Chapter

10

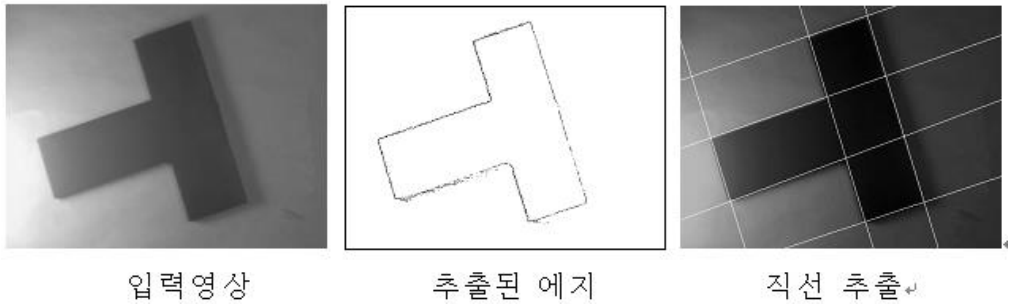
영상분할 및 특징처리 (2)

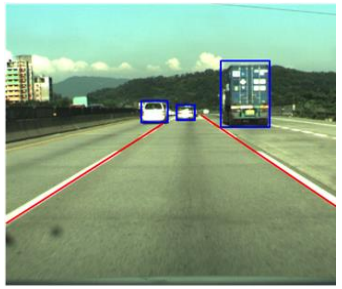
1

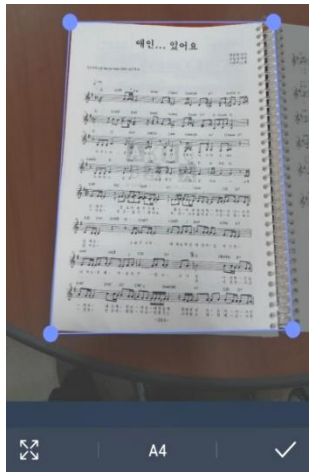
허프변환

허프변환(Hough Transform: HT)

- **영상분석, 컴퓨터비전, 패턴인식 등의 분야에서 널리 사용되는 특징 추출(feature extraction) 방법 중의 하나**
  - 영상 내 직선(lines), 원(circles), 타원(ellipses) 등의 추출에 사용





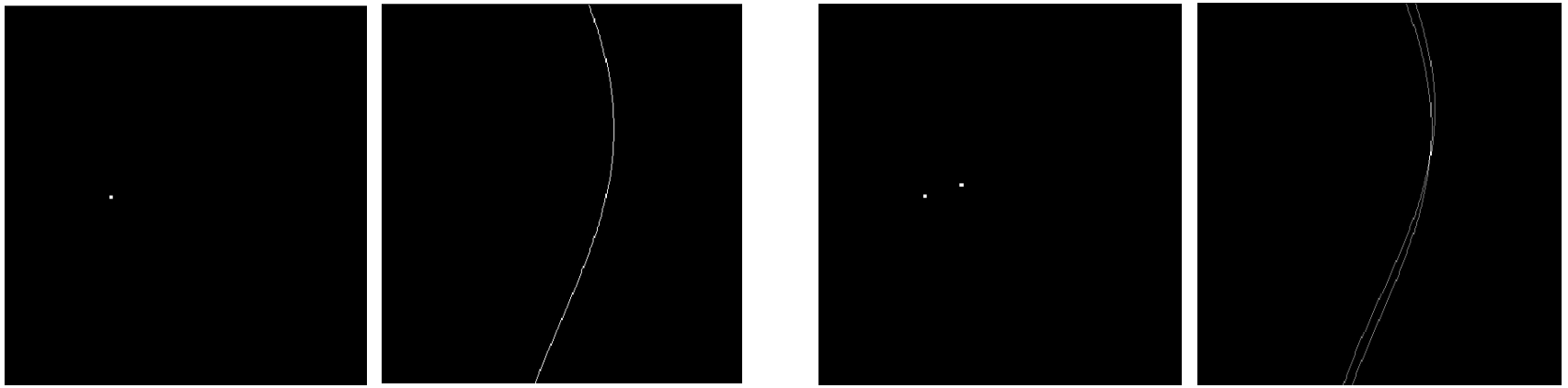


- **기본원리는 영상 내 어떤 기하학적 형태의 불완전한 부분들이 존재할 때 보팅(voting) 기법에 의해 관심 형상을 찾아 내는 것**
  - 보팅과정은 추출하려는 기하형상의 파라메터 공간(parameter space)에서 수행되고,
  - 보팅이 끝난 후 파라메터 공간 내에서 부분 최대치(local maxima)를 탐색함에 의해 형상을 찾아 낼 수 있음

기본원리

❖ 이미지 공간을 파라메터 공간으로 변환

- 이미지 공간상의 한 점은 파라메터 공간에서 직선
- 이미지 공간상에서 직선을 구성하고 있는 모든 점들을 파라메터 공간으로 변환하면 파라메터 공간상의 선들은 서로 교차하게 됨
- 직선을 구성하는 점에 대응하는 파라메터 공간상 직선의 교차점을 voting



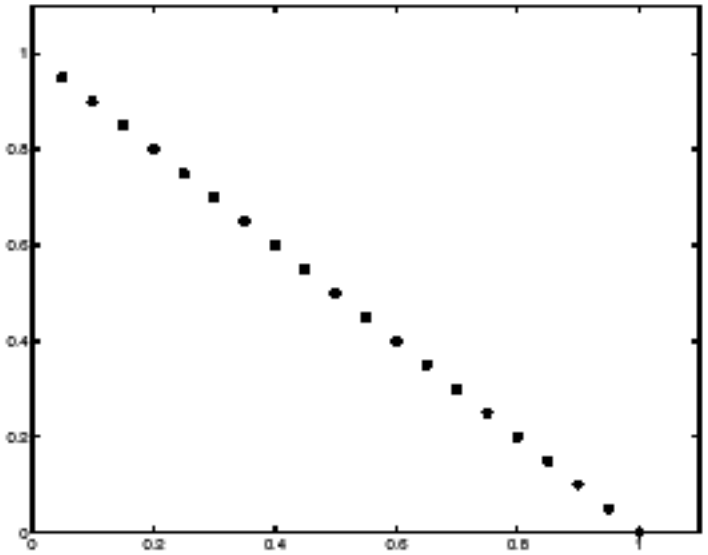
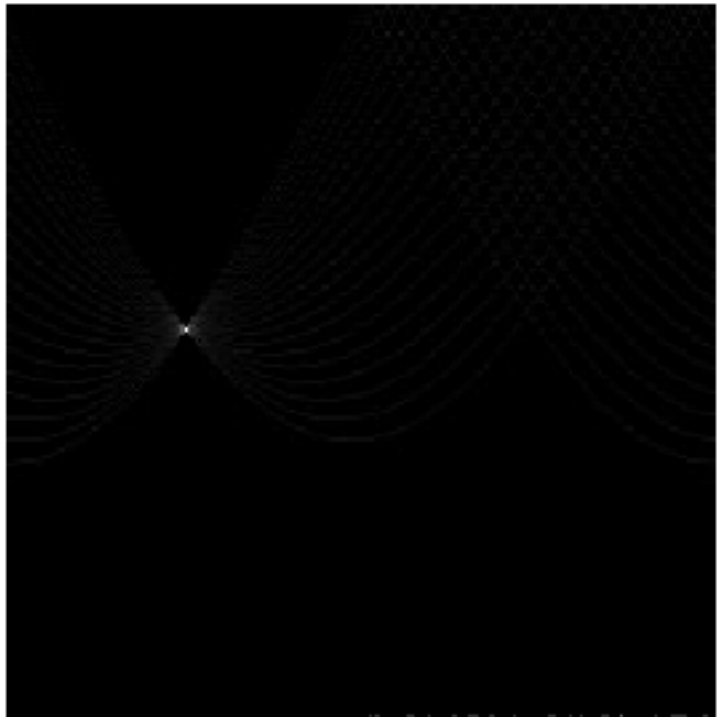
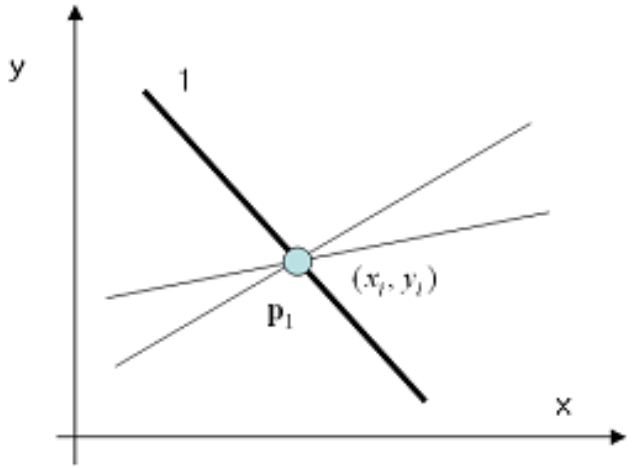


