

보충 자료

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(gradiant)의 강도와 방향 검출 (소벨 마스크)
3. 비최대치 억제(non-maximum suppression)
4. 이력 임계값(hysteresis threshold)으로 에지 결정

7.2.5 캐니 에지 검출

◆1) 블러링 - 5×5 크기의 가우시안 필터 적용

- ❖ 불필요한 잡음 제거 및 필터 크기는 변경가능

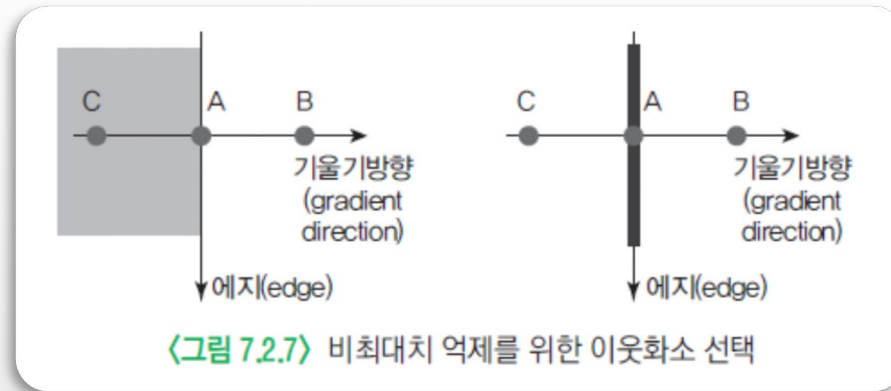
◆2) 화소 기울기(gradient) 검출

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

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
3	4	5
6	7	8

기울기 방향: 0

0	1	2
3	4	5
6	7	8

기울기 방향: 45

0	1	2
3	4	5
6	7	8

기울기 방향: 90

0	1	2
3	4	5
6	7	8

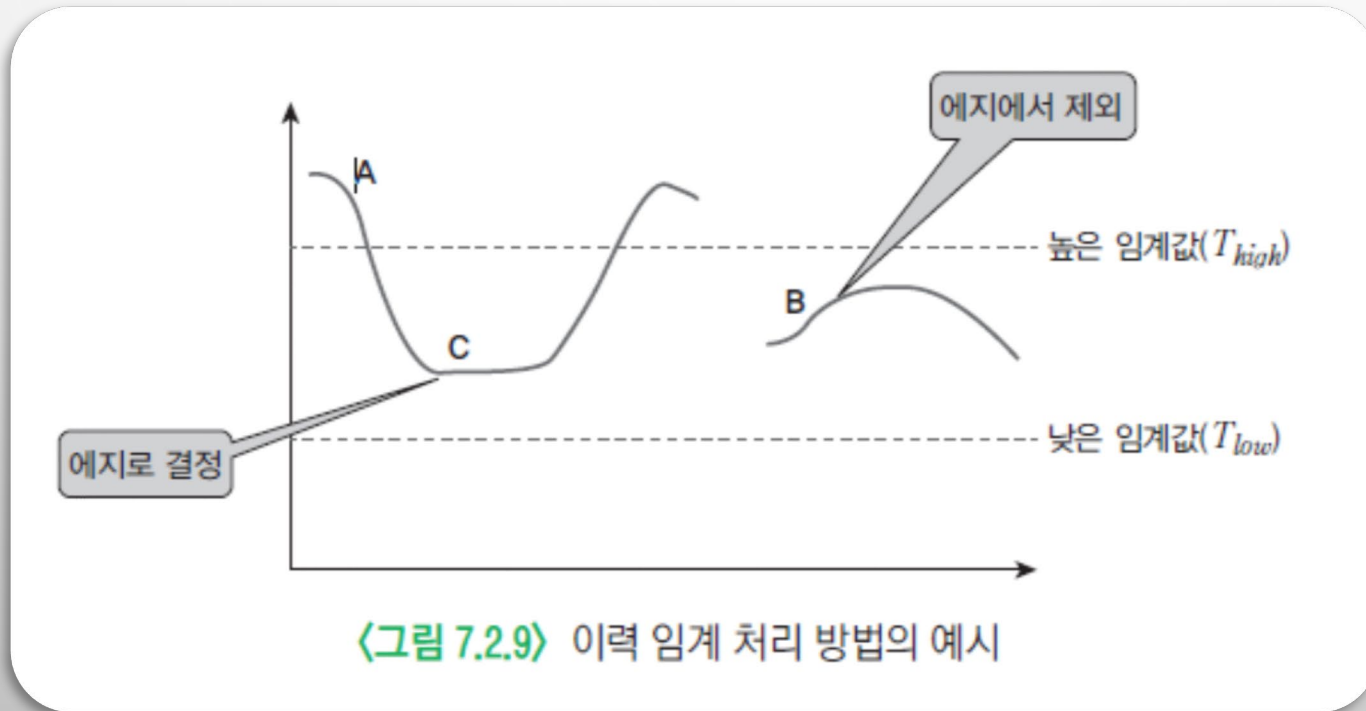
기울기 방향: 135

〈그림 7.2.8〉 비최대치 억제를 위한 이웃 화소 선택

7.2.5 캐니 에지 검출

◆4) 이력 임계값 방법(hysteresis thresholding)

- ❖ 두 개의 임계값(T_{high} , T_{low}) 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



7.2.5 캐니 에지 검출

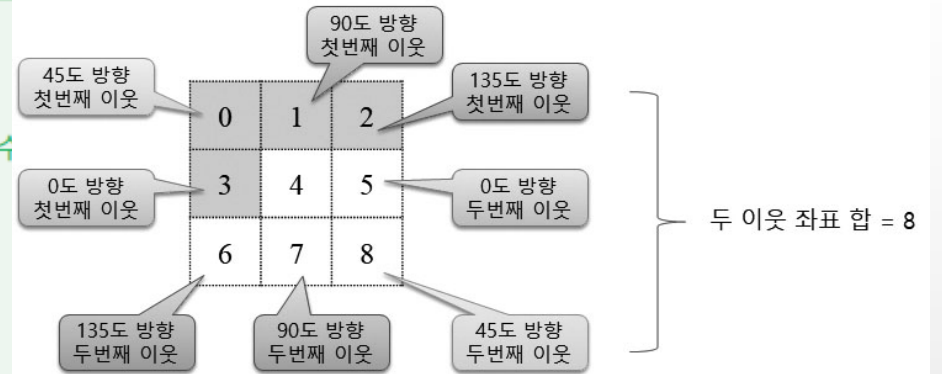
예제 7.2.8

캐니 에지 검출 - 08.edge_canny.py

```
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 참조 통해 이웃 화소 가져오기
09             values = sobel[i-1:i+2, j-1:j+2].flatten() # 중심 에지 주변 9개 화소 가져옴
10             first = [3, 0, 1, 2] # 첫 이웃 화소 좌표 4개
11             id = first[direct[i, j]] # 방향에 따른 첫 이웃화소 위치
12             v1, v2 = values[id], values[8-id] # 두 이웃 화소 가져옴
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
```

기울기 방향에 따라
두 개의 이웃 화소 선택

에지 화소 3x3 범위 1차원 전개



중심화소가 두 이웃화소
보다 작으면 억제

7.2.5 캐니 에지 검출

```
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:            # 추적 조건 확인
31         pos_ck[i, j] = 255                                    # 추적 좌표 완료 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):                               # 에지 영상 순회
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
48
49 image = cv2.imread("images/canny.jpg", cv2.IMREAD_GRAYSCALE)
```

재귀 호출로 8방향으로
추적 수행

7.2.5 캐니 에지 검출

```
49 image = cv2.imread("images/canny.jpg", cv2.IMREAD_GRAYSCALE)
50 if image is None: raise Exception("영상파일 읽기 오류")
51
52 pos_ck = np.zeros(image.shape[:2], np.uint8)           # 추적 완료 점검 행렬
53 canny = np.zeros(image.shape[:2], np.uint8)           # 캐니 에지 행렬
54
55 ## 캐니 에지 검출
56 gaus_img = cv2.GaussianBlur(image, (5, 5), 0.3)
57 Gx = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 1, 0, 3)   # x방향 마스크
58 Gy = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 0, 1, 3)   # y방향 마스크
59 sobel = cv2.magnitude(Gx, Gy)                             # 두 행렬 벡터 크기
60
61 directs = cv2.phase(Gx, Gy) / (np.pi/4)                 # 에지 기울기 계산 및 근사
62 directs = directs.astype(int) % 4                       # 8방향 → 4방향 축소
63 max_sobel = nonmax_suppression(sobel, directs)           # 비최대치 억제
64 hysteresis_th(max_sobel, 100, 150)                     # 이력 임계값
65
66 canny2 = cv2.Canny(image, 100, 150)                     # OpenCV 캐니 에지 검출
67
68 cv2.imshow("image", image)
69 cv2.imshow("canny", canny)
70 cv2.imshow("OpenCV_Canny", canny2)
71 cv2.waitKey(0)
```

