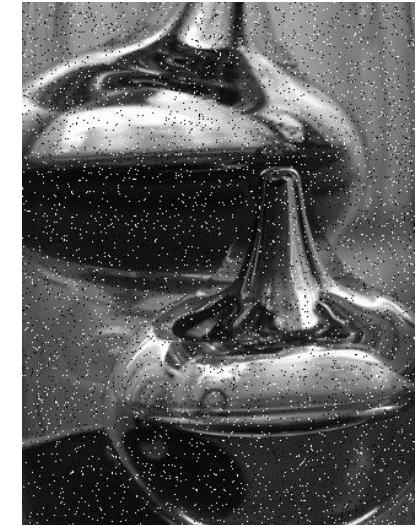
```
# Sobel 연산자 적용  
grad_x = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize = 3)  
grad_y = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize = 3)  
  
# 절댓값을 취해 양수 영상으로 변환  
sobel_x = cv2.convertScaleAbs(grad_x)  
sobel_y = cv2.convertScaleAbs(grad_y)  
  
edge_strength = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0) # Edge 강도 계산
```



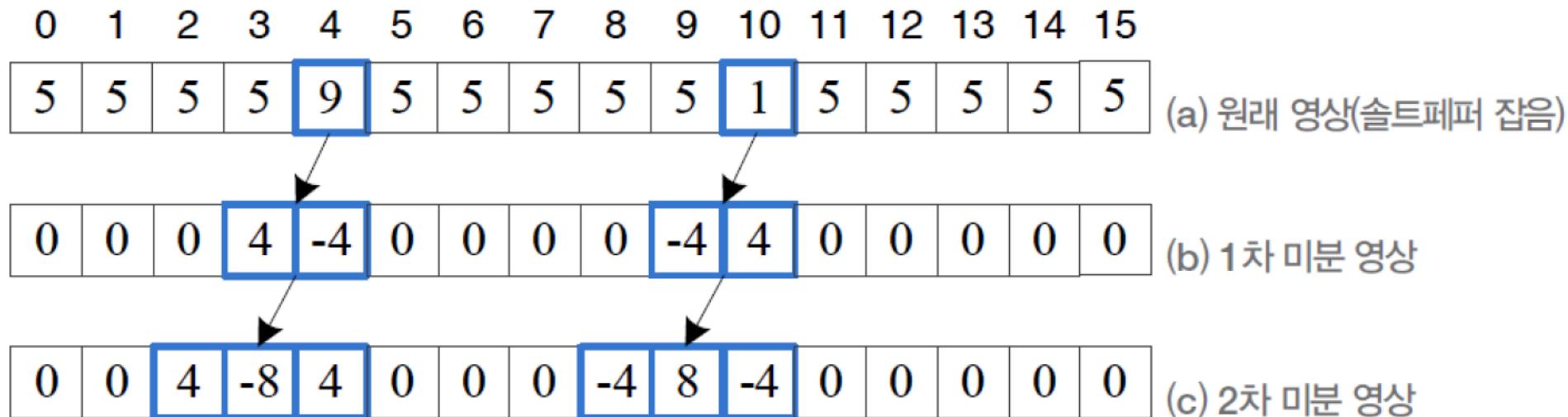
Zero-crossing (2차 미분) 이론

Gaussian과 다중 스케일 효과

- Why Gaussian?
 - 미분은 잡음을 증폭시키므로 smoothing의 적용이 중요함



점 잡음(Salt and pepper noise)

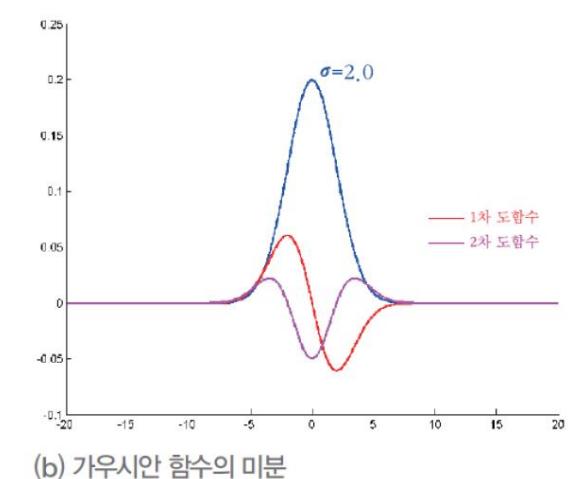
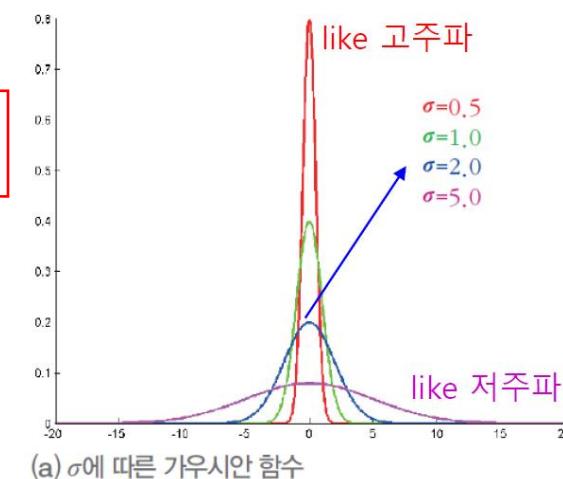
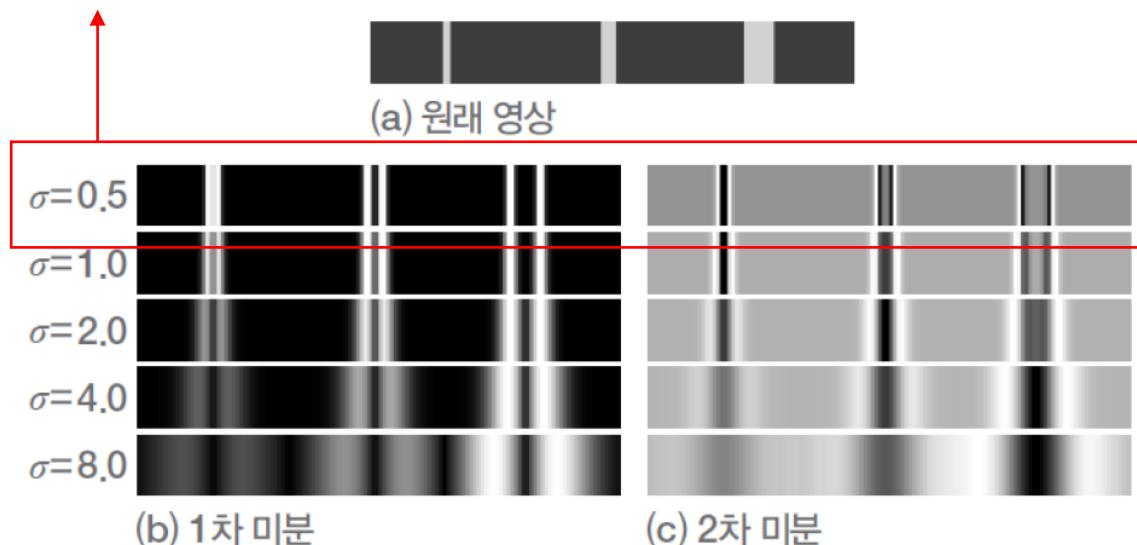


Zero-crossing (2차 미분) 이론

Gaussian과 다중 스케일 효과

- Why Gaussian?
 - σ 를 조절하여 다중 스케일 효과
 - 즉, σ 를 작게 하면 물체 디테일 edge, 크게 하면 큰 물체의 edge를 추출하기에 용이
 - σ 로 스케일을 조절하면 smoothing 정도를 조절 → edge 스케일 조절

Zero-crossing이 선명하게 나타남



Zero-crossing (2차 미분) 이론

LoG 필터

- Laplacian of Gaussian
- * Laplacian: 함수 f 의 각 parameter의 2차 편미분함수의 합

$$\begin{aligned}\nabla^2 f(y, x) &= \frac{\partial f^2}{\partial y^2} + \frac{\partial f^2}{\partial x^2} \\ &= (f(y+1, x) + f(y-1, x) - 2f(y, x)) + (f(y, x+1) + f(y, x-1) - 2f(y, x)) \\ &= f(y+1, x) + f(y-1, x) + f(y, x+1) + f(y, x-1) - 4f(y, x)\end{aligned}$$

→ 이에 해당하는 필터 $L =$

0	1	0
1	-4	1
0	1	0

Zero-crossing (2차 미분) 이론

LoG 필터

⊖:convolution

$$LoG(y, x) = \nabla^2(G(y, x) ⊖ f(y, x)) = (\nabla^2 G(y, x)) ⊖ f(y, x)$$

$$\text{such that } \nabla^2 G(y, x) = \left(\frac{y^2 + x^2 - 2\sigma^2}{\sigma^4} \right) G(y, x)$$

- Gaussian의 역할: 거리에 따라 가중치를 다르게 두는 것
- Laplacian의 역할: 2차 미분을 통해서 기울기의 변화량을 측정하는 것
- → LoG: Gaussian의 2차 미분
 - Gaussian을 통해 잡음 제거 후 Laplacian을 적용하는 두 단계를 한번에 하는 것
 - LoG 필터 적용 후 zero-crossing 부분을 찾은 후 threshold를 기준으로 경계선 인식

Zero-crossing (2차 미분) 이론

Marr-Hildreth edge detection algorithm

- [Theory of edge detection]
- Idea: 2차 미분에서 zero-crossing 검출하여 edge로 설정
- Zero-crossing을 활용한 edge detection
 - 한 점을 기준으로 여덟 개의 마주보는 화소 쌍의 부호를 조사
 - 동-서, 남-북, 북동-남서, 북서-남동
 - (이론상 조건) 마주보는 이웃이 서로 다른 부호를 가질 때
 - (현실적 조건) 두 개 이상이 다른 부호를 가질 때
 - 부호가 다른 쌍의 값 차이가 threshold를 넘는다 → edge