Image space

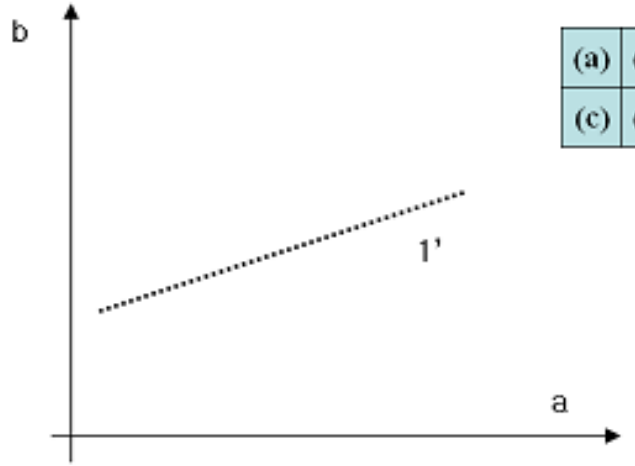


Votes

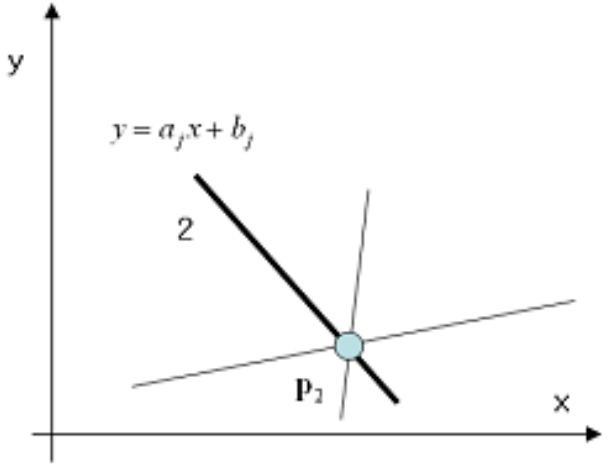
Horizontal axis is θ,
vertical is ρ.



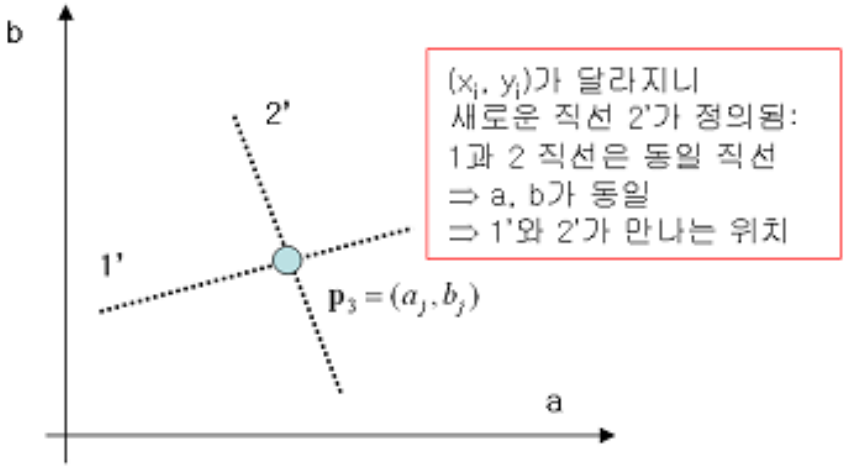
① p_1점을 지나는 모든 직선은
$y_i = ax_i + b$에 의해 표현
(기울기 a와 절편 b가 다름)



② $a-b$면에 표현시 (x_i, y_i)가 같고
a(가로축)를 바꿀수 있으므로
p_1점은 하나의 직선이 된다.



③ 그런데, 또 다른 위치의 점 p_2에 대해

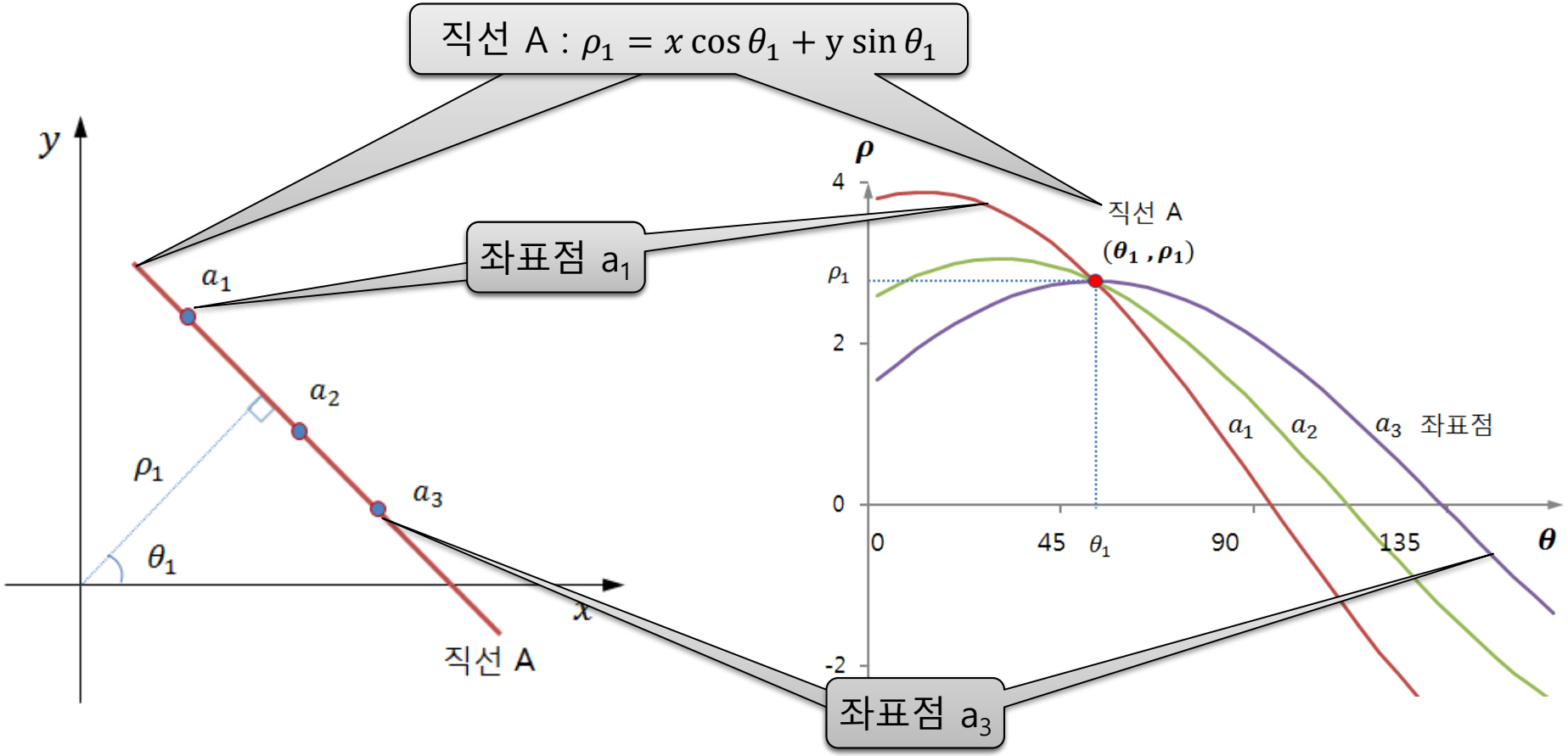


(x_i, y_i)가 달라지니
새로운 직선 2'가 정의됨:
1과 2 직선은 동일 직선
⇒ a, b가 동일
⇒ 1'와 2'가 만나는 위치

④ 2'의 직선이 정의되는데 x-y면에서 p_1p_2선은
a-b 평면에서 동일점을 나타내므로
p_3의 위치에서 두 번 보팅

HT 구현의 이슈

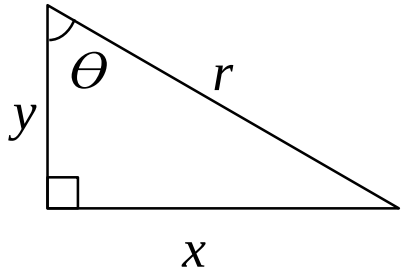
- 직선의 기울기는 $-\infty < a < \infty$ 이고 파라메터 공간에서 의 범위는 무한대.
- 또한 $y = ax + b$와 같은 표현은 $x = c$ 와 같은 직선을 표현할 수 없음.
- 따라서 직선의 표현을 삼각함수를 이용하여 바꾸어 이러한 문제를 극복.