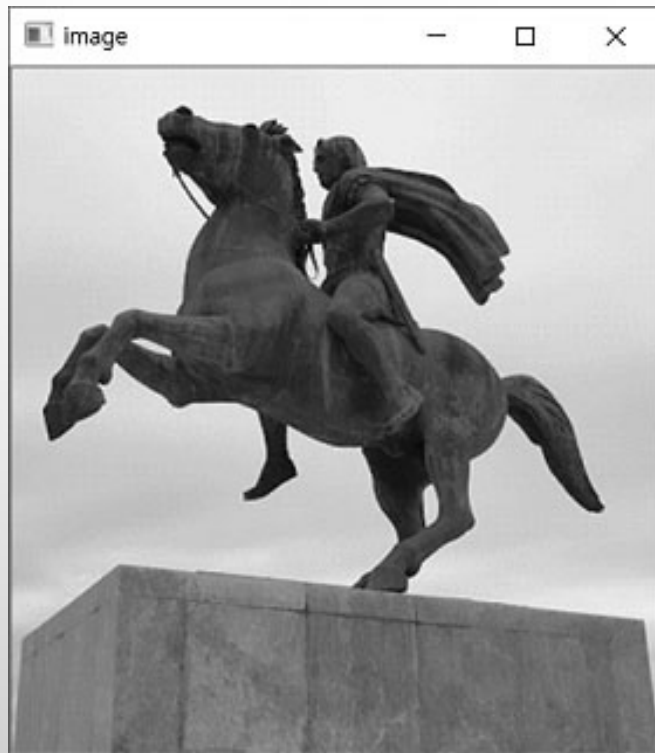
표준 편차

높은 임계값

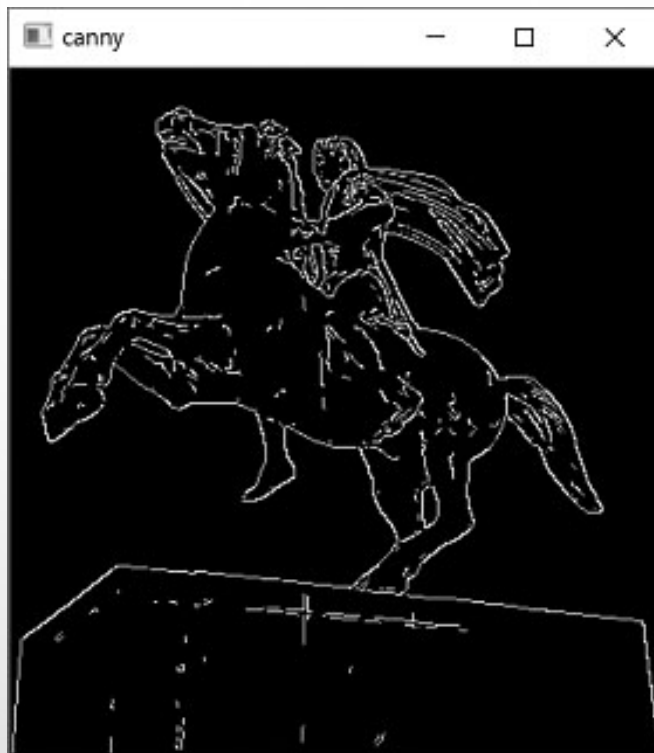
낮은 임계값

7.2.5 캐니 에지 검출

◆ 실행결과



입력영상



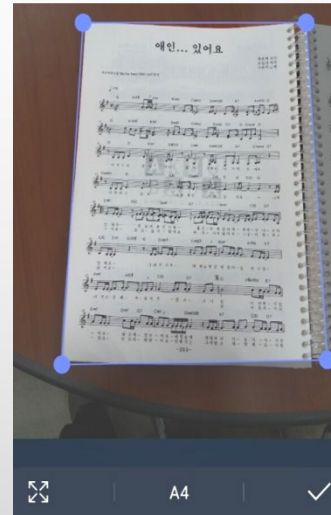
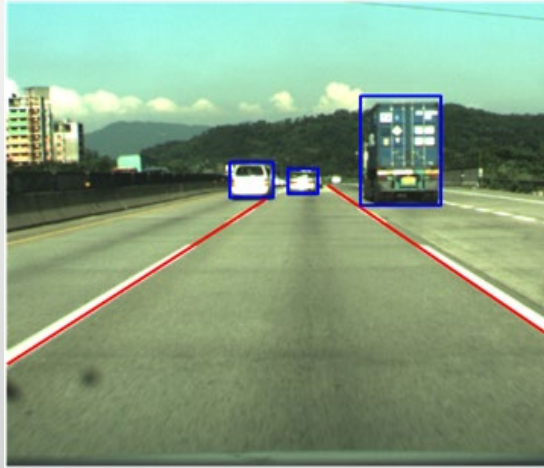
10.1 허프 변환

◆ 직선 검출

- ❖ 영상 내에서 공간 구조를 분석하는데 유용한 도구
- ❖ 영상 처리와 컴퓨터 비전분야에서 많은 연구 진행

◆ 다양한 응용에 사용

- ❖ 차선 및 장애물 자동인식 시스템 - 차선 검출
- ❖ 스캐너의 기능을 대신해 주는 앱 - 네 개 모서리 검출



10.1.1 허프 변환의 좌표계

◆허프변환

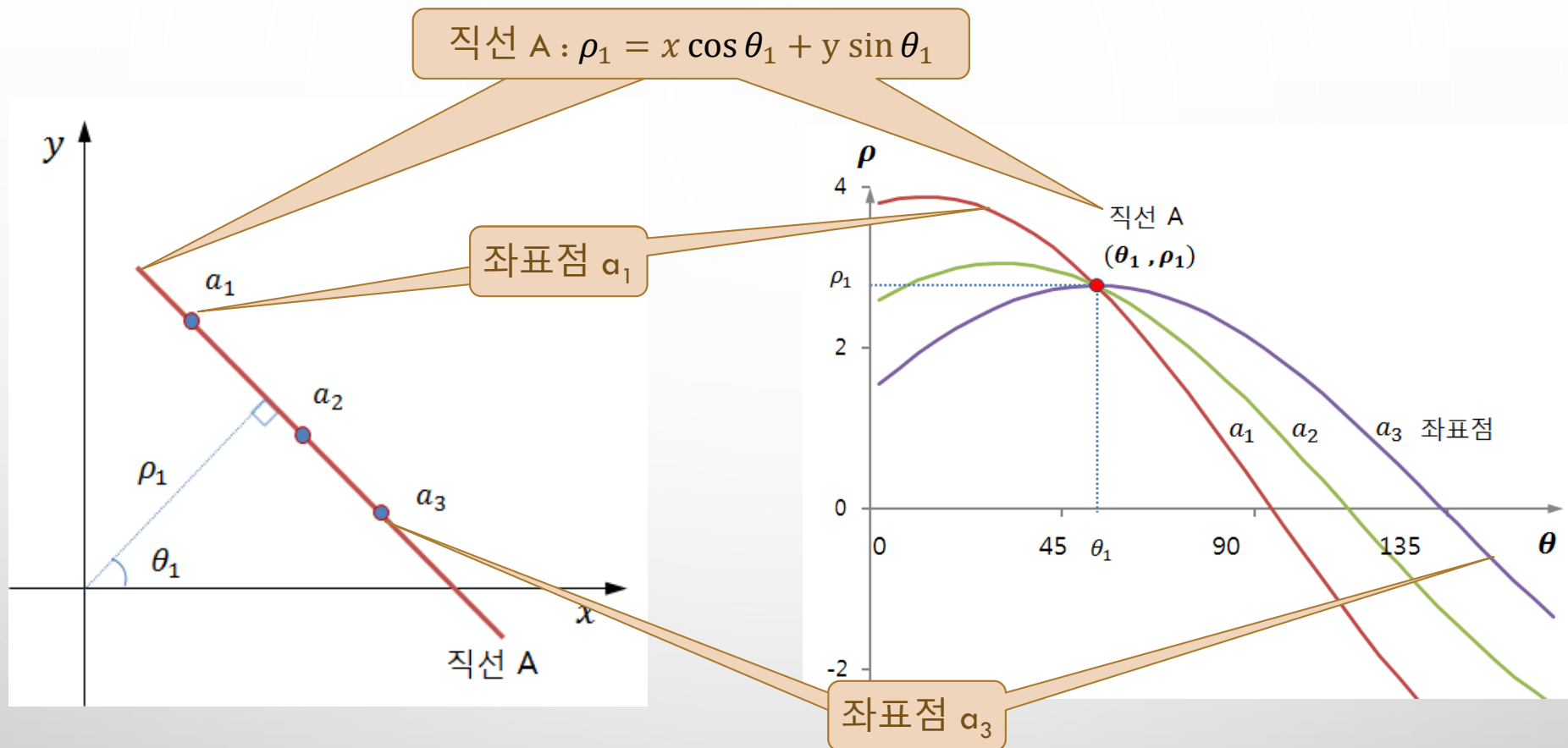
❖ 직교 좌표계로 표현되는 영상의 에지 점들을 극 좌표계로 옮겨, 검출하고자 하는 물체의 파라미터(ρ, θ)를 추출하는 방법

$$y = ax + b \leftrightarrow \rho = x \cdot \cos \theta + y \cdot \sin \theta$$

❖ 직교 좌표계에서 검출의 문제점 (클래식 허프변환)