Zero-crossing (2차 미분) 이론

Marr-Hildreth edge detection algorithm

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

(a) 원래 영상과 LOG 필터를 적용한 영상 g

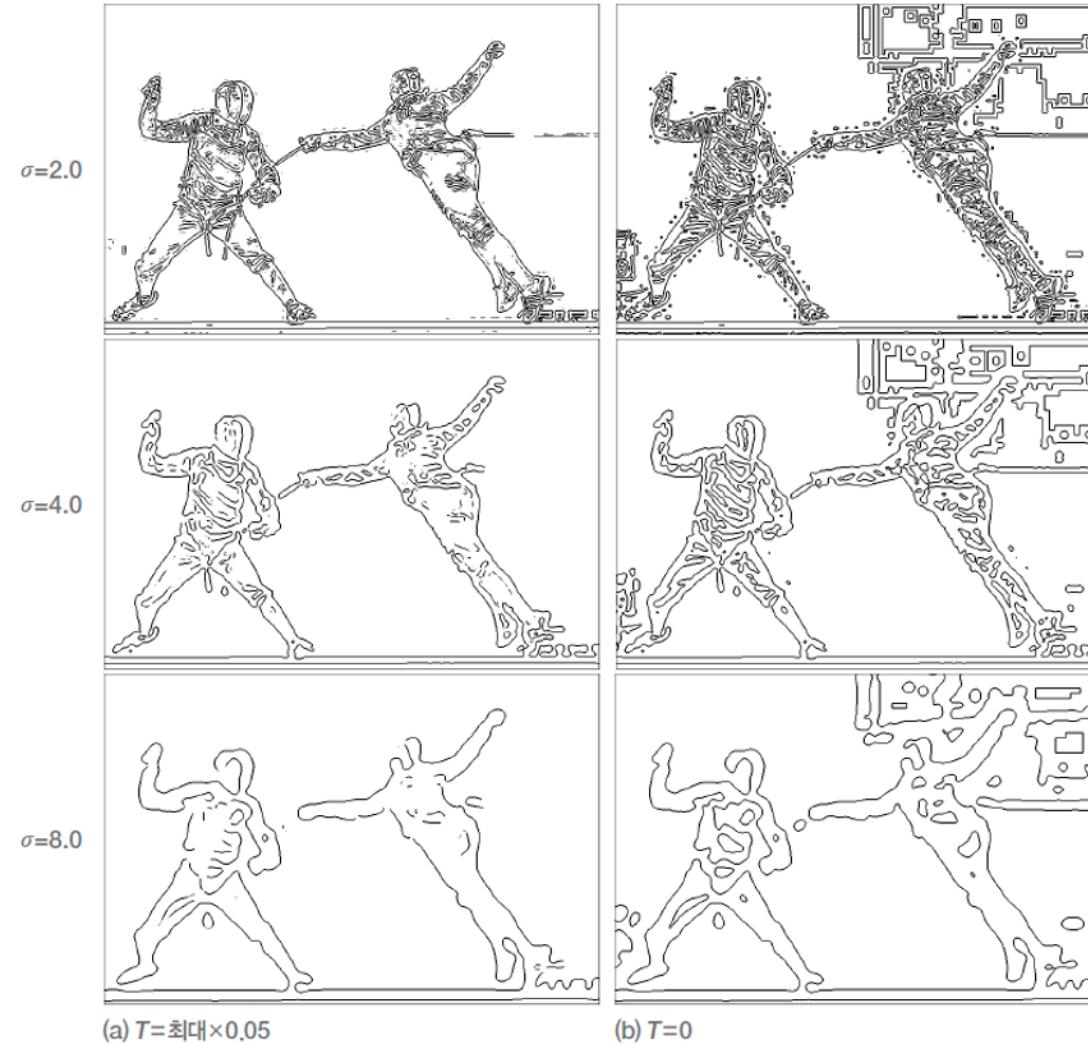
0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	1	1	1	0	1	0	0	
0	1	0	1	0	1	0	0	
0	1	1	1	1	1	0	0	
0	1	0	0	0	1	1	0	
0	0	0	1	1	1	1	0	
0	0	0	0	0	0	0	0	

(b) 영교차 검출($T=1.0$)

0.4038	1.2058	1.2058	0.4038	0	0.4038	0.8021	0.4038
1.2058	-2.4116	-2.0133	1.6096	0	1.2058	-4.0212	1.2058
1.6096	-0.0000	-4.4250	4.0212	0.4038	2.0133	-2.4171	2.0133
1.6096	1.2058	-7.6441	0.4038	1.2058	2.8154	-7.2404	2.8154
1.6096	1.2058	-6.4328	4.4250	7.2404	8.4462	-4.0212	3.6229
1.2058	-1.2058	-3.2246	-7.2404	-11.2616	-9.6574	-7.6441	3.6175
0.4038	1.6096	3.2191	5.5308	3.6175	-6.8311	1.6041	2.0133
0	0	0	0.4038	2.0133	3.2137	2.0133	0.4038

Zero-crossing (2차 미분) 이론

Marr-Hildreth edge detection algorithm 예시



Canny edge detection

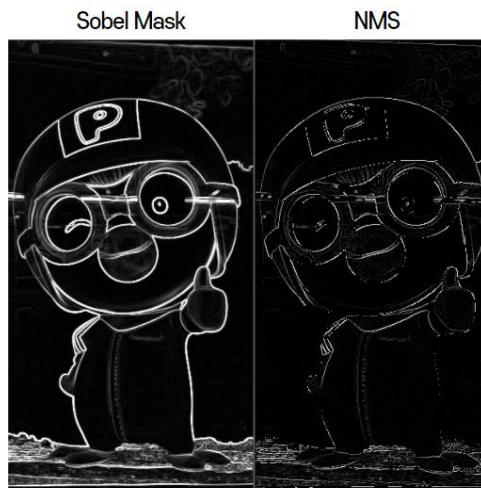
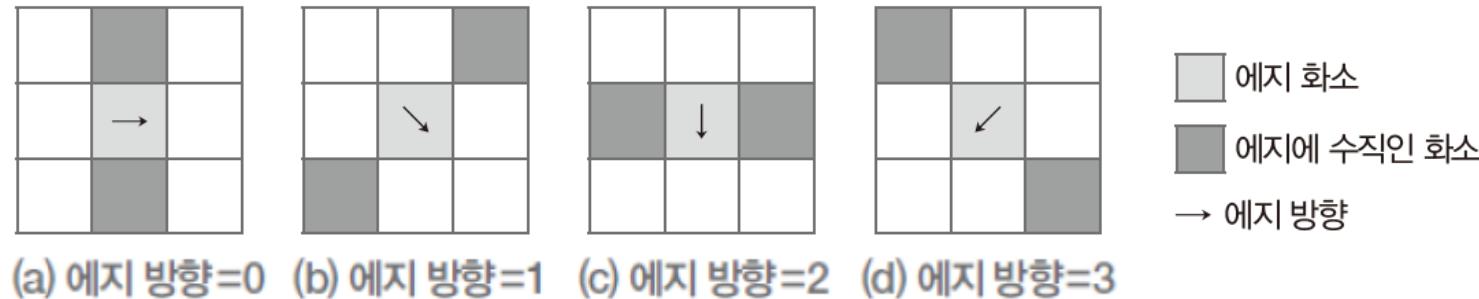
Canny edge

- Edge detection 문제를 최적화 문제로 품
 - Sobel → ‘그럴듯해 보이는’ edge 검출 접근 기법
- 세 가지 기준의 목적 함수 정의
 - 최소 오류율: False positive (FP), false negative (FN)을 최소로 함.
즉, 없는 edge가 생성되거나 edge를 못 찾는 경우를 최소로!
 - 위치 정확도: 검출된 edge는 실제 edge의 위치와 가까워야 함.
 - Edge 두께: 실제 edge에 해당하는 곳에는 한 두께의 edge만 생성
 - → Greedy 알고리즘에 기반한 “근사해” (approximation)를 찾는 방법으로 접근

Canny edge detection

Canny edge

- 비최대 억제 (Non-maximum suppression, NMS)
 - 이웃 두 화소보다 edge 강도가 크지 않으면 suppression



Canny edge detection

Canny edge

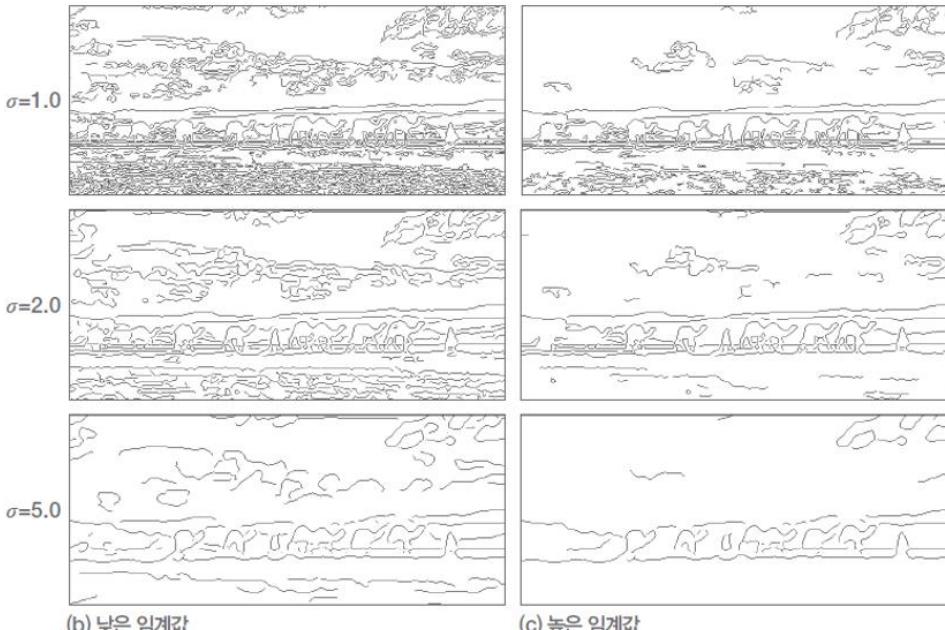
- 이력 임계값 (Hysteresis thresholding, T)
 - FP를 최소화하기 위해 두 개의 T_{high}, T_{low} 활용
 - 두 값을 활용하여 edge를 추적함
 - $s(y, x)$ 가 T_{high} 이상인 edge에서 추적 시작
 - 이후 추적은 T_{low} 이상인 edge를 대상으로 진행
 - → 이웃 화소가 추적 이력이 없으면 신뢰도가 낮더라도 edge로 간주
- $\sigma, T_{high}, T_{low}$ 가 edge의 품질을 결정하는 parameters
 - σ : Gaussian filter에서 standard deviation