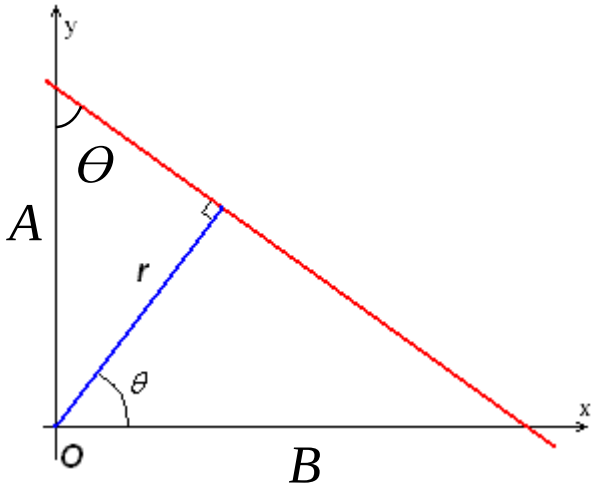
$\rho < d$

$\theta < 180^\circ$

직선방정식($r = cos\theta\ x + sin\theta\ y$)



$$sin\theta = \frac{x}{r} \qquad cos\theta = \frac{y}{r} \qquad tan\theta = \frac{x}{y} = \frac{sin\theta}{cos\theta}$$



$$y = ax + b$$

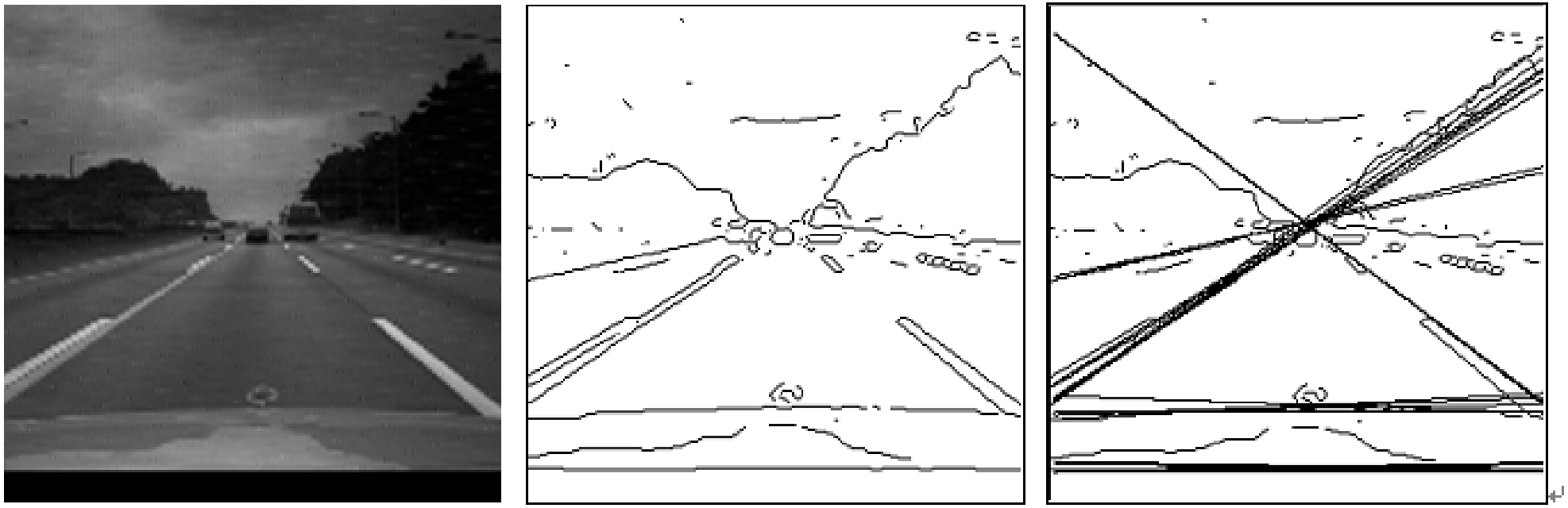
$$\begin{cases} a = \dfrac{A}{B} = \dfrac{1}{tan\theta} = -\dfrac{cos\theta}{sin\theta} \\ b = A = \dfrac{r}{sin\theta} \end{cases} \qquad \therefore sin\theta = \frac{r}{A}$$

$$y = -\frac{cos\theta}{sin\theta}x + \frac{r}{sin\theta}$$

$$\frac{r}{sin\theta} = \frac{cos\theta}{sin\theta}x + y$$

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

직선추출을 위한 HT의 예



하프변환을 이용한 도로의 차선 추출

허프 변환의 전체 과정

1. **극좌표계에서 누적행렬 구성**
   - 파라메터 공간(parameter space) 구성
2. **영상 화소의 직선 검사**
   - 영상에서 잡음제거 후, 에지를 추출(canny 등)한 이진영상 준비
3. **직선좌표에 대한 극좌표 누적행렬 구성**
   - 직선좌표를 극좌표로 변환하여 해당 위치 카운트를 누적
4. **누적행렬의 지역 최대값(local maxima) 선정**
   - 누적행렬을 작은 블록으로 나누고, 각 블록에서 최대값만 보존하고 나머지는 제거
5. **직선 선별**
   - 임계값 이상인 누적값 선택 및 정렬

(1) 파라메터 공간(parameter space) 구성

❖ 허프 변환 좌표계를 위한 행렬의 계산

$$h = \frac{\rho_{max}}{\Delta\rho} * 2$$

$$w = \frac{\theta_{max}}{\Delta\theta}$$

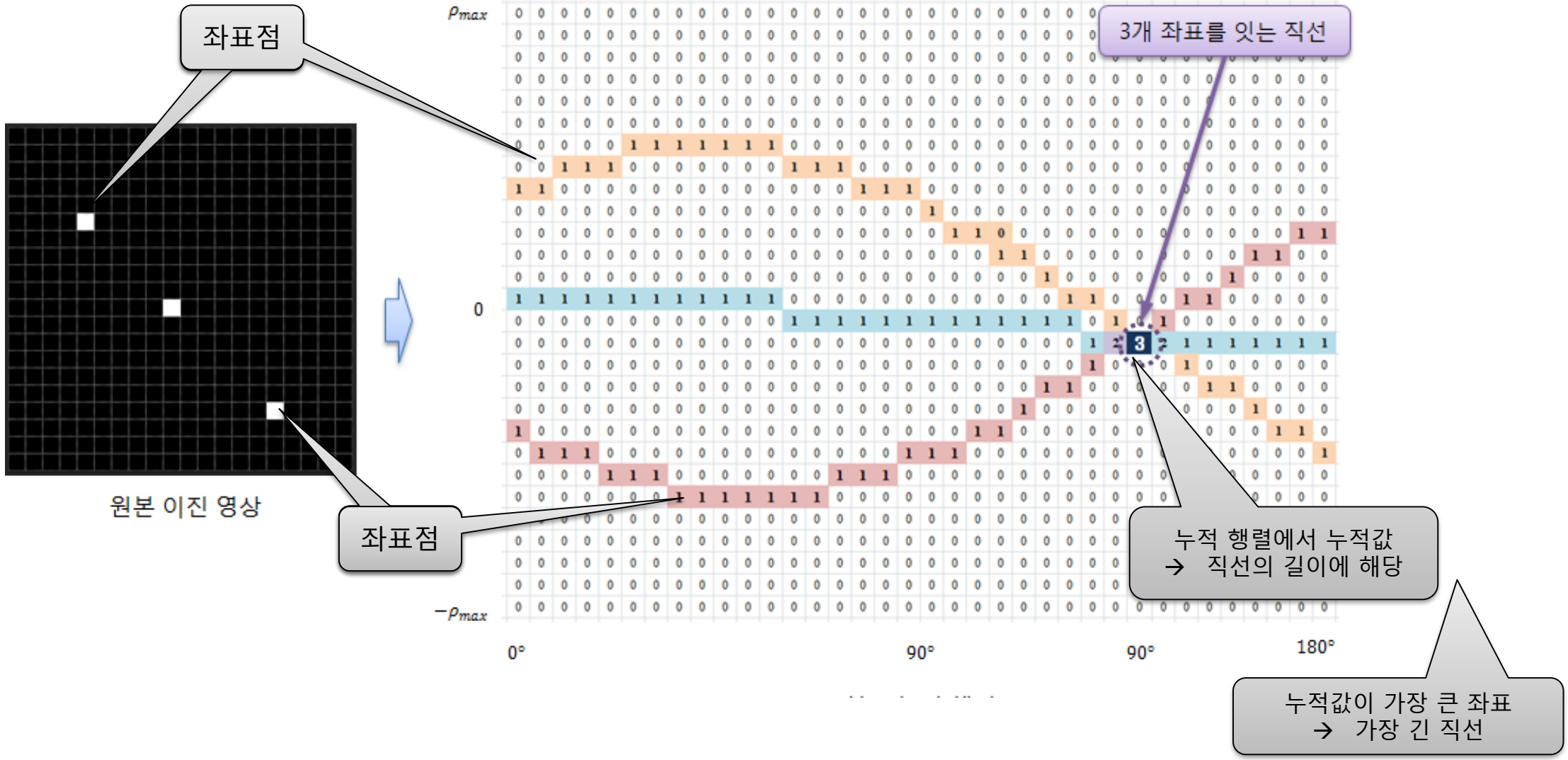
$$-\rho_{max} \leq \rho \leq \rho_{max}$$