- 수직선일 경우에 기울기가 무한대
- 검출되는 직선의 간격이 동일하지 않아서 검출 속도와 정밀도에서 문제

10.1.1 허프 변환의 좌표계



- ❖ 직교좌표의 직선은 허프변환 좌표에서 한점 (ρ_1, θ_1) 으로 표현
- ❖ 직교좌표의 한 점은 허프변환 좌표에서 곡선으로 표현

10.1.2 허프 변환의 전체 과정

1. 극 좌표계에서 누적 행렬 구성
2. 영상 화소의 직선 검사
3. 직선 좌표에 대한 극좌표 누적 행렬 구성
4. 누적 행렬의 지역 최댓값 선정
5. 직선 선별 - 임계값 이상인 누적값 선택 및 정렬

◆ 허프 변환 좌표계를 위한 행렬의 계산

$$\begin{aligned} -\rho_{\max} \leq \rho \leq \rho_{\max}, \quad \rho_{\max} = rows + cols, \quad h = \frac{\rho_{\max}}{\Delta \rho} * 2 \\ 0 \leq \theta < \theta_{\max}, \quad \theta_{\max} = \pi, \quad w = \frac{\pi}{\Delta \theta} \end{aligned}$$

교재 오타

각도간격,
거리간격

10.10 허프 누적 행렬 구성

거리간격, 각도간격

누적 행렬

삼각 함수 행렬

직선 극좌표를 누적 행렬의 인덱스로 계산

```
01 def accumulate(image, rho, theta):
02     h, w = image.shape[:2]
03     rows, cols = (h+w) * 2 // rho, int(np.pi / theta)    # 누적행렬 너비, 높이
04     accumulate = np.zeros((rows, cols), np.int32)         # 직선 누적행렬
05
06     sin_cos = [(np.sin(t*theta), np.cos(t*theta)) for t in range(cols)] # 삼각 함수값 저장
07     pts = np.where(image > 0)                                # 넘파이 함수 활용 - 직선좌표 찾기
08
09     polars = np.dot(sin_cos, pts).T                          # 행렬 곱으로 극좌표 계산
10     polars = (polars / rho + rows / 2).astype('int')        # 해상도 변경 및 위치 조정
11
12     for row in polars:
13         for t, r in enumerate(row):                          # 각도, 수직 거리 가져옴
14             accumulate[r, t] += 1                             # 극좌표에 누적
15     return accumulate
```

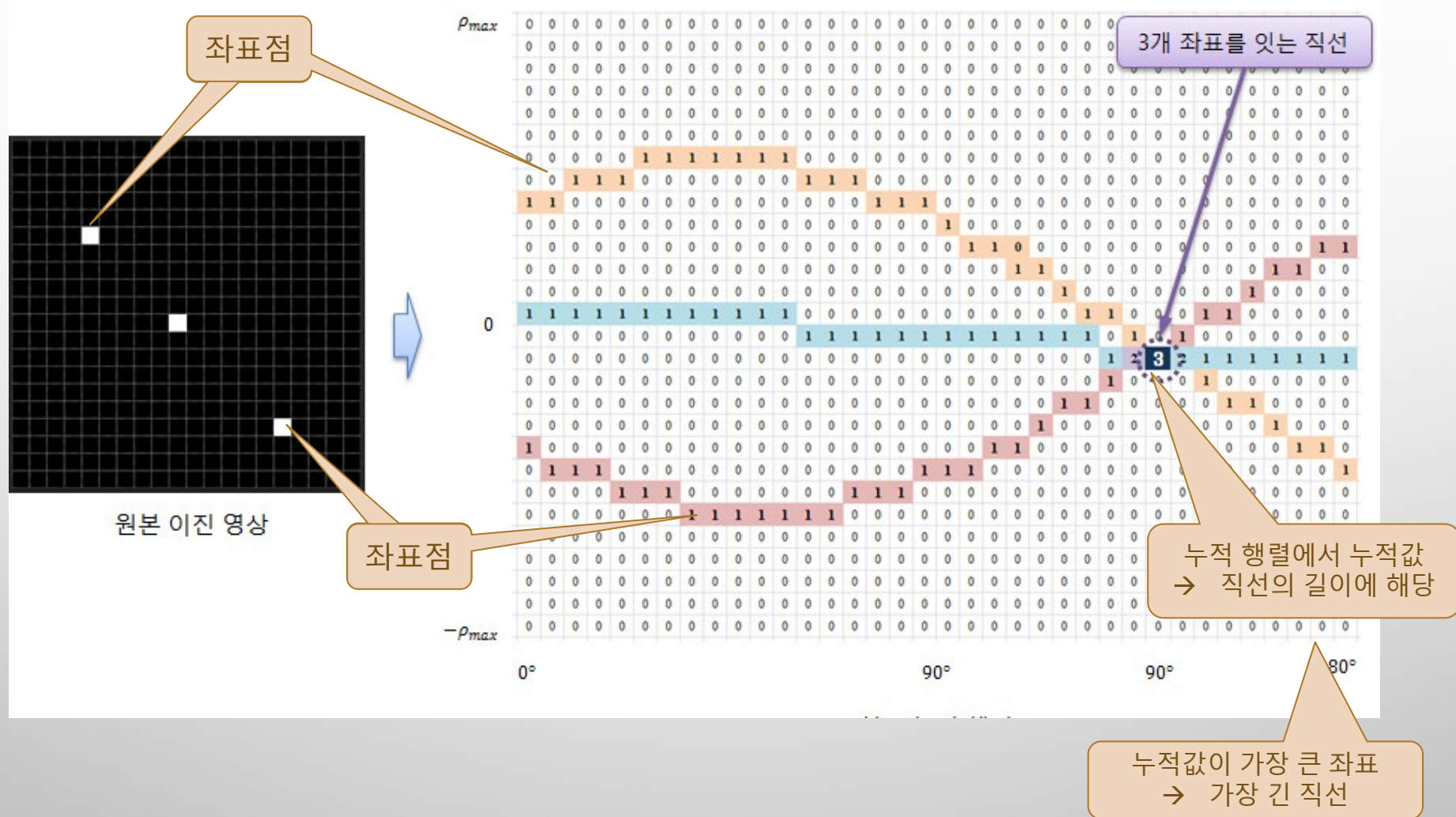
직선 좌표들의 극좌표

0~180도 각도에 대한
삼각함수 값 행렬

$$\begin{bmatrix} \sin \theta_0 & \cos \theta_0 \\ \sin \theta_1 & \cos \theta_1 \\ \sin \theta_2 & \cos \theta_2 \\ \dots & \dots \\ \sin \theta_m & \cos \theta_m \end{bmatrix} \cdot \begin{bmatrix} y_0 & y_1 & y_2 & \dots & y_n \\ x_0 & x_1 & x_2 & \dots & x_n \end{bmatrix} = \begin{bmatrix} \rho_0, \theta_0 & \rho_1, \theta_0 & \rho_2, \theta_0 & \dots & \rho_n, \theta_0 \\ \rho_0, \theta_1 & \rho_1, \theta_0 & \rho_2, \theta_1 & \dots & \rho_n, \theta_1 \\ \rho_0, \theta_2 & \rho_1, \theta_0 & \rho_2, \theta_2 & \dots & \rho_n, \theta_2 \\ \dots & \dots & \dots & \dots & \dots \\ \rho_0, \theta_m & \rho_1, \theta_m & \rho_2, \theta_m & \dots & \rho_n, \theta_m \end{bmatrix}$$