Canny edge detection

Canny edge detection algorithm 예시



(a) 원래 영상(342×800)



→ 기본적으로 물체와 그림자의 edge를 구별하지 못함

Canny edge detection

Canny edge 검출 알고리즘

- 2_2_CannyEdge.py

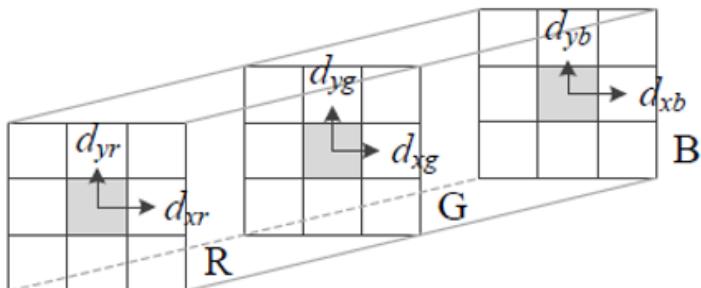
```
canny1 = cv2.Canny(gray, 50, 150) # T_low = 50, T_high = 150  
canny2 = cv2.Canny(gray, 100, 200) # T_low = 100, T_high = 200
```



Color edge

RGB 채널에 독립적으로 적용 후 OR 결합

- Edge 불일치 발생 가능성 높음



- 디 젠조 알고리즘

$$g_{yy} = (d_{yr})^2 + (d_{yg})^2 + (d_{yb})^2$$

$$g_{xx} = (d_{xr})^2 + (d_{xg})^2 + (d_{xb})^2$$

$$g_{yx} = d_{yr}d_{xr} + d_{yg}d_{xg} + d_{yb}d_{xb}$$

$$\text{그레이디언트 방향: } D(y,x) = \frac{1}{2} \arctan\left(\frac{2g_{yx}}{g_{xx} - g_{yy}}\right)$$

$$S(y,x) = \sqrt{0.5 \times ((g_{yy} + g_{xx}) + (g_{xx} - g_{yy})\cos(2D(y,x)) + 2g_{yx}\sin(2D(y,x)))}$$

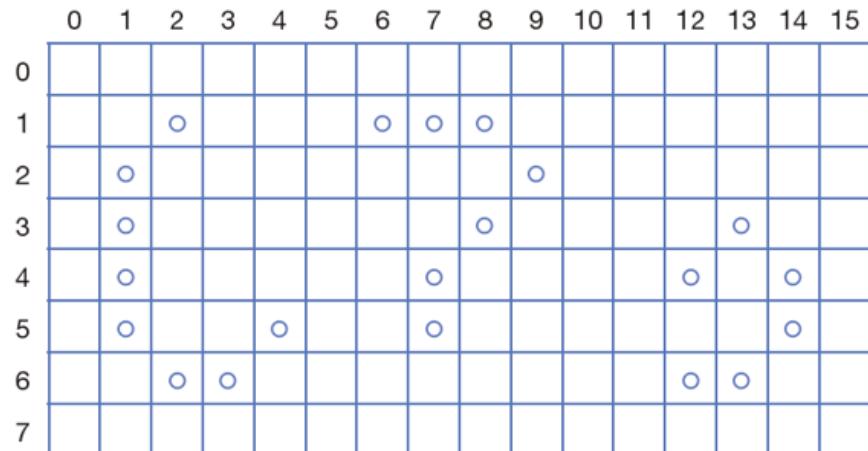
직선 검출

Edge를 명시적으로 연결하여 경계선 (Contour)을 찾고 직선(선분)으로 변환

- 이후 처리 단계인 물체 표현이나 인식에 유리
- Edge map → Edge 토막 → 직선화

경계선 (Contour) 찾기

8-connected edge 화소를 연결해 contour 구성

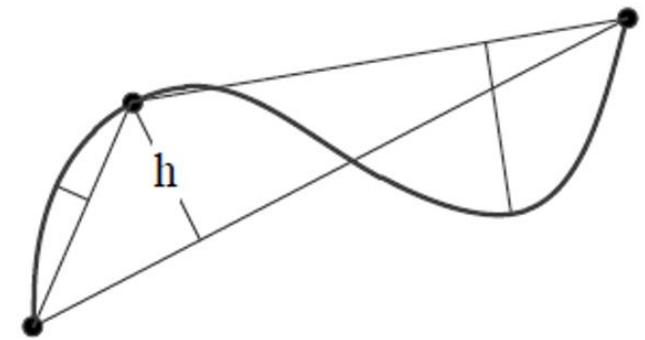


경계선1: (1,2)(2,1)(3,1)(4,1)(5,1)(6,2)(6,3)(5,4)

경계선2: (1,6)(1,7)(1,8)(2,9)(3,8)(4,7)(5,7)

경계선3: (4,12)(3,13)(4,14)(5,14)(6,13)(6,12)

- Line segment (선분 근사)
 - 두 끝점을 잇는 직선으로부터 가장 먼 점까지의 거리(h)가 임계값(T)이내가 될 때까지 재귀적으로 반복
 - T : line의 품질을 결정하는 parameter



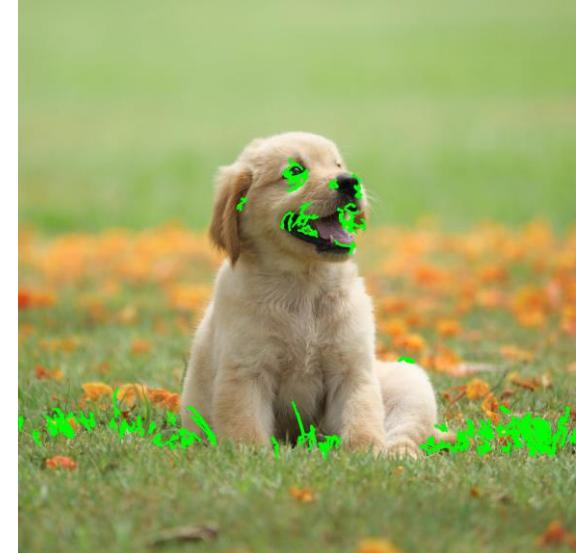
경계선 (Contour) 찾기

Edge map에서 경계선 찾기

- 2_3_contour.py

```
# parameter를 통해 여러가지 근사 방법 제공  
contour, hierarchy = cv2.findContours(canny,  
cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)
```

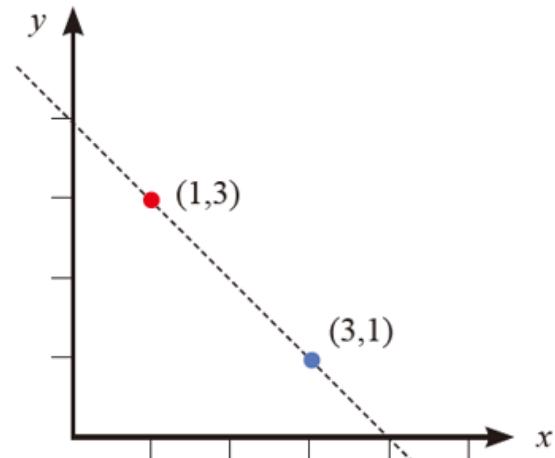
```
lcontour = []  
  
for i in range(len(contour)):  
    if contour[i].shape[0] > 100:  
        # 전체 길이가 100이상이면 하나의 line으로  
        lcontour.append(contour[i])  
  
cv2.drawContours(image, lcontour, -1, (0, 255, 0), 3)
```



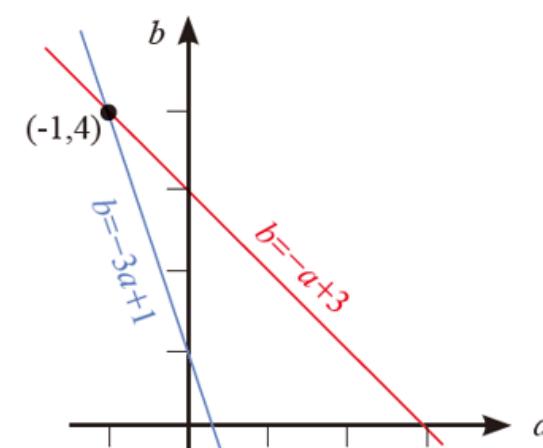
Hough transform (허프 변환)