$$0 \leq \theta < \theta_{max}$$

$$\rho_{max} = rows + cols$$

$$\theta_{max} = \pi$$

(3) 누적행렬 구성


좌표점
원본 이진 영상
좌표점
ρ_{max}
0
$-\rho_{max}$
0°
90°
90°
180°
3개 좌표를 잇는 직선
누적 행렬에서 누적값
→ 직선의 길이에 해당
누적값이 가장 큰 좌표
→ 가장 긴 직선

거리간격, 각도간격

```python
def accumulate(image, rho, theta):
    h, w = image.shape[:2]
    rows, cols = (h+w) * 2 // rho, int(np.pi / theta)          # 누적행렬 너비, 높이
    accumulate = np.zeros((rows, cols), np.int32)              # 직선 누적행렬

    sin_cos = [(np.sin(t*theta), np.cos(t*theta)) for t in range(cols)]   # 삼각 함수값 저장
    pts = np.where(image > 0)                                  # 넘파이 함수 활용 – 직선좌표 찾기

    polars = np.dot(sin_cos, pts).T                            # 행렬 곱으로 극좌표 계산
    polars = (polars / rho + rows / 2).astype('int')           # 해상도 변경 및 위치 조정

    for row in polars:
        for t, r in enumerate(row):                            # 각도, 수직 거리 가져옴
            accumulate[r, t] += 1                              # 극좌표에 누적
    return accumulate
```

누적 행렬

삼각 함수 행렬

직선 극좌표를 누적 행렬의 인덱스로 계산

직선 좌표들의 극좌표

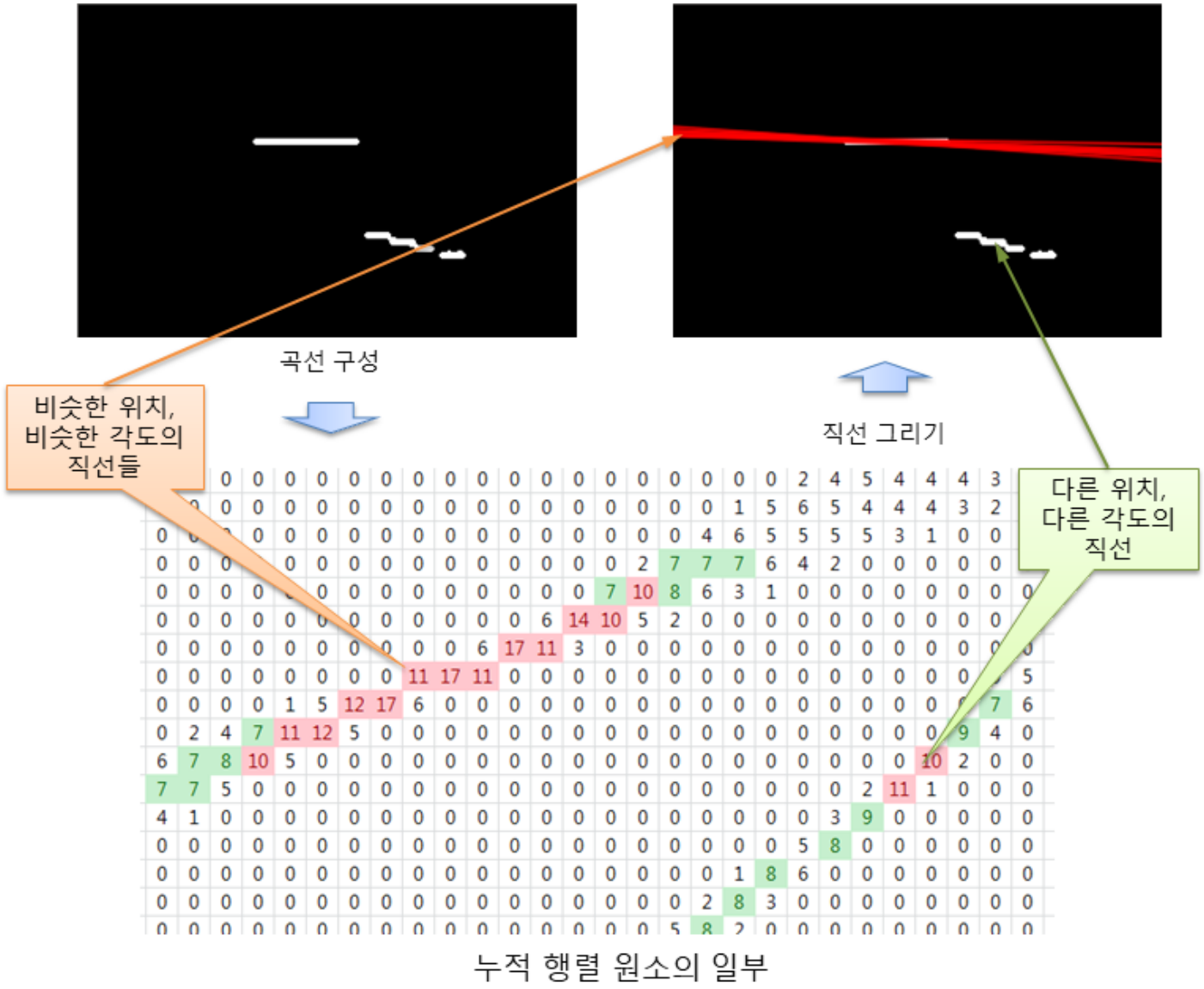
$$\begin{bmatrix} \sin\theta_0, & \cos\theta_0 \\ \sin\theta_1, & \cos\theta_1 \\ \sin\theta_2, & \cos\theta_2 \\ \cdots & \cdots \\ \cdots & \cdots \\ \sin\theta_m, & \cos\theta_m \end{bmatrix} \cdot \begin{bmatrix} y_o, & y_1, & y_2, & \cdots, & y_n \\ x_o, & x_1, & x_2, & \cdots, & x_n \end{bmatrix}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

0~180도 각도에 대한 삼각함수 값 행렬

직선 좌표들

```
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
```

❖ 한 지점에서 여러 직선 검출의 문제



(4) 누적행렬의 지역 최대값 선정

❖ 마스크를 이용해 지역 최대값 선정

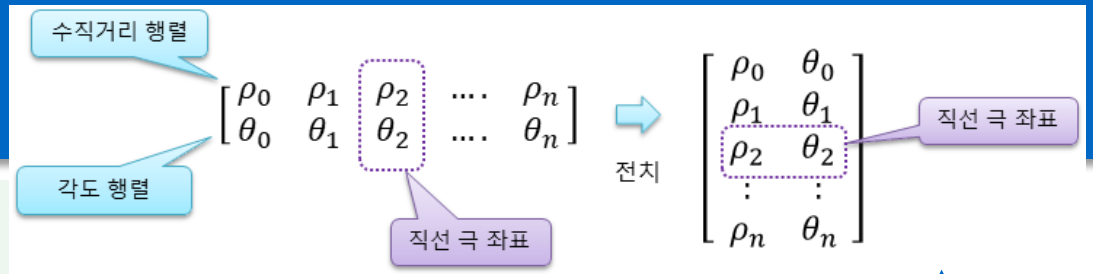
```
01  def masking(accumulate, h, w, thresh):
02      rows, cols = accumulate.shape[:2]
03      rcenter, tcenter = h//2, w//2                    # 마스크 크기 절반
04      dst = np.zeros(accumulate.shape, np.uint32)
05
06      for y in range(0, rows, h):                      # 누적 행렬 조회
07          for x in 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

(5) 직선 선별



극 좌표 행렬의 인덱스

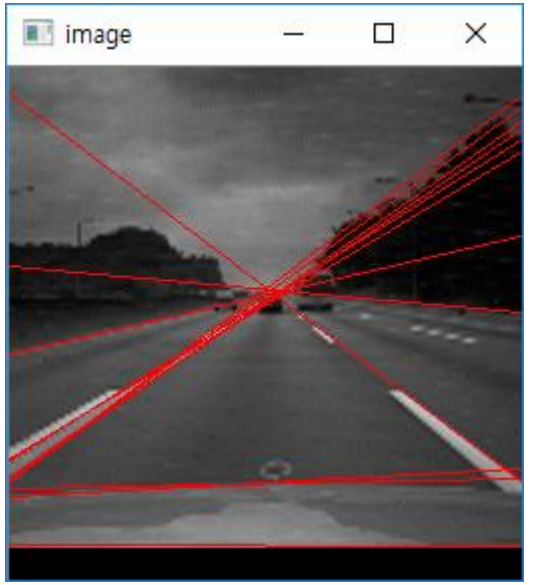
누적값 기준 정렬