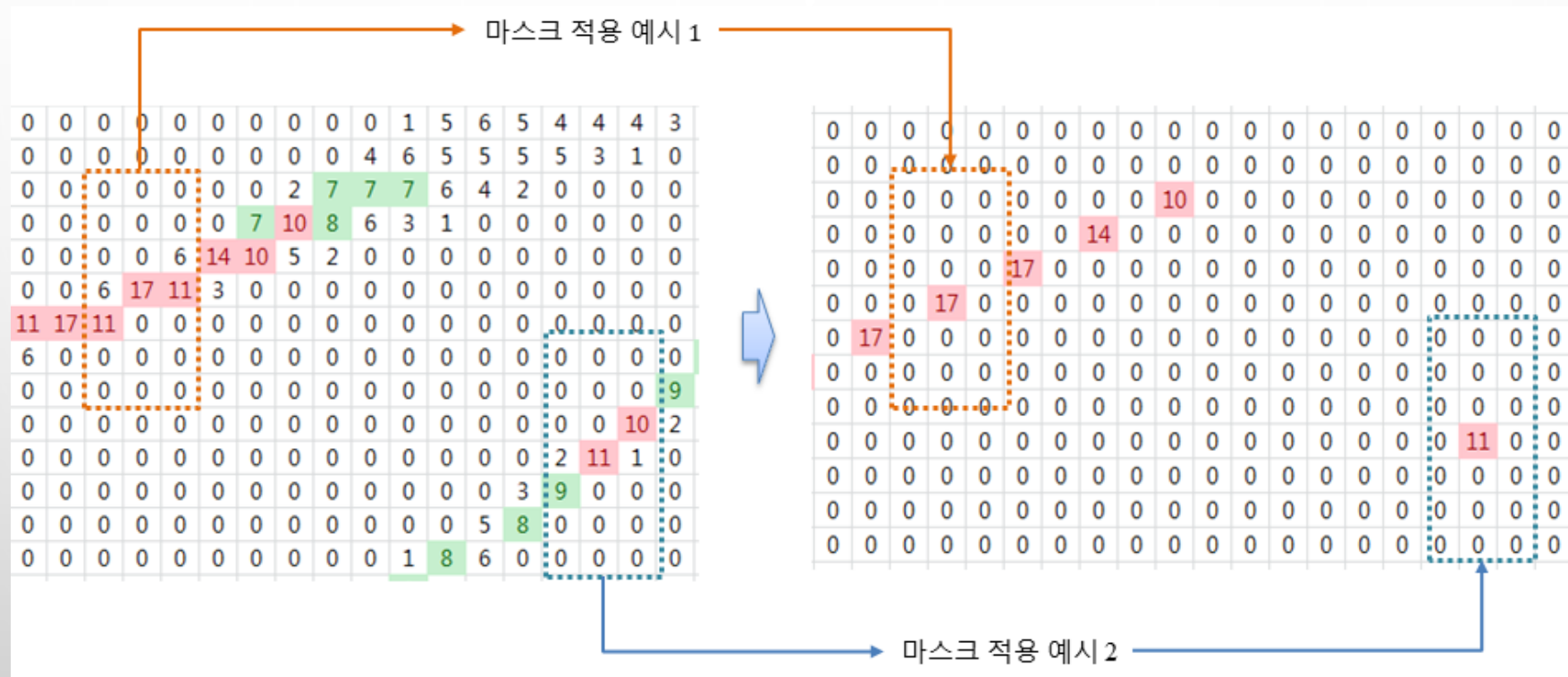
직선 좌표들

10.1.3 허프 누적 행렬 구성



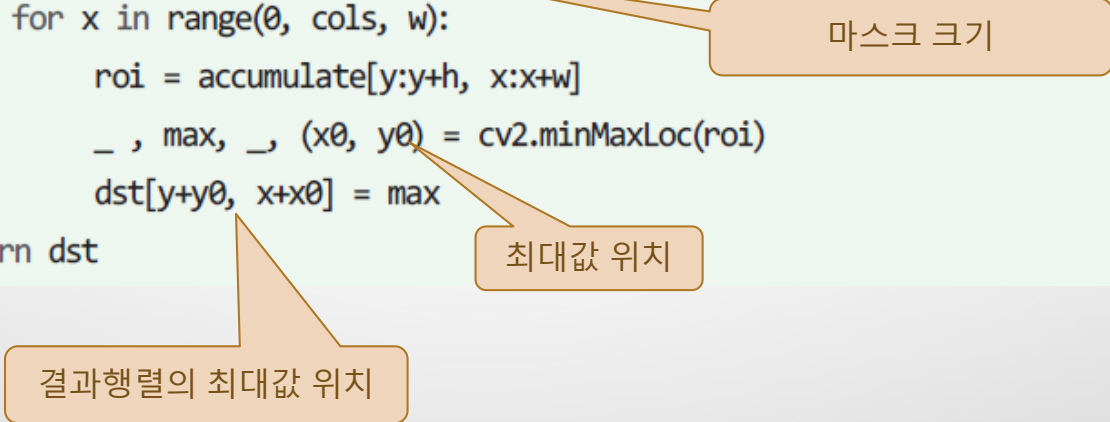
10.1.4 허프 누적 행렬의 지역 최대값 선정

◆ 마스크를 이용해 지역 최대값 선정



10.1.4 허프 누적 행렬의 지역 최대값 선정

```
01 def masking(accumulate, h, w, thresh):
02     rows, cols = accumulate.shape[:2]
03     rcenter, tcenter = h//2, w//2           # 마스크 크기 절반
04     dst = np.zeros(accumulate.shape, np.uint32)
05
06     for y in range(0, rows, h):             # 누적 행렬 조회
07         for x in range(0, cols, w):
08             roi = accumulate[y:y+h, x:x+w]
09             _, max, _, (x0, y0) = cv2.minMaxLoc(roi)
10             dst[y+y0, x+x0] = max
11     return dst
```



마스크 크기

최대값 위치

결과행렬의 최대값 위치

10.1.5 직선(극 좌표) 선택 및 정렬

❖ 직선 선별 함수

극 좌표 행렬의 인덱스

```
01 def select_lines(acc_dst, rho, theta, thresh):
    rows = acc_dst.shape[0]
    03 r, t = np.where(acc_dst>thresh)
    04
    05 rhos = ((r - (rows / 2)) * rho)
    06 radians = t * theta
    07 values = acc_dst[r, t]
    09
    10 idx = np.argsort(values)[::-1]
    11 lines = np.transpose([rhos, radians])
    12 lines = lines[idx, :]
    13 return np.expand_dims(lines, axis=1)
```

누적값 기준 정렬

직선 벡터(lines)에 단일 직선 저장

임계값 이상 인덱스 가져옴

인덱스로 수직 거리 계산

인덱스로 각도 계산

인덱스로 누적값 가져옴

내림차순 정렬 인덱스

리스트 전치하여 행렬 생성

누적값 기준으로 극좌표 정렬

1번(열) 차원 증가

OpenCV 함수와 동일하게 반환 자료형 맞춤

수직거리 행렬

$\begin{bmatrix} \rho_0 & \rho_1 & \rho_2 & \cdots & \rho_n \\ \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_n \end{bmatrix}$

각도 행렬

직선 극 좌표

전치

$\begin{bmatrix} \rho_0 & \theta_0 \\ \rho_1 & \theta_1 \\ \rho_2 & \theta_2 \\ \vdots & \vdots \\ \rho_n & \theta_n \end{bmatrix}$

직선 극 좌표

10.1.7 최종 완성 프로그램

예제 10.1.1

허프 변환을 이용한 직선 검출 - 01.hough_lines.py

```
01 import numpy as np, cv2, math
02 from Common.hough import accumulate, masking, select_lines    # 허프 변환 함수 импорт
03
04 def houghLines(src, rho, theta, thresh):                      # 허프 변환 함수
05     acc_mat = accumulate(src, rho, theta)                    # 직선 누적 행렬 계산
06     acc_dst = masking(acc_mat, 7, 3, thresh)                 # 마스킹 처리 - 7행, 3열
07     lines = select_lines(acc_dst, rho, theta, thresh)         # 임계 직선 선택
08     return lines
09
10 def draw_houghLines(src, lines, nline):                      # 검출 직선 그리기 함수
11     dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)             # 컬러 영상 변환
12     min_length = min(len(lines), nline)
13
14     for i in range(min_length):
15         rho, radian = lines[i, 0, 0:2]                       # 수직 거리, 각도 - 3차원 행렬임
16         a, b = math.cos(radian), math.sin(radian)
17         pt= (a * rho, b * rho)                                # 검출 직선상의 한 좌표
18         delta= (-1000 * b, 1000 * a)                          # 직선상의 이동 위치
19         pt1 = np.add(pt, delta).astype('int')
20         pt2 = np.subtract(pt, delta).astype('int')
21         cv2.line(dst, tuple(pt1), tuple(pt2), (0, 255, 0), 2, cv2.LINE_AA)
22
23     return dst
24
```

허프변환
전체 과정 수행 함수

직선위 2개 좌표 계산

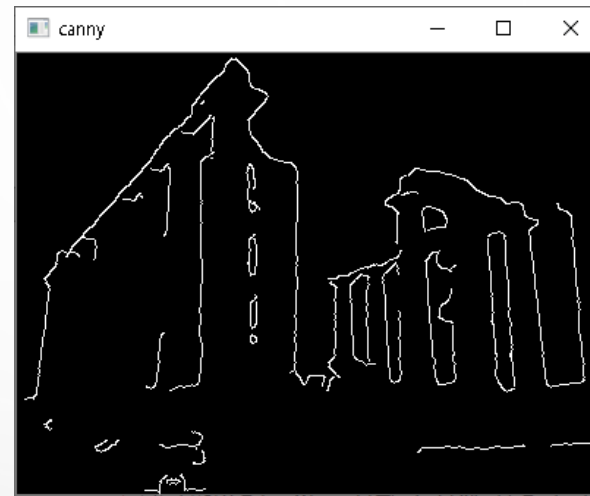
10.1.7 최종 완성 프로그램

❖ 영상에 검출 직선 그리기 함수

```
25 image = cv2.imread("images/hough.jpg", cv2.IMREAD_GRAYSCALE)
26 if image is None: raise Exception("영상파일 읽기 에러")
27 blur = cv2.GaussianBlur(image, (5, 5), 2, 2)           # 가우시안 블러링
28 canny = cv2.Canny(blur, 100, 200, 5)                  # 캐니 에지 추출
29
30 rho, theta = 1, np.pi / 180                          # 수직거리 간격, 각도 간격
31 lines1 = houghLines(canny, rho, theta, 80)             # 저자 구현 함수
32 lines2 = cv2.HoughLines(canny, rho, theta, 80)         # OpenCV 함수
33 dst1 = draw_houghLines(canny, lines1, 7)               # 직선 그리기
34 dst2 = draw_houghLines(canny, lines2, 7)
35
36 cv2.imshow("image", image)
37 cv2.imshow("canny", canny);
38 cv2.imshow("detected lines", dst1)
39 cv2.imshow("detected lines_OpenCV", dst2)
40 cv2.waitKey(0)
```