Hough 변환은 끊긴 edge를 모아 직선/원 등을 검출하는 방법

- [Method and means for recognizing complex patterns, 1962]
- Pixel들 중에서 서로 직선 관계를 갖는 pixel들만 골라내보자!!
- 직선 검출 Idea
 - 각각의 점 (y_i, x_i) 에 대해 (b, a) 공간에 직선 $b = -ax_i + y_i$ 를 그림
 - (b, a) 공간에서 직선이 만나는 점을 절편과 기울기로 취함 \Rightarrow 교점 (a_1, b_1) 계산 (**투표**)



(a) (y, x) 로 표현되는 영상 좌표

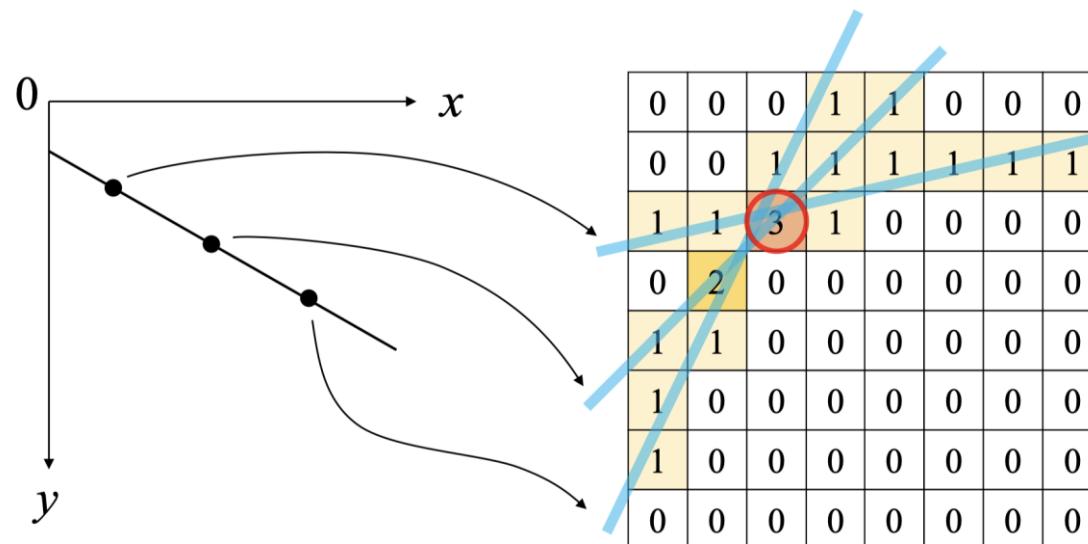


(b) (b, a) 로 표현되는 공간으로 매핑

Hough transform (허프 변환)

Hough 변환은 끊긴 edge를 모아 직선/원 등을 검출하는 방법

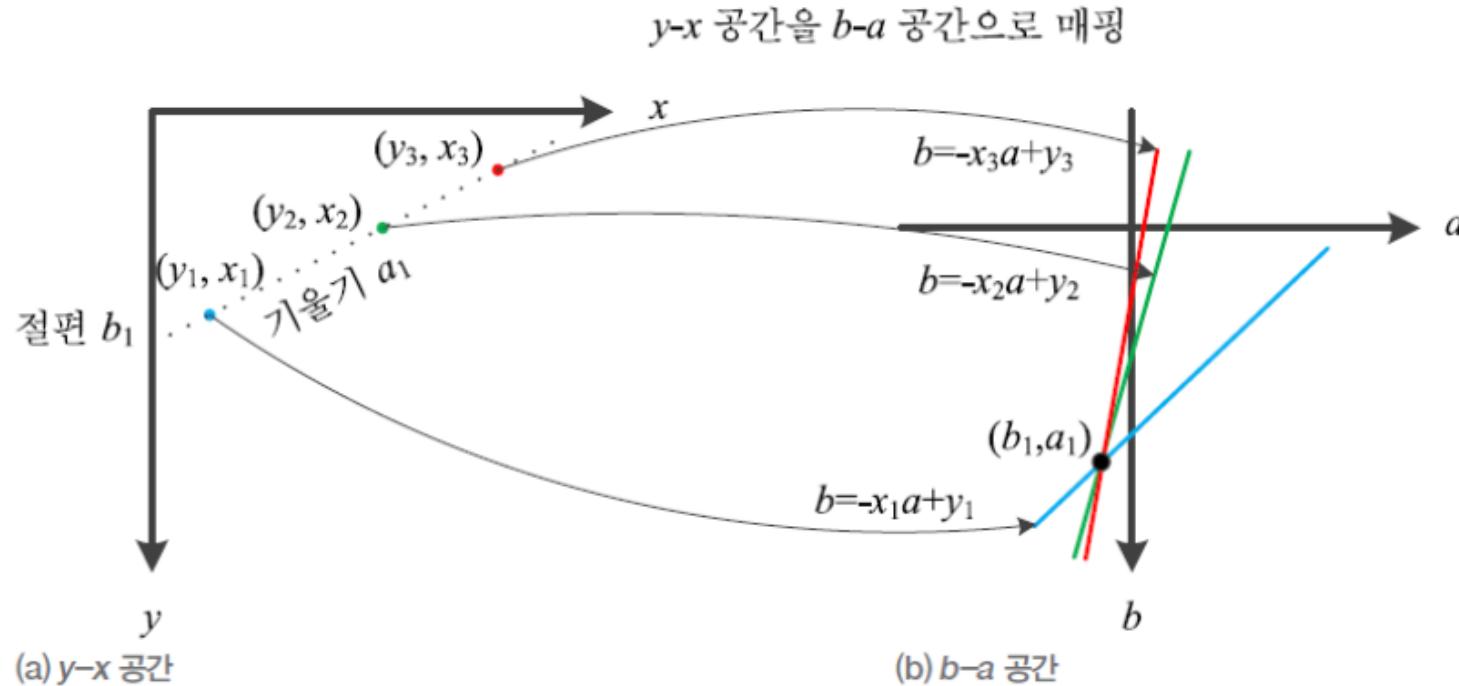
- Voting?
 - 직선들의 교점 (a_1, b_1) 에 겹치는 직선이 많을수록 $y = m_1x + b_1$ 이라는 직선이 존재할 가능성이 높아짐을 의미
 - ➔ (단순하게) 해당 교점 (a_1, b_1) 을 지나는 직선으면 검출에 활용



Hough transform

Edge 연결 과정 (filter, mask, window와 같은 local 연산) 없이 선분 검출

- → Global 연산(like histogram)을 이용한 지각 군집화 (perceptual grouping)
- 즉, 사람이 일직선상에 있다고 지각하는 점을 한 곳으로 모으는 원리
- 영상 공간 ($y-x$)을 기울기/절편 공간($b-a$)로 mapping



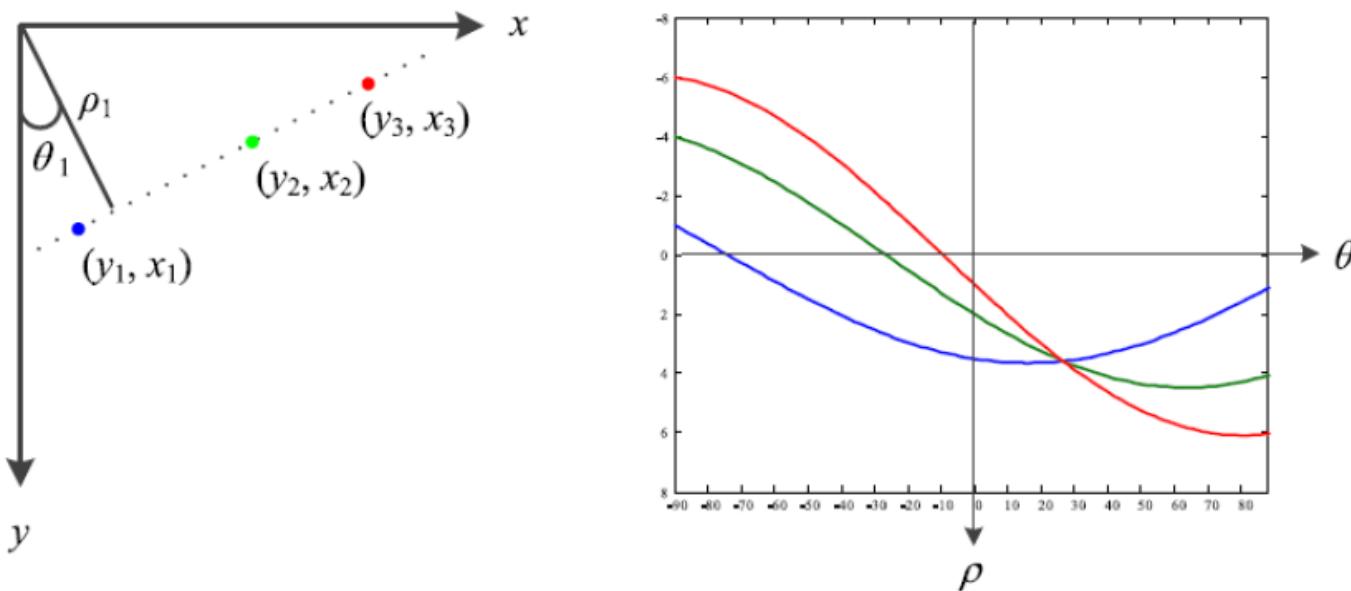
Hough transform

만약 수직선의 기울기가 infinite라면?