```
01  def select_lines(acc_dst, rho, theta, thresh):
02      rows = acc_dst.shape[0]
03      r, t = np.where(acc_dst>thresh)                 # 임계값 이상 인덱스 가져옴
04
05      rhos = ((r - (rows / 2)) * rho)                 # 인덱스로 수직 거리 계산
06      radians = t * theta                             # 인덱스로 각도 계산
07      values = acc_dst[r, t]                          # 인덱스로 누적값 가져옴
08
09      idx = np.argsort(values)[::-1]                  # 내림차순 정렬 인덱스
10      lines = np.transpose([rhos, radians])           # 리스트 전치하여 행렬 생성
11      lines = lines[idx, :]                           # 누적값 기준으로 극좌표 정렬
12
13      return np.expand_dims(lines, axis=1)                # 1번(열) 차원 증가
```

직선추출을 위한 HT 함수

- **cv2.HoughLines(image, rho, theta, threshold, lines=None, srn=None, stn=None, min_theta=None, max_theta=None) -> lines**
  - image : 입력 에지 영상
  - rho : 누적행렬에서 ρ값의 간격. (e.g.) 1.0 → 1픽셀 간격
  - theta : 누적행렬에서 θ값의 간격. (e.g.) np.pi / 180
  - threshold : 누적행렬에서 직선으로 판단할 임계값
  - lines : 직선 파라미터(rho, theta) 정보를 담고 있는 np.ndarray. shape=(N, 1, 2). dtype=numpy.float32.
  - srn, stn : 멀티 스케일 허프 변환에서 ρ 해상도, θ 해상도를 나누는 값. 기본값은 0이고, 이 경우 일반 허프 변환 수행.
  - min_theta, max_theta : 검출할 선분의 최소, 최대 θ값

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    edges = cv2.Canny(gray, 50, 150)
    lines = cv2.HoughLines(edges, 1, np.pi/180, 80) # th = 80
    for line in lines:
        r, theta = line[0]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * r
        y0 = b * r
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))
        cv2.line(img, (x1, y1), (x2, y2), (0,0,255), 1)

    cv2.imshow('image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```

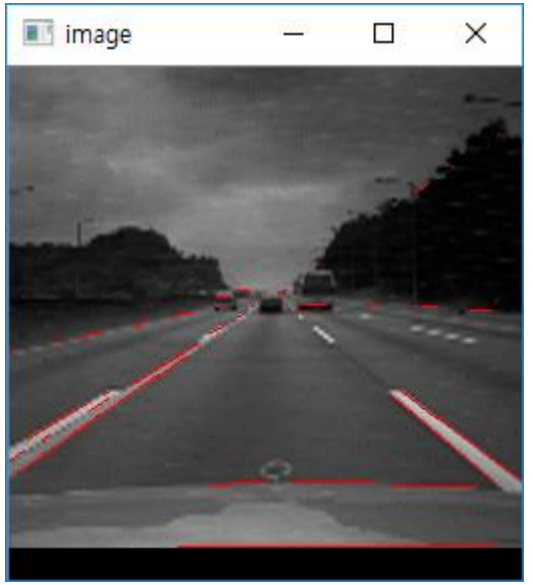


❖ **연산량을 줄이기 위해, 모든 점을 고려하지 않고 무작위로 선정한 픽셀에 대해 허프 변환을 수행하고 점차 그 수를 증가시키는 방법**

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

- image : 입력 에지 영상
- rho : 누적행렬에서 ρ값의 간격. (e.g.) 1.0 → 1픽셀 간격
- theta : 누적행렬에서 θ값의 간격. (e.g.) np.pi / 180
- threshold : 누적행렬에서 직선으로 판단할 임계값
- lines : 선분의 시작과 끝 좌표(x1, y1, x2, y2) 정보를 담고 있는 np.ndarray. shape=(N, 1, 4). dtype=numpy.int32.
- minLineLength : 검출할 선분의 최소 길이
- maxLineGap : 직선으로 간주할 최대 에지 점 간격

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    edges = cv2.Canny(gray, 50, 150)
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, 60) # th = 60

    for line in lines:
        for x1, y1, x2, y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), (0,0,255), 1)

    cv2.imshow('image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```



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 임포트
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)
```

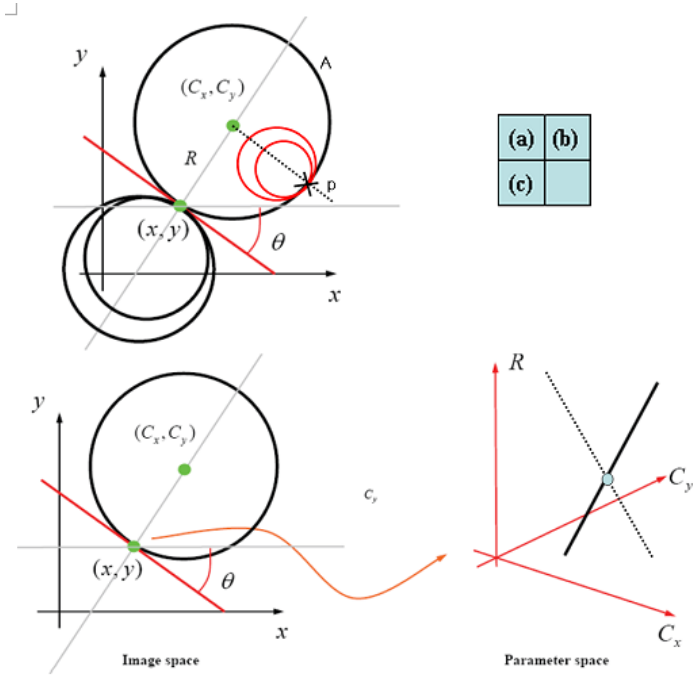
```
10  def draw_houghLines(src, lines, nline):                    # 검출 직선 그리기 함수
11      dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)            # 컬러 영상 변환
12      min_length = min(len(lines), nline)
13
14      for i in range(min_length):
15          rho, radian = lines[i, 0, 0:2]                     # 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
```

```
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)
```

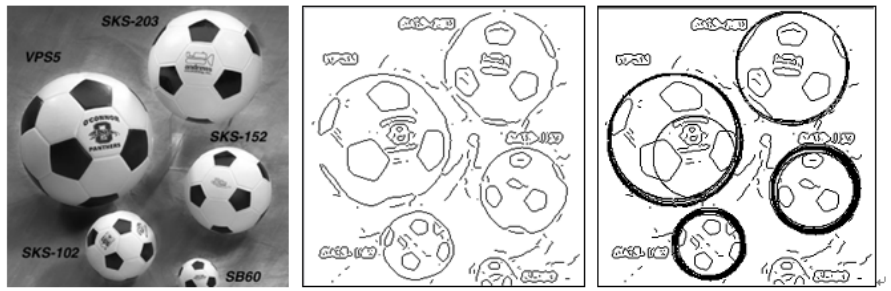
❖ 실행결과

원 추출을 위한 HT

- 원(circle)은 직선과 달리 3개의 파라메터를 가지고 있으며 이것은 원의 중심좌표 (c_x, c_y)와 원의 반지름 r를 가리킴
- 따라서 보팅을 위한 허프공간은 3차원이 되어야 하며 (c_x, c_y, r)과 같은 보팅 공간을 정의해야 함