10.1.7 최종 완성 프로그램

◆ 실행결과



■ 허프 변환에 의한 선분 검출

```
cv2.HoughLines(image, rho, theta, threshold, lines=None, srn=None, stn=None,  
               min_theta=None, max_theta=None) -> lines
```

- image: 입력 에지 영상
- rho: 축적 배열에서 rho 값의 간격. (e.g.) 1.0 → 1픽셀 간격.
- theta: 축적 배열에서 theta 값의 간격. (e.g.) $\text{np.pi} / 180 \rightarrow 1^\circ$ 간격.
- threshold: 축적 배열에서 직선으로 판단할 임계값
- lines: 직선 파라미터(rho, theta) 정보를 담고 있는 `numpy.ndarray`.
`shape=(N, 1, 2)`. `dtype=numpy.float32`.
- srn, stn: 멀티 스케일 허프 변환에서 rho 해상도, theta 해상도를 나누는 값.
기본값은 0이고, 이 경우 일반 허프 변환 수행.
- min_theta, max_theta: 검출할 선분의 최소, 최대 theta 값

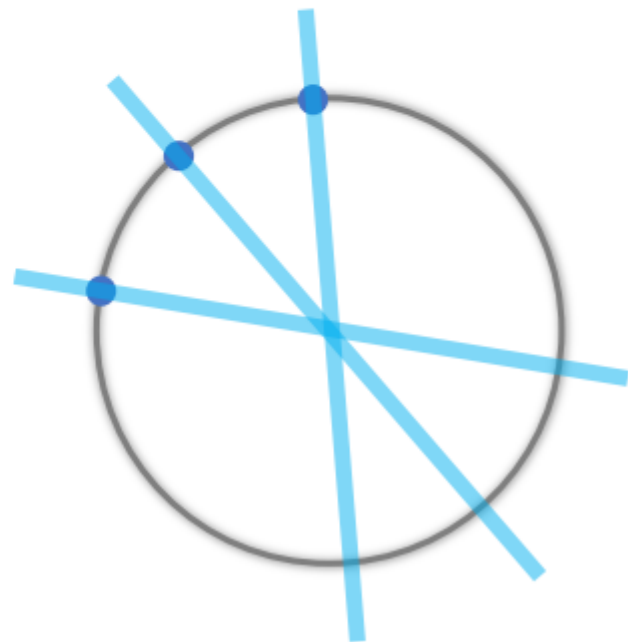
■ 확률적 허프 변환에 의한 선분 검출

```
cv2.HoughLinesP(image, rho, theta, threshold, lines=None,  
                 minLineLength=None, maxLineGap=None) -> lines
```

- image: 입력 에지 영상
- rho: 축적 배열에서 rho 값의 간격. (e.g.) 1.0 → 1픽셀 간격.
- theta: 축적 배열에서 theta 값의 간격. (e.g.) $\text{np.pi} / 180 \rightarrow 1^\circ$ 간격.
- threshold: 축적 배열에서 직선으로 판단할 임계값
- lines: 선분의 시작과 끝 좌표(x1, y1, x2, y2) 정보를 담고 있는 `numpy.ndarray`.
shape=(N, 1, 4). dtype=`numpy.int32`.
- minLineLength: 검출할 선분의 최소 길이
- maxLineGap: 직선으로 간주할 최대 에지 점 간격

원 검출 가능?

- 허프 변환을 응용하여 원을 검출할 수 있음
 - 원의 방정식: $(x - a)^2 + (y - b)^2 = c^2 \rightarrow 3$ 차원 축적 평면?
- 속도 향상을 위해 Hough gradient method 사용
 - 입력 영상과 동일한 2차원 평면 공간에서 축적 영상을 생성
 - 에지 픽셀에서 그래디언트 계산
 - 에지 방향에 따라 직선을 그리면서 값을 누적
 - 원의 중심을 먼저 찾고, 적절한 반지름을 검출
 - 단점
 - 여러 개의 동심원을 검출 못함
 \rightarrow 가장 작은 원 하나만 검출됨



■ 허프 변환 원 검출 함수

```
cv2.HoughCircles(image, method, dp, minDist, circles=None, param1=None,  
                 param2=None, minRadius=None, maxRadius=None) -> circles
```

- image: 입력 영상. (에지 영상이 아닌 일반 영상)
- method: OpenCV 4.2 이하에서는 `cv2.HOUGH_GRADIENT`만 지정 가능
- dp: 입력 영상과 축적 배열의 크기 비율. 1이면 동일 크기.
2이면 축적 배열의 가로, 세로 크기가 입력 영상의 반.
- minDist: 검출된 원 중심점들의 최소 거리
- circles: (cx, cy, r) 정보를 담은 `numpy.ndarray`. shape=(1, N, 3), dtype=np.float32.
- param1: Canny 에지 검출기의 높은 임계값
- param2: 축적 배열에서 원 검출을 위한 임계값
- minRadius, maxRadius: 검출할 원의 최소, 최대 반지름

10.3 k-최근접 이웃 분류기

- ◆ 10.3.1 k-최근접 이웃 분류기의 이해
- ◆ 10.3.2 k-NN을 위한 KNearest 클래스의 이해
- ◆ 10.3.3 k-NN 응용

10.3.1 k-최근접 이웃 분류기의 이해

◆최근접 이웃 알고리즘

- ❖ 기존에 가지고 있는 데이터들을 일정한 규칙에 의해 분류된 상태에서 새로운 입력 데이터의 종류를 예측하는 분류 알고리즘
- ❖ 학습 클래스의 샘플들과 새 샘플의 거리가 가장 가까운(nearest)클래스로 분류
 - ‘가장 가까운 거리’
 - ✓ 미지의 샘플과 학습 클래스 샘플간의 유사도가 가장 높은 것을 의미
 - ✓ 유클리드 거리(euclidean distance), 해밍 거리(hamming distance), 차분 절대값

10.3.1 k-최근접 이웃 분류기의 이해

◆k-최근접 이웃 분류(k-Nearest Neighbors: k-NN)

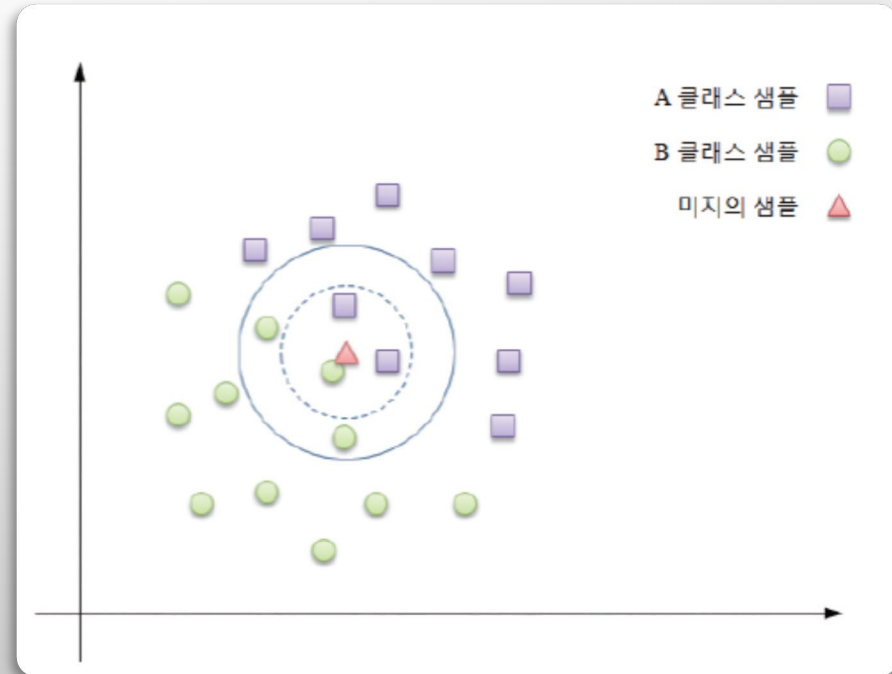
❖ 학습된 클래스들에서 여러 개(k)의 가까운 이웃을 선출하고 이를 이용하여 미지의 샘플들을 분류하는 방법