- 극좌표계 (Polar coordinate)를 활용하여 해결

$$y \cos \theta + x \sin \theta = \rho$$

y - x 공간을 ρ - θ 공간으로 매핑



Hough transform

구현

- (b, a) 공간을 이산화하고 누적 배열을 만들어 투표 기록
- 직선은 자신이 지나는 칸에 1만큼씩 투표
- 다수 표를 얻은 점을 결정할 때 NMS를 적용하여 지역 최대점을 찾음

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

Hough transform

Hough transform을 통해 직선 찾기

- 2_4_HoughLines.py

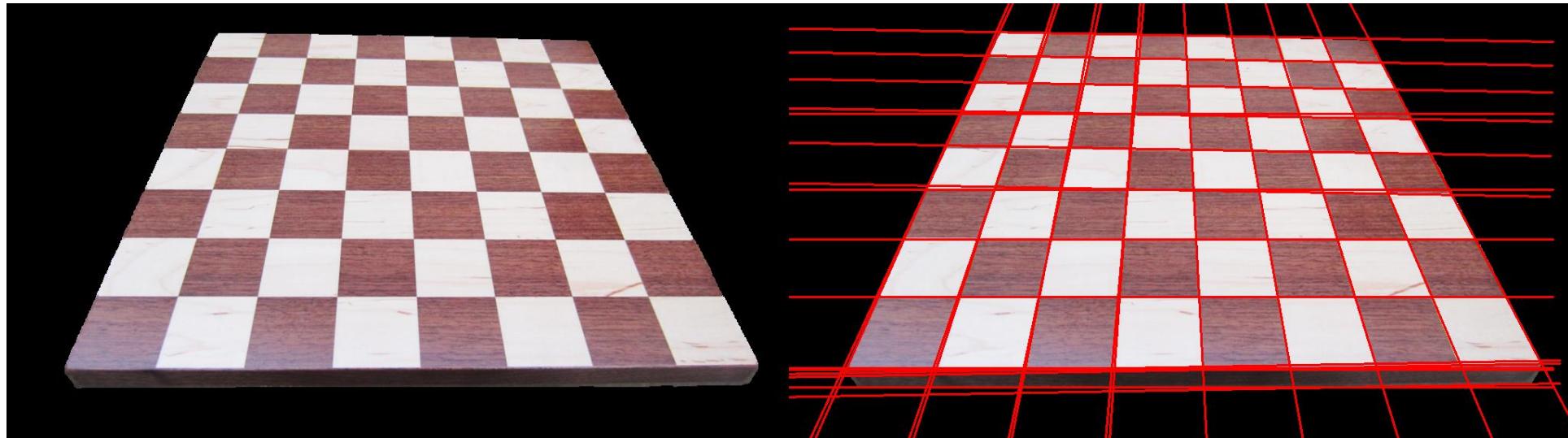
```
cv2.HoughLines(image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]]) → lines
```

image – 8bit, single-channel binary image, canny edge를 선 적용.

rho – r 값의 범위 (0 ~ 1 실수)

theta – θ 값의 범위(0 ~ 180 정수)

threshold – 만나는 점의 기준, 숫자가 작으면 많은 선이 검출되지만 정확도가 떨어지고, 숫자가 크면 정확도가 올라감



Hough transform

Hough transform을 통해 사과 검출하기

- Circle 검출 → 3차원 누적 배열 사용
- $(x - a)^2 + (y - b)^2 = r^2 \rightarrow (a, b, r)$ 에 대해 voting
- 2_5_HoughCircles.py
 - 함수 상세 사용법: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d



```
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 200,  
                           param1 = 150, param2 = 20,  
                           minRadius = 30, maxRadius = 100)  
  
for i in circles[0]:  
    cv2.circle(image, (int(i[0]), int(i[1])), int(i[2]), (255, 0, 0), 2)
```

2. Region

Preview

Region segmentation

- 영역 분할
- 물체가 점유한 영역을 구분하는 작업
- Human → 물체의 3차원 모델을 사용하여 주의 집중을 적용하여 의미 분할 수행
- 본 장에서는 명암/컬러 정보만으로 영역을 결정 → 의미 분할까지는 불가능

분할의 개념적 정의

$$r_i \cap r_j = \emptyset, \quad 1 \leq i, j \leq n, i \neq j$$

$$\bigcup_{i=1,n} r_i = f$$

$$Q(r_i) = \text{참}, \quad 1 \leq i \leq n$$

$$Q(r_i \cup r_j) = \text{거짓}, \quad 1 \leq i, j \leq n, \quad r_i \text{와 } r_j \text{는 이웃한 영역}$$

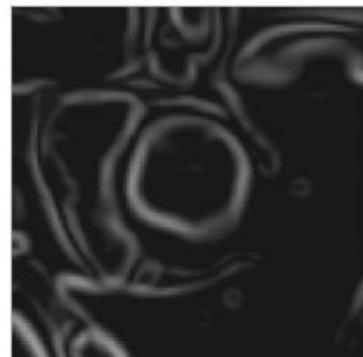
단순한 영상의 영역 분할

단순한 영상의 예

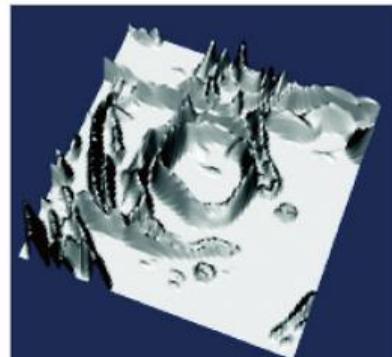
- 스캔한 책, 컨베이어 벨트 위를 흐르는 물체 영상 등

Region segmentation algorithm

- 이진화 알고리즘 적용
 - Ostu algorithm, clustering 알고리즘 등
- Watershed 알고리즘 적용



(a) 에지 강도 맵



(b) 지형으로 간주

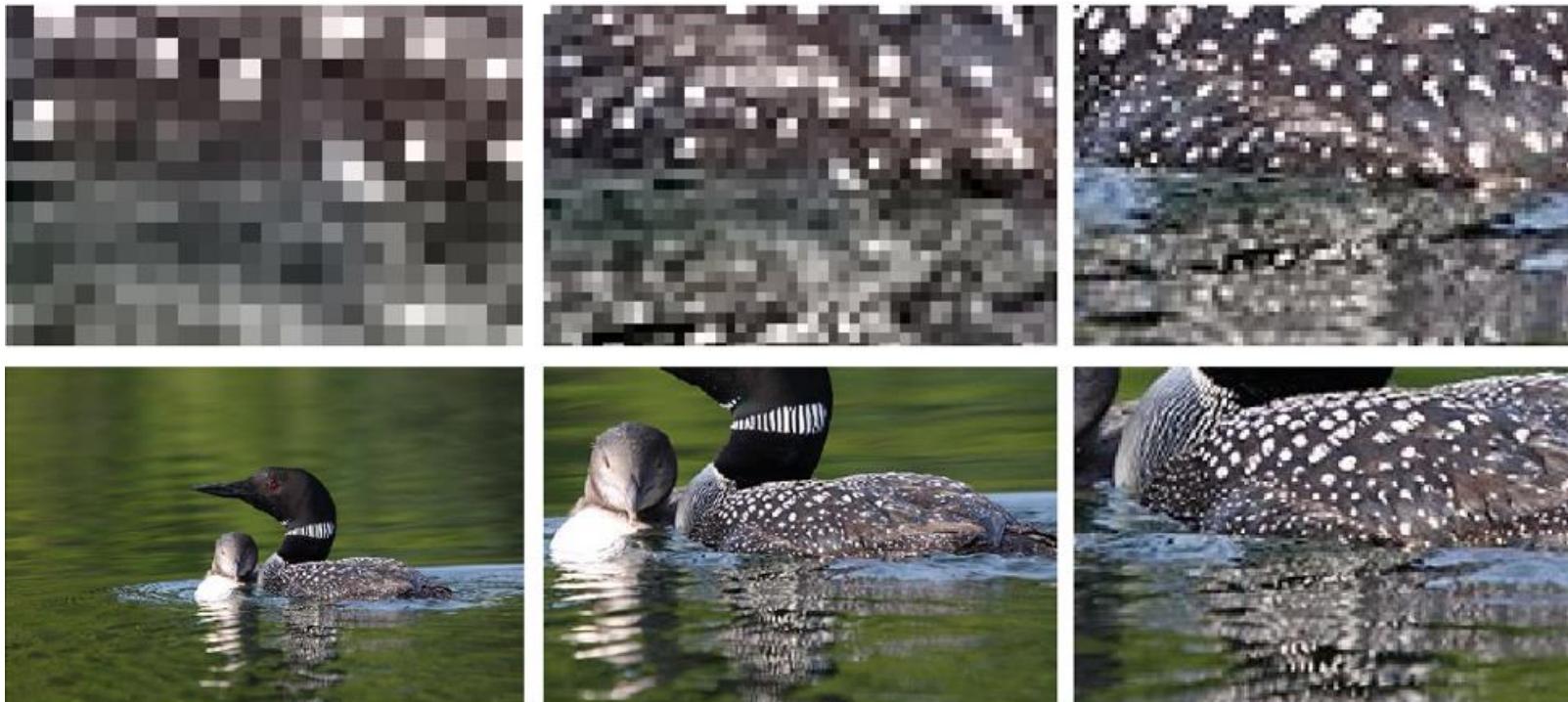


(c) 워터셰드

영상 분할의 원리

분할의 어려움

- 이웃 화소 몇 개를 보고 영역 내부 또는 외부를 판단할 수 있는가?
- → Global 연산의 필요성



영상 분할의 원리

Edge vs. Region

- 개념적으로 edge == region의 경계
- But, edge만으로는 region 검출에 한계
 - False Positive / False Negative → 폐곡선을 이루지 못함

여러가지 영상 분할 방법

- 임계화 기법
- 군집화
- 분할합병
- 그래프
- 최적화 기법 등등...
- → 2011년 이후로 별로 진전 없음...