원 추출을 위한 HT 의 적용



입력 영상 에지 추출 원 검출 결과

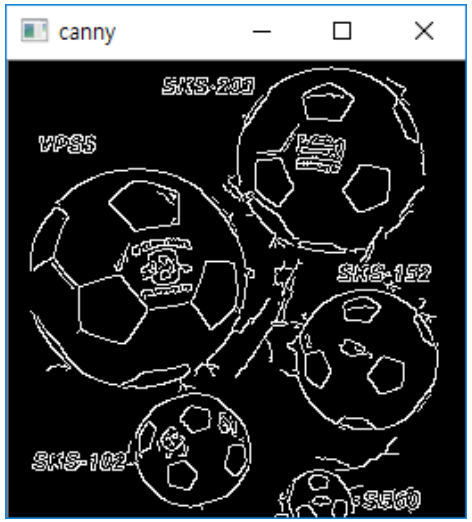
입력 영상 에지 추출 원 검출 결과

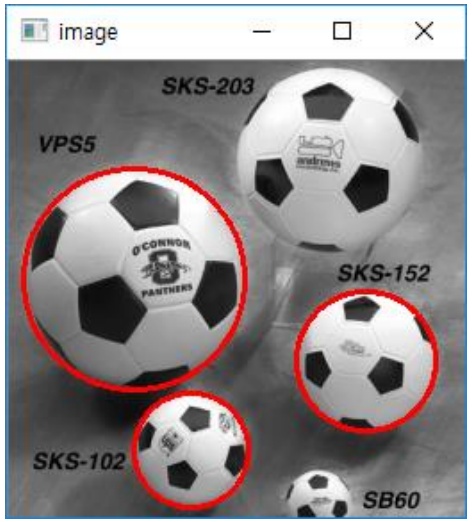
HT 에 의한 원의 추출

❖ cv.HoughCircles(image, method, dp, minDist, circles, param1, param2, minRadius, maxRadius) -> circles

- img : 입력 이미지, 1채널 배열
- method : 검출 방식 선택 (현재 cv2.HOUGH_GRADIENT만 가능)
- dp : 입력 영상과 누적행렬의 해상도 반비례율(보통 1)
  - 1 : 입력과 동일
- minDist : 원들의 중심간 최소 거리 (0: 에러, 0이면 동심원이 검출 불가하므로)
- circles(optional) : 검출 원 결과, N x 1 x 3 부동 소수점 배열 (x, y, 반지름)
- param1(optional) : Canny의 maxValue (최소 값은 최대 값의 1/2값을 전달)
- param2(optional) : voting 임계값(값이 작을수록 잘못된 원 검출)
- minRadius, maxRadius(optional) : 원의 최소 반지름, 최대 반지름 (0이면 이미지 전체의 크기)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    canny = cv2.Canny(gray, 50, 150)
    cv2.imshow('canny', canny)

    circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 5, param1=100,
                               param2=50, maxRadius=100)
    if circles is not None:
        print(circles)
        circles = np.uint16(np.around(circles[0,:]))
        for (x, y, r) in circles:
            cv2.circle(img, (x,y), r, (0, 0, 255), 1)

    cv2.imshow('image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
showImage()
```





2

해리스 코너검출

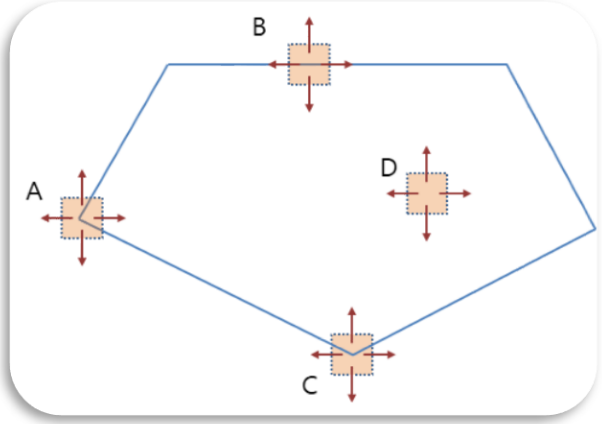
특징검출

❖ 무엇을 특징점으로 쓸 것인가?

- 에지?
  - 영상 구조 파악 및 객체 검출에는 도움이 되지만, 영상매칭에는 큰 도움이 되지 않음
  - 에지 강도와 방향 정보만 가지므로 영상 매칭하기엔 정보 부족

❖ 다른곳과 두드러지게 달라 풍부한 정보 추출 가능한 곳

- 에지 토막에서 곡률이 큰 지점을 코너로 검출
  - 코너 검출, dominant point 검출 등의 주제로 80년대 왕성한 연구
  - 90년대 소강 국면, 2000년대 사라짐
  - 더 좋은 대안이 떠올랐기 때문
- 지역 특징이라는 새로운 물줄기
  - 명암 영상에서 직접 검출



지역특징의 성질

❖ 지역 특징

- <위치,스케일,방향,특징벡터>=((*y*,*x*), *s*, *θ*, **x**)로 표현
  - 검출 단계: 위치와 스케일 알아냄
  - 기술 단계: 방향과 특징 벡터 알아냄

❖ 지역 특징이 만족해야 할 특성

- 반복성
- 분별력
- 지역성
- 정확성
- 적당한 양
- 계산 효율

❖ 이들 특성은 tradeoff 관계

코너찾기 알고리즘의 종류

❖ 모라벡

$$C = \min(S(0,1), S(0,-1), S(1,0), S(-1,0))$$

❖ 해리스 코너

$$C = det(\mathbf{A}) - k \times trace(\mathbf{A})^2 = (pq - r^2) - k(p+q)^2$$

❖ 헤시안의 행렬식

$$C = det(\mathbf{H}) = d_{yy}(\sigma) d_{xx}(\sigma) - d_{yx}(\sigma)^2$$

❖ LOG

$$C = \nabla^2 = trace(\mathbf{H}) = d_{yy}(\sigma) + d_{xx}(\sigma)$$

❖ 슈산

$$C = \begin{cases} q - usan\_area(r_0), & usan\_area(r_0) \le t_2 \\ 0, & \text{그렇지 않으면} \end{cases}$$

코너찾기 알고리즘의 특징

- **비최대 억제**
  - 이웃 화소보다 크지 않으면 억제됨 ⇒ 즉, 지역 최대만 특징점으로 검출됨
- **이동과 회전에 불변인가?**
  - 이동이나 회전 변환이 발생하여도 같은 지점에서 관심점이 검출되나? ⇒ 예
- **스케일에 불변인가?**
  - 스케일이 변해도 같은 지점에서 관심점이 검출되나? ⇒ 아니오
    - 연산자 크기가 고정되어 있어 그렇지 않다.
    - 스케일 변화에 대처하려면 연산자 크기를 조절하는 기능이 필수

모라벡(Moravec)

❖ 윈도우 $\omega(x,y)$내 영상 변화량(SSD: Sum of Squared Difference)

$$E(u,v) = \sum_y \sum_x \omega(x,y) \cdot (I(x+u, y+v) - I(x,y))^2$$

- 현재 화소에서 u, v 방향으로 이동했을 때의 밝기 변화량의 제곱
- (u,v)를 (1, 0), (1, 1), (0, 1), (-1, 1)의 4개 방향으로 한정

$$C = \min(S(0,1), S(0,-1), S(1,0), S(-1,0))$$

❖ 문제점

- 0과 1의 값만 갖는 이진 윈도우 사용으로 노이즈에 취약
- 4개 방향으로 한정시켰기 때문에 45도 간격의 에지만 고려

해리스(Harris)

- 이진 윈도우 $\omega(u,v)$ 대신에 점진적으로 변화하는 가우시안 마스크 $G(x,y)$ 적용

$$E(u,v) = \sum_y \sum_x G(x,y) \cdot (I(x+u, y+v) - I(x,y))^2$$

- 모든 방향에서 검출할 수 있도록 미분 도입

$$I(x+u, y+v) \cong I(x,y) + v d_y(x,y) + u d_x(x,y)$$

$$E(u,v) \cong \sum_y \sum_x G(x,y) \cdot (v d_y(x,y) + u d_x(x,y))^2$$

$$E(u,v) \cong \sum_y \sum_x G(x,y) \cdot (v d_y + u d_x)^2$$

$$= \sum_y \sum_x G(x,y) \cdot (v^2 d_y^2 + u^2 d_x^2 + 2vu d_x d_y)$$

$$= \sum_y \sum_x G(x,y) \cdot (u\ v) \begin{pmatrix} d_x^2 & d_x d_y \\ d_x d_y & d_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

$$= (u\ v) M \begin{pmatrix} u \\ v \end{pmatrix}, \qquad M = \sum_y \sum_x G(x,y) \begin{pmatrix} d_x^2 & d_x d_y \\ d_x d_y & d_y^2 \end{pmatrix}$$

특징 가능성을 직접 계산하는 대신 행렬 *M*의 식으로 정리

- **행렬 *M*에서 고유벡터를 구하면 경계선 방향에 수직인 벡터 두 개 얻음 : 행렬 M의 고유값(λ_1, λ_2)으로 코너 응답 함수(CRF) 계산**

$$R = \lambda_1, \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2$$

❖ **고유값 분해에 걸리는 시간을 최소화하기 위해, 고유값을 직접 계산하지 않고, 행렬식과 대각합을 통해 코너응답함수(CRF)로 사용**

$$M=\begin{pmatrix} d_x^2 & d_x d_y \\ d_x d_y & d_y^2 \end{pmatrix}=\begin{pmatrix} a & c \\ c & b \end{pmatrix}$$

$$R=det(M)-k\cdot trace(M)^2=(ab-c^2)-k\cdot(a+b)^2$$

- 2x2행렬의 고유벡터(eigen vector)가 상호수직 방향으로 얼마나 큰 값을 갖는지 측정
- dx : 가로방향의 미분값
- dy : 세로방향의 미분값
- 두 벡터가 수직일 경우, 가장 좋은 코너점

해리스 코너검출의 과정

- **Sobel로 미분행렬 계산**
  - 수직방향 및 수평방향 Sobel 마스크 적용
- **미분행렬의 곱 계산**
  - 두 방향 미분행렬의 곱$(dx * dx,\ dy * dy,\ dx * dy)$을 각각 계산
- **곱 행렬에 가우시안 마스크 적용**
  - Gaussian Blurring 수행 - cv2.GaussianBlur()
- **코너응답함수(CRF) 계산**
  - $R = \lambda_1\lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2$

$$M=\begin{pmatrix} d_x^2 & d_xd_y \\ d_xd_y & d_y^2 \end{pmatrix}=\begin{pmatrix} a & c \\ c & b \end{pmatrix}$$
$$R=det(M)-k\cdot trace(M)^2=(ab-c^2)-k\cdot(a+b)^2$$

- **비최대치 억제**
  - 지역적인 최대값을 찾아 그 값만 남기고 나머지 값은 모두 삭제

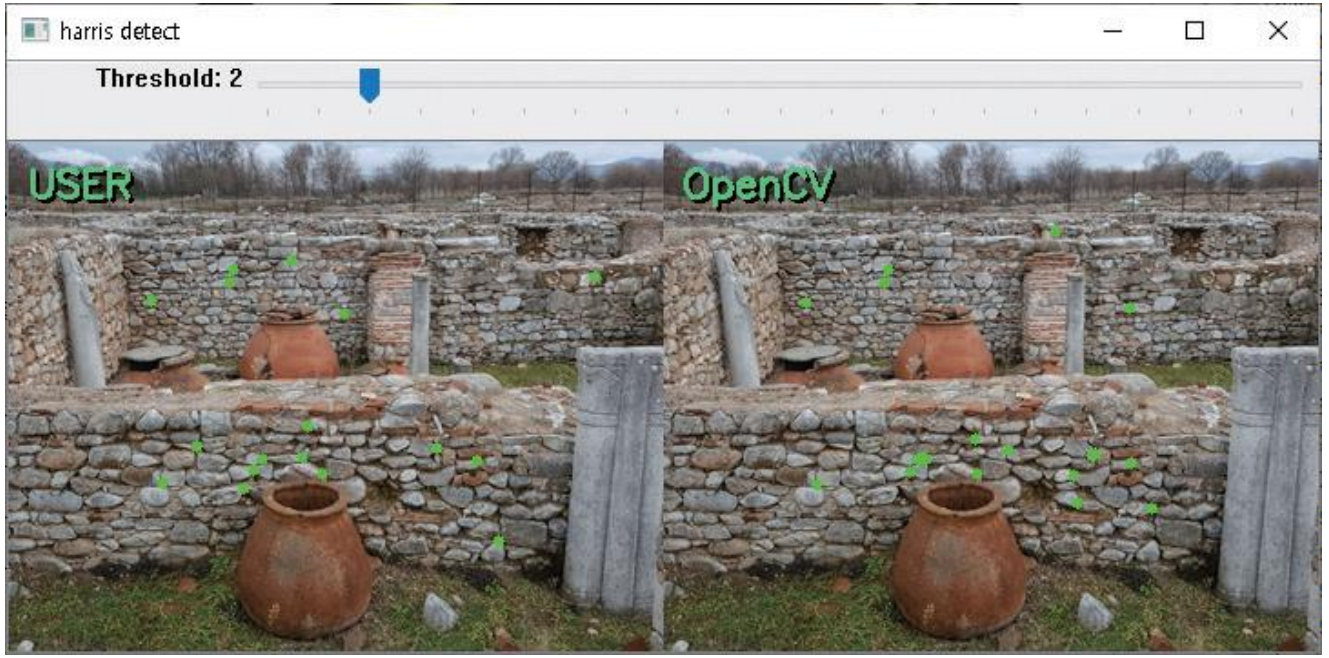
예제 10.2.1 헤리스 코너 검출 – 03.harris_dectect.py