❖ k가 3일 경우

- 미지 샘플 주변 가장 가까운 이웃 3개 선출
- 이 중 많은 수의 샘플을 가진 클래스로 미지의 샘플 분류
- A클래스 샘플 2개, B클래스 샘플 1개 → A클래스 분류

❖ k가 5일 경우

- 실선 큰 원내에 있는 가장 가까운 이웃 5개 선출
- 2개 A 클래스, 3개 B 클래스 → B클래스로 분류



10.3.2 k-NN을 위한 KNearest 클래스의 이해

예제 10.3.1

임의 좌표 생성 - 04.kNN_exam.py

```
01 import numpy as np, cv2
02
03 def draw_points(image, group, color):
04     for p in group:
05         pt = tuple(p.astype(int))
06         cv2.circle(image, pt, 3, color, cv2.FILLED)
07
08 nsample = 50
09 traindata = np.zeros((nsample*2, 2), np.float32)
10 label = np.zeros((nsample*2, 1), np.float32)
11
12 cv2.randn(traindata[:nsample], 150, 30)
13 cv2.randn(traindata[nsample:], 250, 60)
14 label[:nsample], label[nsample:] = 0, 1
15
16 K = 7
17 knn = cv2.ml.KNearest_create()
18 knn.train(traindata, cv2.ml.ROW_SAMPLE, classlable)
19
```

반지름 3의 원 표시 - 점으로 표시

정수 원소 튜플

반환 행렬

그룹당 학습 데이터 수

전체 학습 데이터 행렬

레이블 행렬 생성

행렬 원소에 정규분포를
따르는 임의값 지정

정규분포 랜덤 값 생성

레이블 기준값 지정

두 그룹에 레이블 값 지정

kNN 클래스로 객체 생성

학습 수행

10.3.2 k-NN을 위한 KNearest 클래스의 이해

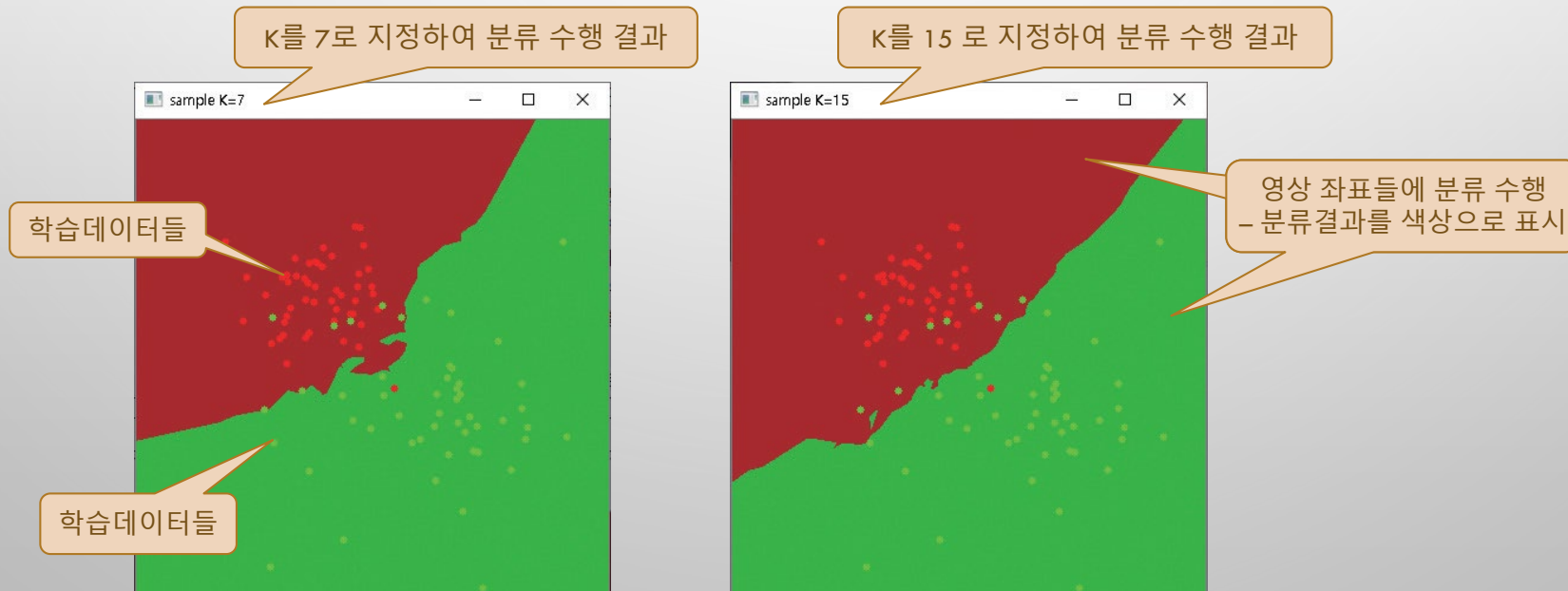
```
20 points = [(x, y) for y in range(400) for x in range(400) ]      # 검사 좌표 리스트 생성
21 ret, resp, neig, dist = knn.findNearest(np.array(points, np.float32), K)  # 분류 수행
22
23 colors = [(0, 180, 0) if p else (0, 0, 180) for p in resp]
24 image = np.reshape(colors, (400, 400, 3)).astype('uint8')      # 3채널 컬러
25
26 draw_points(image, traindata[:nsample], color=(0, 0, 255))
27 draw_points(image, traindata[nsample:], color=(0, 255, 0))
28 cv2.imshow("sample K="+ str(K), image)
29 cv2.waitKey(0)
```