임계화 기법

이진화를 통한 영역 분할

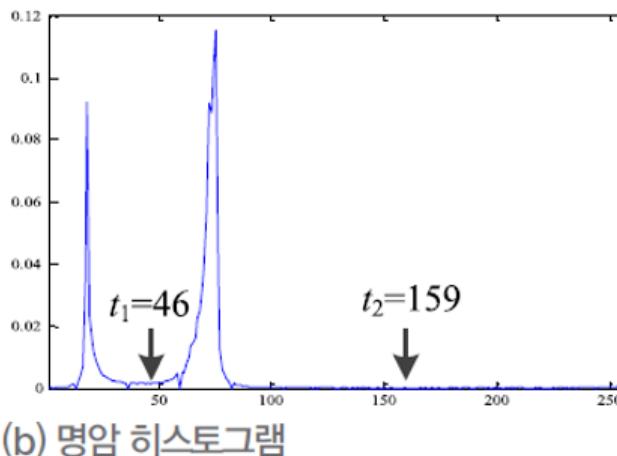
- Otsu algorithm
 - 명암단계가 둘 이상인 경우에 성능 떨어짐

삼진화로 확장

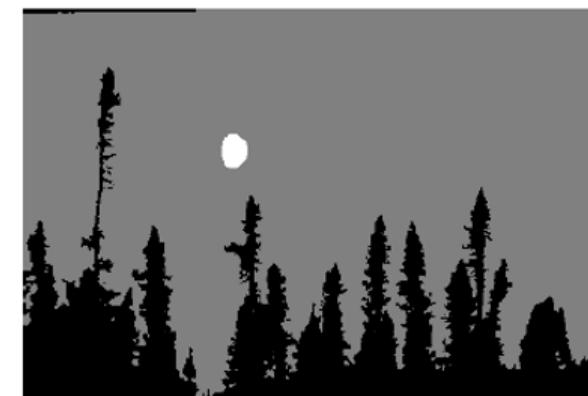
- 이중 임계값 t_1, t_2 사용



(a) 원래 영상



(b) 명암 히스토그램



(c) 이중 임계값으로 분할한 영상

임계화 기법

적응적 임계값 사용

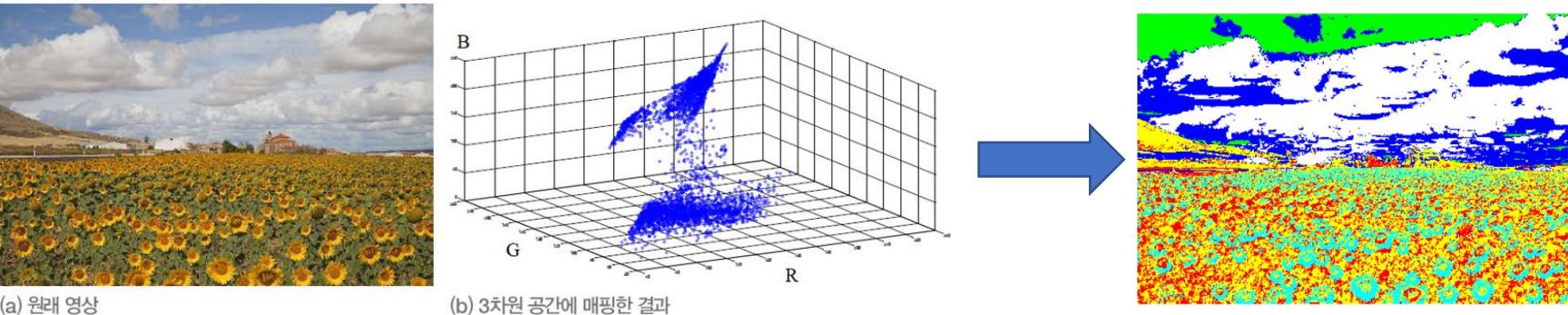
- 하나 또는 두 개의 임계값을 영상 전체에 적용하는 방법 (global 접근법)의 한계
- 지역적으로 명암 분포가 다른 경우 낮은 분할 품질
- → 지역에 따라 다른 임계값(적응적 임계값) 결정
 - Adaptive thresholding
 - Key: 어떻게 $t(j, i)$ 를 결정할 것인가??

$$b(j, i) = \begin{cases} 1, & f(j, i) \geq t(j, i) \\ 0, & \text{else} \end{cases}$$

군집화 기법

군집화를 활용한 영상 분할 기법

- Pixel을 RGB 3차원 컬러 공간으로 mapping한 후 k-means clustering algorithm 적용



- 슈퍼 화소 기법
 - 화소보다 크지만 물체보다 작은 세부 영역으로 분할하는 기법
 - SLIC (Simple Linear Iterative Clustering) algorithm
 - K-means clustering과 비슷하게 동작
 - 화소 할당 단계, 군집 중심 갱신 단계를 반복

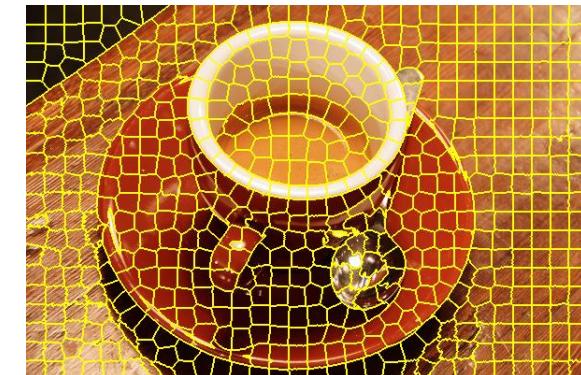


군집화 기법

SLIC algorithm으로 입력 영상을 슈퍼 화소 분할하기

- 2_6_SLIC.py
 - \$ pip install scikit-image 후 사용

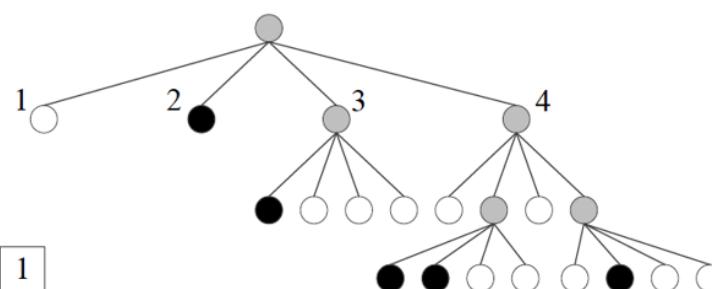
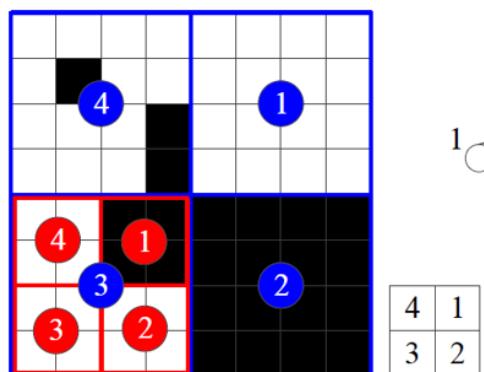
```
slic1 = skimage.segmentation.slic(image, compactness = 20, n_segments = 600)
sp_image1 = skimage.segmentation.mark_boundaries(image, slic1)
sp_image1 = np.uint8(sp_image1 * 255.0)
```



Divide-and-conquer (분할 합병)

원리

- 영역의 “균일성”을 측정하는 $Q(r_i)$ 를 이용하여 분할-합병 반복
 - $Q(r_i)$ 이 균일성 검증을 통과하지 못하면 → 네 개의 영역으로 분할
 - $Q(r_i \cup r_j)$ 이 균일하면 → r_i 와 r_j 를 합병하여 한 영역으로 취급
 - 재귀 반복
- 분할 결과는 4진 트리로 표현



Graph 방법

Graph

- $G = (V, E)$

- Node 집합 $V = \{v_1, v_2, \dots, v_n\}$

- Edge 집합 E

- 이웃 노드 간 edge 설정

- 두 노드 v_p 와 v_q 를 연결하는 edge의 가중치 w_{pq}

- 가중치는 거리 또는 유사도를 기반으로 측정

거리 $\begin{cases} d_{pq} = \|f(v_p) - f(v_q)\|, \text{ 만일 } v_q \in neighbor(v_p) \\ \infty, \text{ 그렇지 않으면} \end{cases}$

유사도 $\begin{cases} s_{pq} = D - d_{pq} \text{ 또는 } \frac{1}{e^{d_{pq}}}, \text{ 만일 } v_q \in neighbor(v_p) \\ 0, \text{ 그렇지 않으면} \end{cases}$

	0	1	2	3	4		v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	\dots	v_{23}	v_{24}	
0	v_0	v_1	v_2	v_3	v_4		0	4	-	-	-	0	-	-	-	-	-	-	-	-	-	-	
1	3	-4	7	-5	2	-0	2	-0	2	-0	2	0	1	7	6	1	-	-	-	-	-	-	
2	3	-3	6	-3	9	-1	8	-7	1	-	1	1	2	2	4	4	0	-	-	-	-	-	
3	2	-6	8	-1	7	-3	4	-3	1	-	1	1	7	3	1	3	-	-	-	-	-	-	
4	1	-0	1	-3	4	-1	5	-1	4	-	1	1	1	1	3	4	3	-	-	-	-	-	
5	2	-0	2	-1	1	-0	1	-0	1	-0	1	2	1	2	1	2	3	0	3	-	2	-	-
6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

(a) 입력 영상(화소를 잇는 값(파란색)은 에지 가중치)

(b) 인접 행렬 표현

Graph 방법

원리

- 유사도가 높은 node 쌍 같은 영역 (connected component)
 유사도가 낮은 node 쌍은 다른 영역에 배치
- 유사도가 낮은 edge가 분할선(경계선)이 될 가능성이 높음
- “가능성이 높다”
 - 지역적(Local)으로 유사도가 낮더라도 전역적(Global)으로 판단하면 아닐 수도 있음
 → Global 최적해의 중요성*