```
01  import numpy as np, cv2
02  from Common.utils import put_string                    # 영상에 글쓰기 함수 임포트
03
04  def cornerHarris(image, ksize, k):                     # 해리스 코너 검출 함수
05      dx = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize)     # 미분 행렬– 수평 소벨 마스크
06      dy = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize)     # 미분 행렬– 수직 소벨 마스크
07
08      a = cv2.GaussianBlur(dx * dx, (5, 5), 0)           # 가우시안 블러링 수행
09      b = cv2.GaussianBlur(dy * dy, (5, 5), 0)
10      c = cv2.GaussianBlur(dx * dy, (5, 5), 0)
11
12      corner = (a * b - c**2) - k * (a + b)**2           # 코너 응답 함수–행렬 연산 적용
13      return corner
```

```
15  def drawCorner(corner, image, thresh):                    # 임계값 이상 코너 표시
16      cnt = 0
17      corner = cv2.normalize(corner, 0, 100, cv2.NORM_MINMAX)
18      corners = []
19      for i in range(1, corner.shape[0]-1):                 # 비최대치 억제
20          for j in range(1, corner.shape[1]-1):
21              neighbor = corner[i-1:i+2, j-1:j+2].flatten()      # 이웃 화소 가져옴
22              max = np.max(neighbor[1::2])                       # 상하좌우 값만
23              if thresh < corner[i, j] > max: corners.append((j, i)) # 코너 확정 좌표 저장
24
25      for pt in corners:                                         # 코너 확정 좌표 순회
26          cv2.circle(image, pt, 3, (0, 230, 0), -1)              # 좌표 표시
27      print("임계값: %2d, 코너 개수: %2d" %(thresh, len(corners)) )
28      return image
```

```
30  def onCornerHarris(thresh):                                    # 트랙바 콜백 함수
31      img1 = drawCorner(corner1, np.copy(image), thresh)
32      img2 = drawCorner(corner2, np.copy(image), thresh)
33
34      put_string(img1, "USER", (10, 30), "" )            # 영상에 문자표시 함수 호출
35      put_string(img2, "OpenCV", (10, 30), "")
36      dst = cv2.repeat(img1, 1, 2)                               # 두 개 영상을 한 윈도우 표시
37      dst[:, img1.shape[1]:, :] = img2
38      cv2.imshow("harris detect", dst)
39
40  image = cv2.imread("images/harris.jpg", cv2.IMREAD_COLOR)
41  if image is None: raise Exception("영상파일 읽기 에러")
42
43  blockSize = 4                                                  # 이웃 화소 범위
44  apertureSize = 3                                               # 소벨 마스크 크기
45  k = 0.04
46  thresh = 2                                                     # 코너 응답 임계값
```