분류된 좌표들에 색지정

3채널 컬러

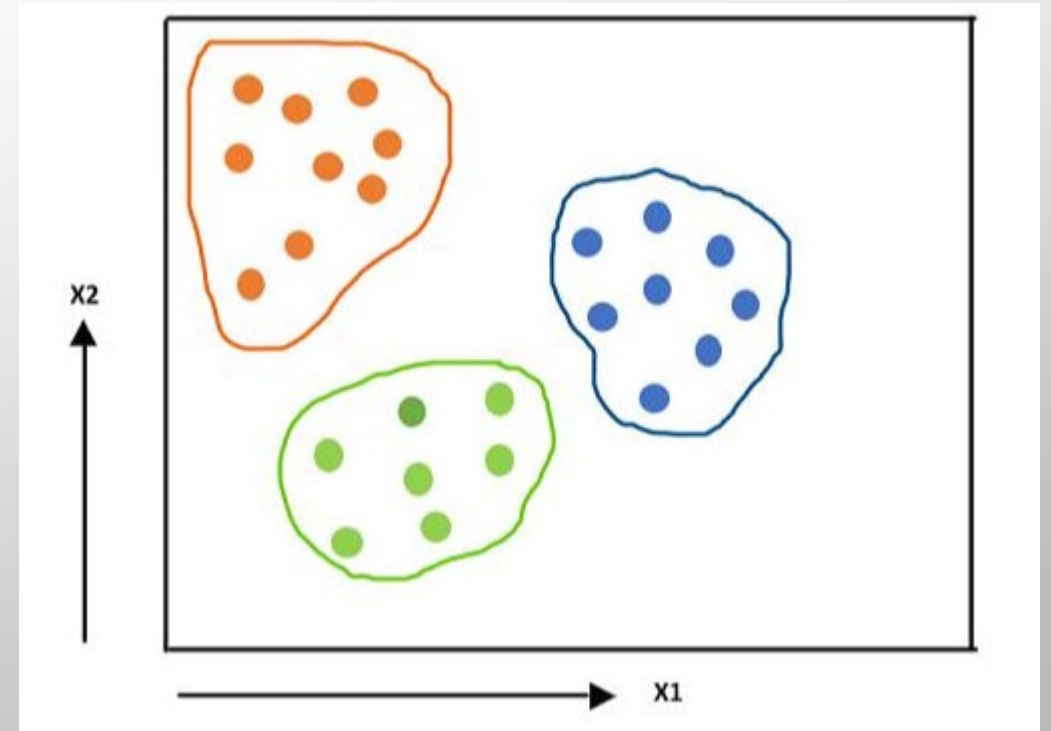
좌표들을 400x400 크기
영상으로 형태 변경

◆ 실행결과



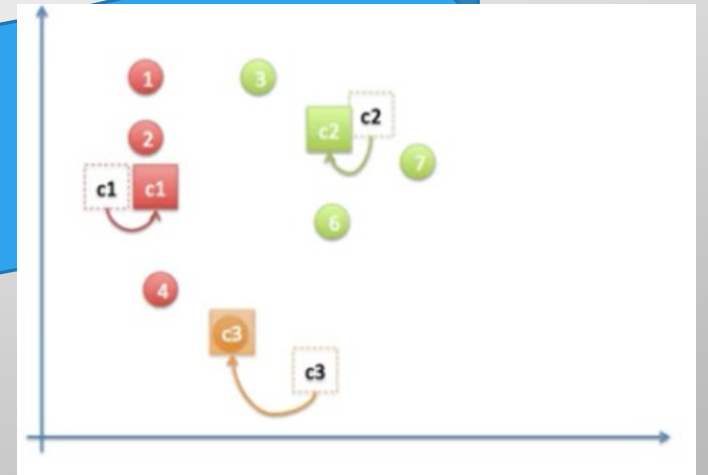
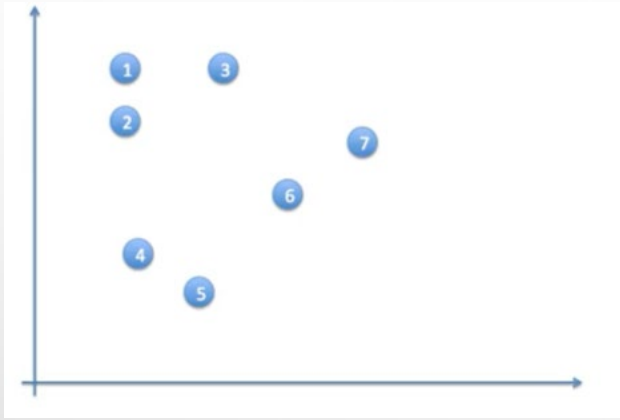
10.3 K-means clustering(추가)

- ◆ 비지도 학습(Unsupervised Learning) 알고리즘 중 하나.
 - ❖ 데이터 셋에서 서로 유사한 그룹으로 묶어 분류하는 것
 - ❖ 어떤 데이터 셋이 있고 k 개의 클러스터로 분류하겠다면,
 - 데이터 셋에는 k 개의 중심이 존재.
 - 각 데이터들은 유클리디안 거리를 기반으로 가까운 중심에 할당



10.3 K-means clustering(추가)

◆ k-means clustering step



외곽선 검출

◆ 외곽선

- ❖ 객체를 둘러싸고 있는 점들의 조합

◆ 외곽선 검출이란

- ❖ 객체의 외곽선 좌표를 모두 추출하는 작업. *Contour tracing*
- ❖ 바깥쪽 & 안쪽(홀) 외곽선 → 외곽선의 계층 구조도 표현 가능

◆ 외곽선 객체 하나의 표현 방법

- ❖ $\text{shape} = (K, 1, 2)$ (K는 외곽선 좌표 개수)
- ❖ $\text{dtype} = \text{numpy.int32}$



외곽선 검출

◆ 외곽선 검출 함수

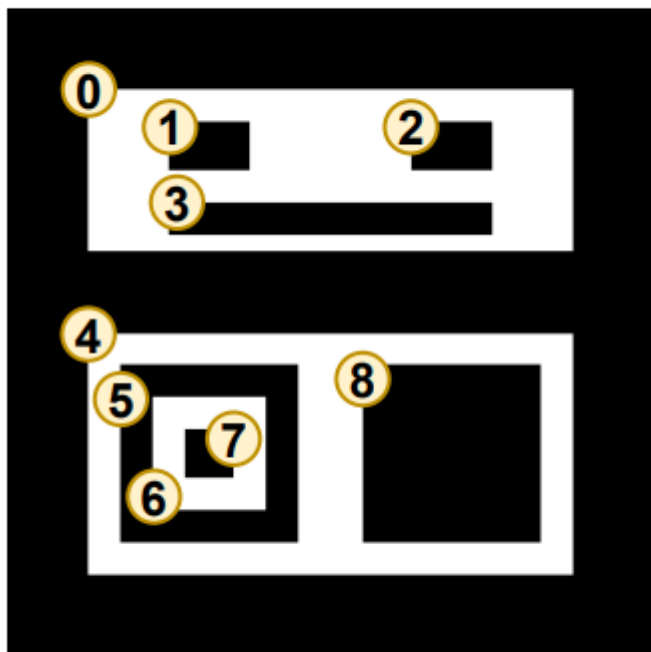
```
cv2.findContours(image, mode, method, contours=None, hierarchy=None,  
                offset=None) -> contours, hierarchy
```

- image: 입력 영상. non-zero 픽셀을 객체로 간주함.
- mode: 외곽선 검출 모드. cv2.RETR_로 시작하는 상수.
- method: 외곽선 근사화 방법. cv2.CHAIN_APPROX_로 시작하는 상수.
- contours: 검출된 외곽선 좌표. `numpy.ndarray`로 구성된 리스트.
`len(contours)=전체 외곽선 개수(N)`.
`contours[i].shape=(K, 1, 2)`. `contours[i].dtype=numpy.int32`.
- hierarchy: 외곽선 계층 정보. `numpy.ndarray`. `shape=(1, N, 4)`. `dtype=numpy.int32`.
`hierarchy[0, i, 0] ~ hierarchy[0, i, 3]`이 순서대로 next, prev, child, parent
외곽선 인덱스를 가리킴. 해당 외곽선이 없으면 -1.
- offset: 좌표 값 이동 오프셋. 기본값은 (0, 0).

외곽선 검출

◆ mode

• mode:



▪ cv2.RETR_EXTERNAL 0 ↔ 4

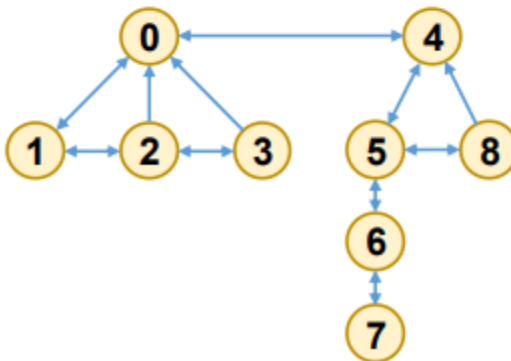
▪ cv2.RETR_LIST



▪ cv2.RETR_CCOMP



▪ cv2.RETR_TREE



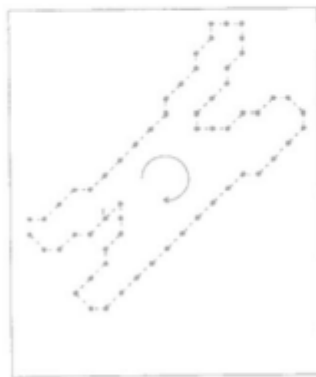
계층 정보 X

계층 정보 O

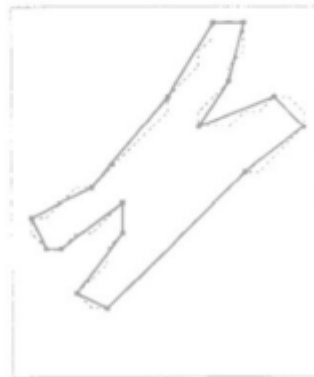
외곽선 검출

◆ method

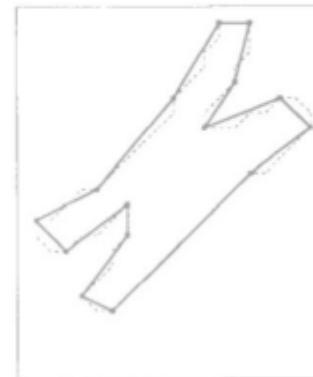
- **cv2.CHAIN_APPROX_NONE** : 근사화 없음
- **cv2.CHAIN_APPROX_SIMPLE** : 수직선, 수평선, 대각선에 대해 끝점만 저장
- **cv2.CHAIN_APPROX_TC89_L1** : Teh & Chin L1 근사화
- **cv2.CHAIN_APPROX_TC89_KCOS** : Teh & Chin k cos 근사화



contours



L1



k cos

외곽선 검출

◆ 외곽선 그리기 함수

```
cv2.drawContours(image, contours, contourIdx, color, thickness=None,  
                 lineType=None, hierarchy=None, maxLevel=None, offset=None)  
-> image
```

- image: 입출력 영상
- contours: (cv2.findContours() 함수로 구한) 외곽선 좌표 정보
- contourIdx: 외곽선 인덱스. 음수(-1)를 지정하면 모든 외곽선을 그린다.
- color: 외곽선 색상
- thickness: 외곽선 두께. thickness < 0이면 내부를 채운다.
- lineType: LINE_4, LINE_8, LINE_AA 중 하나 지정
- hierarchy: 외곽선 계층 정보.
- maxLevel: 그리기를 수행할 최대 외곽선 레벨. maxLevel = 0 이면 contourIdx로 지정된 외곽선만 그린다.