Global optimization 문제의 구현 시 요구사항

- 어떤 분할의 좋은 정도를 측정하는 목적 함수 → 분할 품질 관련
- 목적 함수를 최대/최소화 하는 최적해를 찾는 효율적인 알고리즘 → 처리 속도 관련

Graph 방법

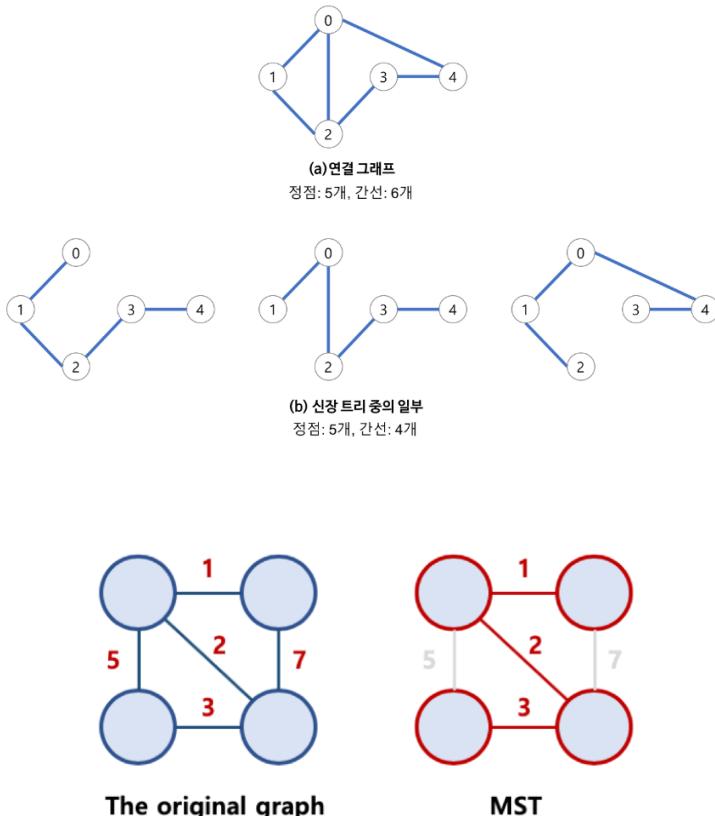
Global 최적 해의 중요성

- 기존 영역 분할 방법의 한계
 - 단순 영상, 슈퍼 화소 분할 등
 - → 지역적 명암 변화만을 활용함
 - 양말의 색이 배경과 비슷하면 양말이 배경영역에 섞임
- 전역적 정보를 고려한 문제 해결 방법 필요
 - 지역적으로 색상 변화가 약하지만 전역적으로 유리하다면 물체 경계로 간주
 - 영상을 그래프로 표현하고 최적화 알고리즘으로 분할 문제를 해결함

Minimum Spanning Tree (최소 신장 트리)

* MST란

- 모든 노드를 연결하면서 tree의 조건을 만족하는 그래프 중에서 edge간 가중치(weight)의 합이 최소인 spanning tree



v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}	v_{21}	v_{22}	v_{23}	v_{24}	
3	-4-	7	-5-	2	-0-	2	-0-	2	1	3	-3-	6	-3-	9	-1-	8	-7-	1	1	2	2	4	0	0	
0	1	7	1	6	7	1	7	6	1	3	2	6	3	9	8	7	1	1	2	2	4	0	0	0	
v ₅	v ₆	v ₇	v ₈	v ₉	v ₁₀	v ₁₁	v ₁₂	v ₁₃	v ₁₄	v ₁₅	v ₁₆	v ₁₇	v ₁₈	v ₁₉	v ₂₀	v ₂₁	v ₂₂	v ₂₃	v ₂₄	v ₂₅	v ₂₆	v ₂₇	v ₂₈		
3	-3-	6	-3-	9	-1-	8	-7-	1	1	2	2	2	2	4	0	0	0	0	0	0	0	0	0	0	0
1	2	1	2	1	1	2	1	2	1	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2

$$MST(C) = \{(v_1, v_6), (v_6, v_{11}), (v_{11}, v_{12}), (v_{12}, v_7)\}$$

MST를 이용한 영상 분할

원리

- 거리를 weight로 갖는 graph 생성
- Edge 를 오름차순 정렬
- 초기 분할 수행 (1 pixel = 연결 요소 1개)
- 모든 edge에 대해 균일성 측정을 반복하여 계산
 - (두 연결 요소 간 거리) < (두 연결요소 간 균일성) \rightarrow 하나의 연결 요소

MST를 이용한 영상 분할

연결 요소 C 의 균일성 측정

$$intra(C) = \max_{e \in MST(C)} w_e$$

- 위 그림에서 $intra(C) = 2$

v_0	v_1	v_2	v_3	v_4
3	-4-	7	-5-	2
0	1	7	6	1
3	-3-	6	-3-	9
1	2	2	2	4
2	-6-	8	-1-	7
1	7	3	1	3
1	-0-	1	-3-	4
1	1	3	4	3
2	-0-	2	-1-	1
		...	v_{23}	v_{24}
			4	1
			-0-	1

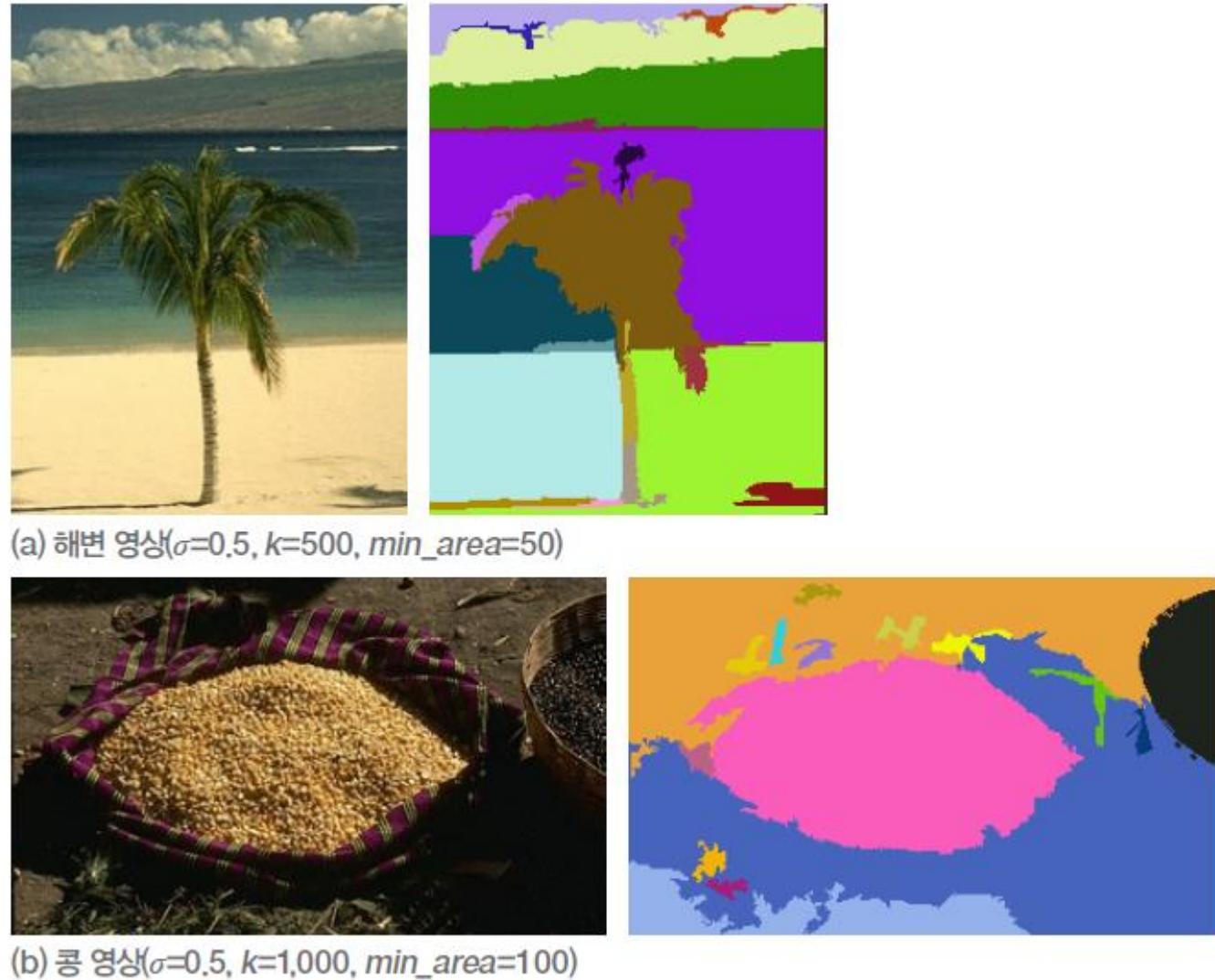
두 연결요소의 균일성 측정

$$multiIntra(C_i, C_j) = \min \left(intra(C_i) + \tau(C_i), intra(C_j) + \tau(C_j) \right); \tau(C) = \frac{k}{|C|}$$

- 새로운 노드 추가 시
 - (모든 인접 노드와 비교) $v_0, v_5, v_{10}, v_{16}, v_{17}, v_{13}, v_8, v_2$
 - 각 인접 노드를 추가했을 때 $multiIntra(C_i, C_j)$ 계산 $\rightarrow v_8$ 추가 \rightarrow 반복