계층 정보를 사용한 외곽선 검출

```
import random
import numpy as np
import cv2

src = cv2.imread("images/contours.bmp", cv2.IMREAD_GRAYSCALE)

if src is None: raise Exception("파일 에러")

contours, hier = cv2.findContours(src, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_NONE)

dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)

idx = 0
while idx >= 0:
    c = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    cv2.drawContours(dst, contours, idx, c, 2, cv2.LINE_8, hier)
    idx = hier[0, idx, 0]

cv2.imshow('src', src)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

계측 정보를 사용하지 않은 외곽선 검출

```
import random
import numpy as np
import cv2

src = cv2.imread("images\milkdrop.bmp", cv2.IMREAD_GRAYSCALE)

if src is None: raise Exception("파일 에러")

_, src_bin = cv2.threshold(src, 0, 255, cv2.THRESH_OTSU)

contours, _ = cv2.findContours(src_bin, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)

h, w = src.shape[:2]
dst = np.zeros((h, w, 3), np.uint8)

for i in range(len(contours)):
    c = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    cv2.drawContours(dst, contours, i, c, 1, cv2.LINE_AA)

cv2.imshow('src', src)
cv2.imshow('src_bin', src_bin)
cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```


다양한 외곽선 처리 함수

◆ 외곽선 처리 함수

함수 이름	설명
<code>cv2.arcLength()</code>	외곽선 길이를 반환
<code>cv2.contourArea()</code>	외곽선이 감싸는 영역의 면적을 반환
<code>cv2.boundingRect()</code>	주어진 점을 감싸는 최소 크기 사각형(바운딩 박스) 반환
<code>cv2.minEnclosingCircle()</code>	주어진 점을 감싸는 최소 크기 원을 반환
<code>cv2.minAreaRect()</code>	주어진 점을 감싸는 최소 크기 회전된 사각형을 반환
<code>cv2.minEnclosingTriangle()</code>	주어진 점을 감싸는 최소 크기 삼각형을 반환
<code>cv2.approxPolyDP()</code>	외곽선을 근사화(단순화)
<code>cv2.fitEllipse()</code>	주어진 점에 적합한 타원을 반환
<code>cv2.fitLine()</code>	주어진 점에 적합한 직선을 반환
<code>cv2.isContourConvex()</code>	컨벡스인지를 검사
<code>cv2.convexHull()</code>	주어진 점으로부터 컨벡스 헐을 반환
<code>cv2.convexityDefects()</code>	주어진 점과 컨벡스 헐로부터 컨벡스 디펙트를 반환

다양한 외곽선 처리 함수

◆ 외곽선 길이 구하기

```
cv2.arcLength(curve, closed) -> retval
```

- curve: 외곽선 좌표. `numpy.ndarray`. `shape=(K, 1, 2)`
- closed: True이면 폐곡선으로 간주
- retval: 외곽선 길이

◆ 면적 구하기

```
cv2.contourArea(contour, oriented=None) -> retval
```

- contour: 외곽선 좌표. `numpy.ndarray`. `shape=(K, 1, 2)`
- oriented: True이면 외곽선 진행 방향에 따라 부호 있는 면적을 반환. 기본값은 False.
- retval: 외곽선으로 구성된 영역의 면적

다양한 외곽선 처리 함수

◆ 바운딩 박스

```
cv2.boundingRect(array) -> retval
```

- array: 외곽선 좌표. `numpy.ndarray`. `shape=(K, 1, 2)`
- retval: 사각형 정보. (x, y, w, h) 튜플.

◆ 바운딩 서클

```
cv2.minEnclosingCircle(points) -> center, radius
```

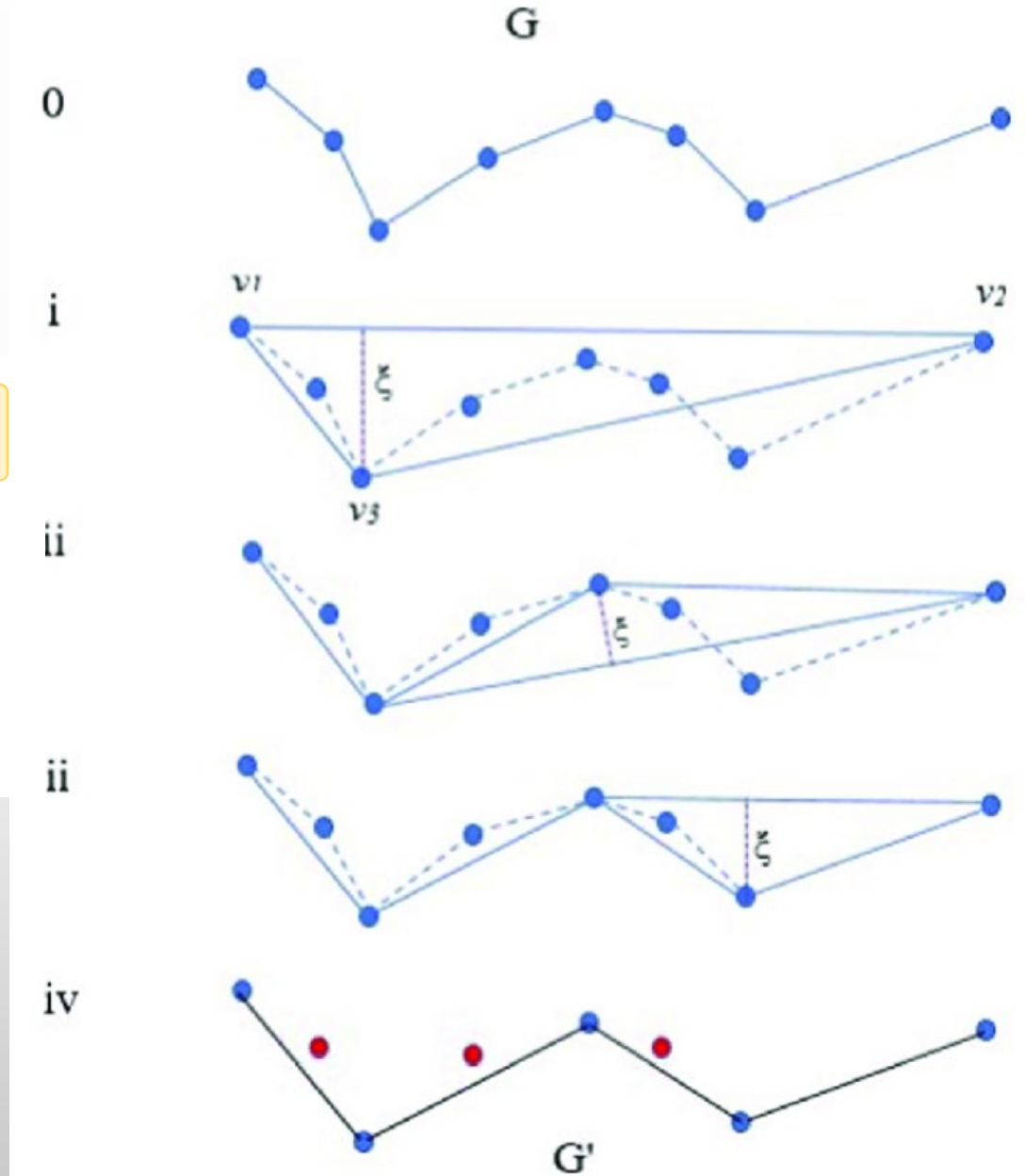
- points: 외곽선 좌표. `numpy.ndarray`. `shape=(K, 1, 2)`
- center: 바운딩 서클 중심 좌표. (x, y) 튜플.
- radius: 바운딩 서클 반지름. 실수.

다양한 외곽선 처리 함수

◆ 외곽선 근사화

```
cv2.approxPolyDP(curve, epsilon, closed, approxCurve=None) -> approxCurve
```

- curve: 입력 곡선 좌표. `numpy.ndarray.shape=(K, 1, 2)`
- epsilon: 근사화 정밀도 조절. 입력 곡선과 근사화 곡선 간의 최대 거리. e.g) `cv2.arcLength(curve) * 0.02`
- closed: True를 전달하면 폐곡선으로 인식
- approxCurve: 근사화된 곡선 좌표. `numpy.ndarray.shape=(K', 1, 2)`

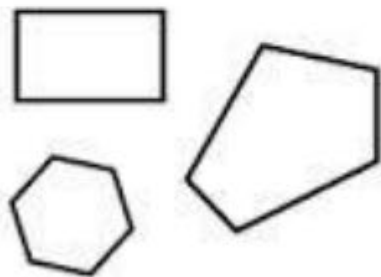


다양한 외곽선 처리 함수

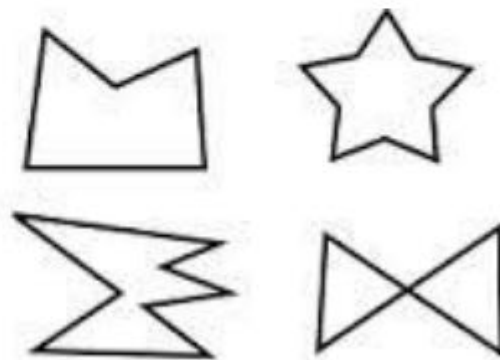
◆ Convex 검사

```
cv2.isContourConvex(contour) -> retval
```

- contour: 입력 곡선 좌표. `numpy.ndarray`. `shape=(K, 1, 2)`
- retval: 컨벡스이면 True, 아니면 False.



Convex Polygons



Non-convex Polygons