MST를 이용한 영상 분할

MST를 이용한 영상 분할 예시



Normalized cut을 통한 영상 분할

Normalized cut algorithm

- MST 기반의 영상 분할 방법은 C1, C2간 거리만을 활용함
- “cut”은 영상을 두 영역 (C1, C2)으로 분할했을 때, 분할의 좋은 정도를 측정하는 목적 함수
 - 유사도를 활용

$$cut(C_1, C_2) = \sum_{v_p \in C_1, v_q \in C_2} s_{pq}$$

- C1과 C2가 커질수록 둘 사이에 edge가 많아짐 → cut이 덩달아 커짐
→ 영상 분할 시 작은 크기의 연결요소로 세밀하게 분할되는 경향
- “ncut”: cut을 정규화 하여 영역의 크기에 중립이 되게 함

$$ncut(C_1, C_2) = \frac{cut(C_1, C_2)}{cut(C_1, C)} + \frac{cut(C_1, C_2)}{cut(C_2, C)}$$

거리 $\begin{cases} d_{pq} = \|f(v_p) - f(v_q)\|, & \text{만일 } v_q \in neighbor(v_p) \\ \infty, & \text{그렇지 않으면} \end{cases}$

유사도 $\begin{cases} s_{pq} = D - d_{pq} \text{ 또는 } \frac{1}{e^{d_{pq}}}, & \text{만일 } v_q \in neighbor(v_p) \\ 0, & \text{그렇지 않으면} \end{cases}$

Normalized cut을 통한 영상 분할

Normalized cut algorithm으로 영역 분할하기

- 2_7_NormalizedCut.py

```
slic = skimage.segmentation.slic(image, compactness = 20, n_segments = 600, start_label = 1)

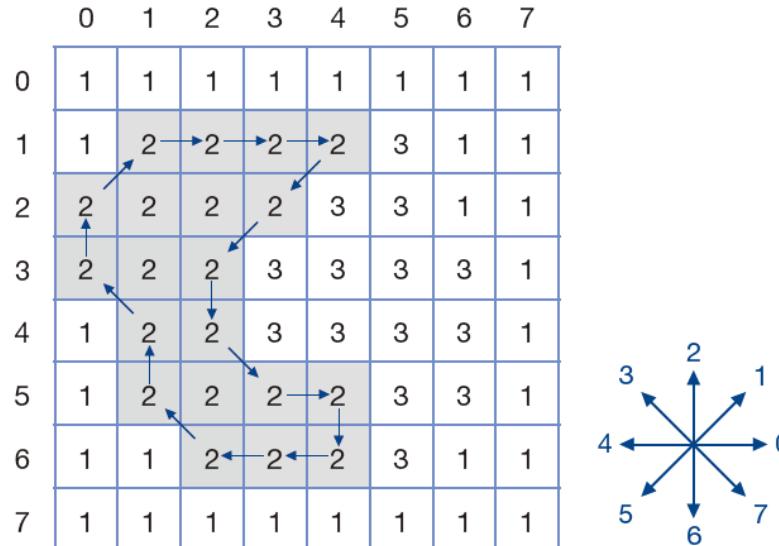
g = skimage.graph.rag_mean_color(image, slic, mode = 'similarity')
ncut = skimage.graph.cut_normalized(slic, g)
print(image.shape, 'Coffee 영상 분할 처리 시간: ', time.time() - start , '초')

marking = skimage.segmentation.mark_boundaries(image, ncut)
ncut_image = np.uint8(marking * 255.0)
```

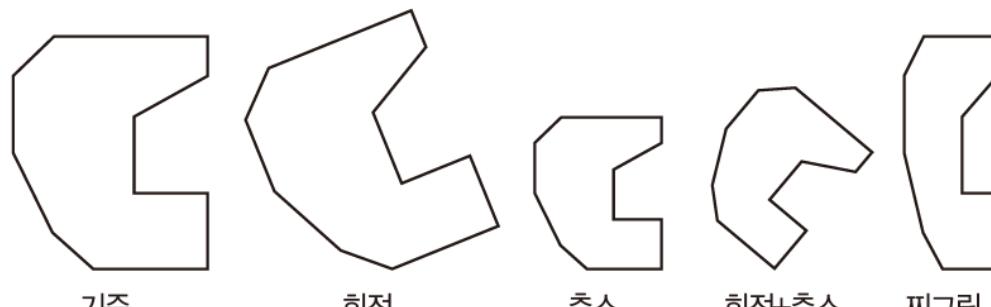


영역 특징

영역의 labeling과 기하 변환 예시



(a) 영역의 레이블링



(b) 영역의 기하 변환

영역 특징

특징의 Invariant / Equivariant

- 변환을 해도 값이 변하지 않으면 불변성(Invariant)이 있는 특징
- 변환에 따라 값이 변하면 등변성(Equivariant)이 있는 특징
 - Example
 - 성별: 나이에 불변
 - 면적: 회전에 불변, 축소에는 등변
 - 주축: 축소에 불변, 회전에는 등변

주어진 문제에 맞는 특징을 선택하는 것이 중요

영역 특징

Moment / Shape

- 영역 R 의 moment

$$m_{qp}(R) = \sum_{(y,x) \in R} y^R x^R$$

- Moment-based features

면적: $a = m_{00}$

중점: $(\dot{y}, \dot{x}) = \left(\frac{m_{10}}{a}, \frac{m_{01}}{a} \right)$

$$\mu_{qp} = \sum_{(y,x) \in R} (y - \dot{y})^q (x - \dot{x})^p$$

열 분산: $\nu_{cc} = \frac{\mu_{20}}{a}$

행 분산: $\nu_{rr} = \frac{\mu_{02}}{a}$

열행 분산: $\nu_{rc} = \frac{\mu_{11}}{a}$

$$\eta_{qp} = \frac{\mu_{qp}}{\mu_{00}^{\left(\frac{q+p}{2}+1\right)}}$$

영역 특징

영역의 둘레와 둑근 정도

$$\text{둘레: } p = n_{even} + n_{odd} \sqrt{2}$$

$$\text{둥근 정도: } r = \frac{4\pi a}{p^2}$$

주축의 방향

$$\text{주축의 방향: } \theta = \frac{1}{2} \arctan \left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \right)$$

영역 특징

Texture 특징

- Texture는 일정한 패턴의 반복
- Edge 통계량으로 texture 특징을 측정

$$T_{edge} = (busy, mag(i), dir(j)), 0 \leq i \leq q-1, 0 \leq j \leq 7$$

- LBP (Local Binary Pattern), LTP (Local Ternary Pattern)
 - LBP: 중심 화소와 주위 화소의 명암을 비교해서 texture 측정
 - LTP: 중심 화소에서 t를 기준으로 +/- 기준에 따라 texture 측정

137	115	92	95	101
141	122	99	98	102
145	129	105	99	103
146	135	116	97	100
146	138	123	96	99

(a) LBP 계산

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

11100001=225

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

i=10

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

11000001=193
00011100=28

(b) LTP 계산

영역 특징

이진 영역에서 특징 추출 예시

- 2_8_regionFeature.py

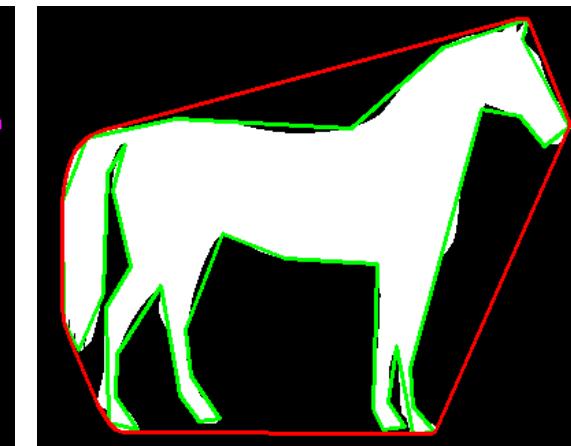
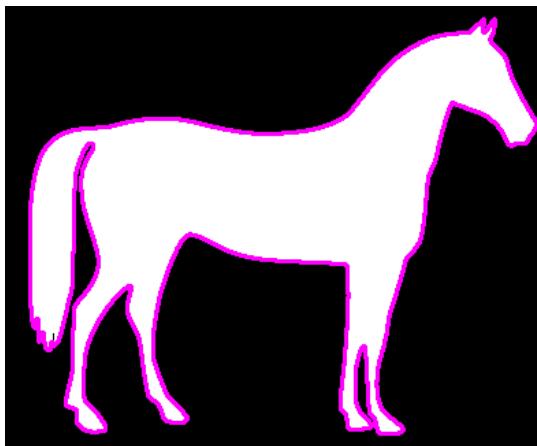
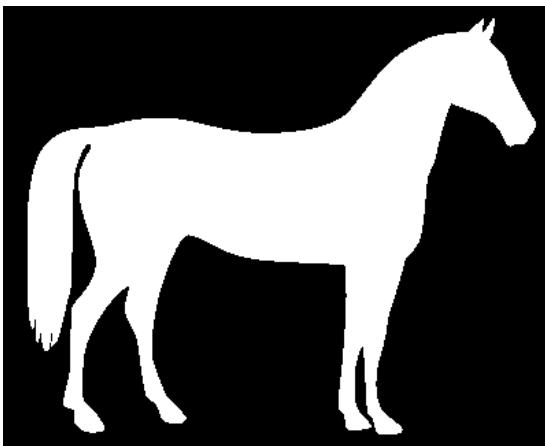
```
m = cv2.moments(contour)
```

```
area = cv2.contourArea(contour)
```

```
cx, cy = m['m10'] / m['m00'], m['m01'] / m['m00']
```

```
perimeter = cv2.arcLength(contour, True)
```

```
roundness = (4.0 * np.pi * area) / (perimeter * perimeter)
```



```
Area = 42390.0
Center Point = ( 187.72464024534088 , 144.43640402610677 )
Perimeter = 2296.7291333675385
Roundness = 0.1009842680321435
```

End of slide