```
47  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
48  corner1 = cornerHarris(gray, apertureSize, k)                  # 사용자 정의 함수
49  corner2 = cv2.cornerHarris(gray, blockSize, apertureSize, k)   # OpenCV 제공 함수
50
51  onCornerHarris(thresh)
52  cv2.createTrackbar("Threshold", "harris detect", thresh, 20, onCornerHarris)
53  cv2.waitKey(0)
```

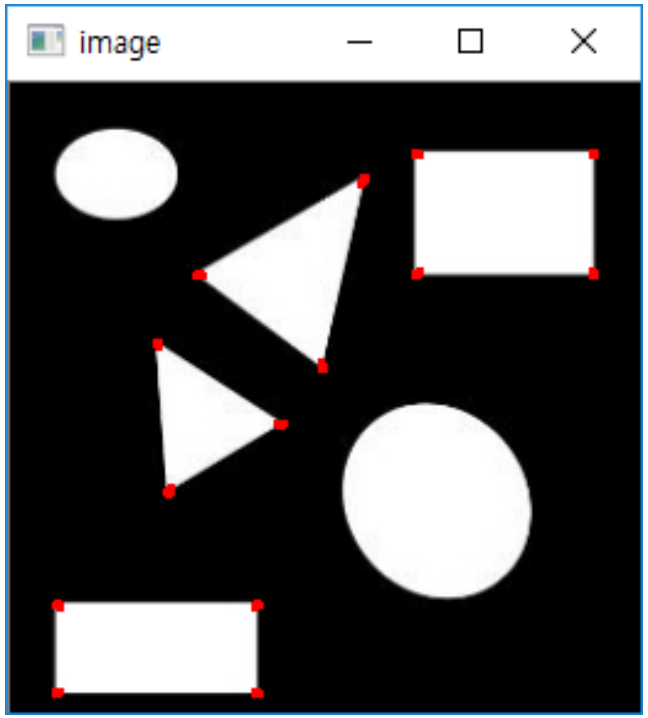


cv2.cornerHarris()

❖ **cv2.cornerHarris(src, blockSize, ksize, k, dst=None, borderType=None)** -> dst

- src : 입력영상(단일채널 8비트 또는 실수형)
- blockSize : 코너 응답 함수 계산에서 고려할 이웃 픽셀 크기(보통 2~5)
- ksize : (미분을 위한) 소벨 연산자를 위한 커널 크기(보통 3)
- k : 해리스 코너 검출 상수 (보통 0.04~0.06)
- dst : 해리스 코너 응답 계수. src와 같은 크기의 행렬(np.ndarray), dtype=np.float32
- borderType : 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER_DEFAULT

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    result = cv2.cornerHarris(gray, 2, 3, 0.04)
    result = cv2.dilate(result, None)

    img[result > 0.01*result.max()] = [0,0,255]

    cv2.imshow('image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```




3
Contours

Contours

- 동일한 색 또는 동일한 강도를 가지고 있는 영역의 경계선(boundary)을 연결한 선(예: 등고선)
- 어떤 대상의 외형을 파악하는 데 유용하게 사용됨
- 관련 opencv 함수
  - `cv2.findContours(image, mode, method[, contours[, hierarchy[, offset]]]) → image, contours, hierarchy`
    - 이미지는 이진 이미지(binary image) 사용 - 검은색 배경, 흰색 물체
    - 원본 이미지를 수정(in-place)하므로, 복사해서 사용한다.
  - `drawContours(image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]])`

findContours()

❖ **cv2.findContours(image, mode, method)** → contours, hierarchy

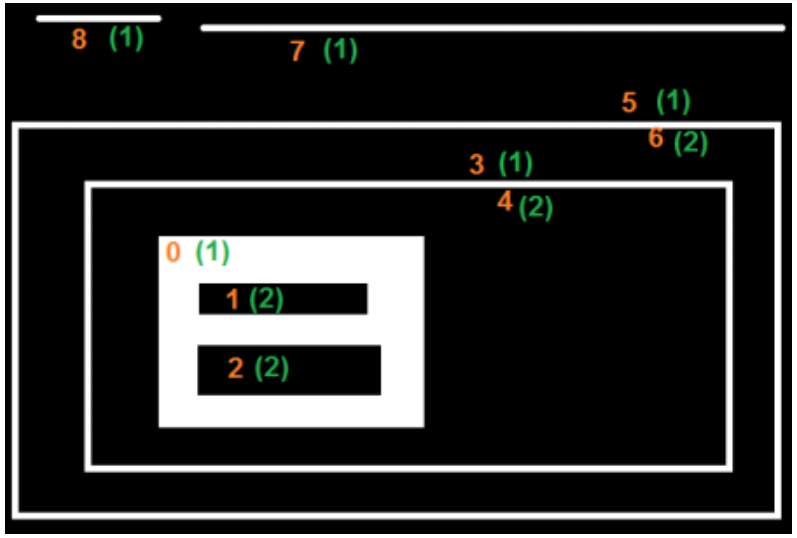
- **image** : 8-bit single-channel image. binary image
- **mode** : contour를 찾는 방법
  - cv2.RETR_EXTERNAL : 가장 바깥쪽 contour만 찾음
  - cv2.RETR_LIST : 계층구조 관계를 구성하지 않고 찾음
  - cv2.RETR_CCOMP : contour들을 2단계 계층구조로 구성(1단계: 외곽경계, 2단계: hole경계)
  - cv2.RETR_TREE : 모든 contour들의 모든 계층구조 관계를 tree로 구성
- **method** : contour를 찾을 때 사용하는 근사방법
  - cv2.CHAIN_APPROX_NONE : 모든 contours point 저장.
  - cv2.CHAIN_APPROX_SIMPLE : contours line을 그릴 수 있는 point 만 저장(ex: 사각형이면 4개 point)
  - cv2.CHAIN_APPROX_TC89_L1 : Teh_Chin 연결 근사 알고리즘 L1 버전 적용
  - cv2.CHAIN_APPROX_TC89_KCOS : Teh_Chin 연결 근사 알고리즘 KCOS 버전 적용

계층구조(hierarchy)

- **hierarchy**
  - shape : (1, # of contours, 4)
  - **[Next, Previous, FirstChild, Parent]**
    - Next : 다음 contour의 index
    - Previous : 이전 contour의 index
    - FirstChild : 첫번째 내부 contour의 index
    - Parent : 현재 contour의 부모 index(-1이면, 최외곽 contour)
- **cv2.RETR_CCOMP**
  - 2단계로만 계층구조를 표현
    - level-1 : boundary contour
    - level-2 : hole contour
- **cv2.RETR_TREE**
  - 모든 contour의 포함관계를 tree로 표현

❖ cv2.RETR_CCOMP

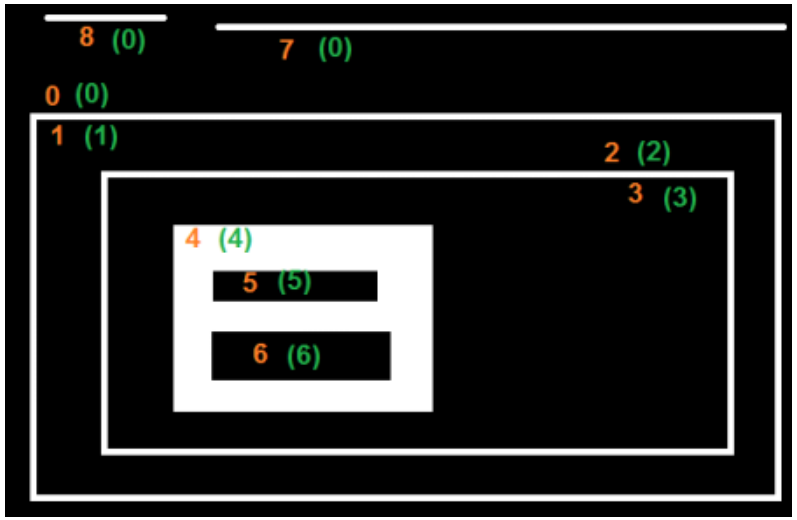
- 2단계로만 계층구조를 표현
  - level-1 : boundary contour
  - level-2 : hole contour



적색 : 번호
녹색 : (레벨)

❖ cv2.RETR_TREE

- 모든 contour의 포함관계를 tree로 표현

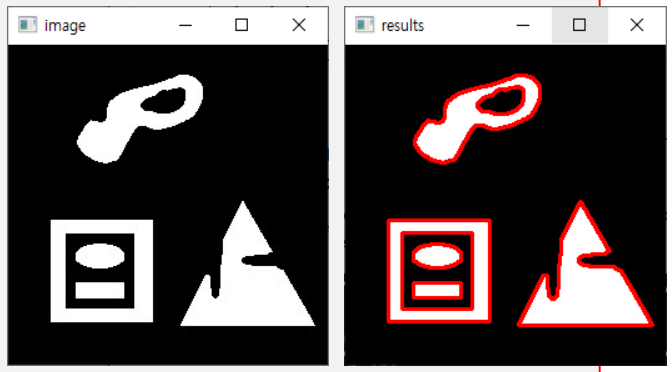


drawContours()

❖ **cv2.drawContours(image, contours, contourIdx, color, thickness, lineType)**

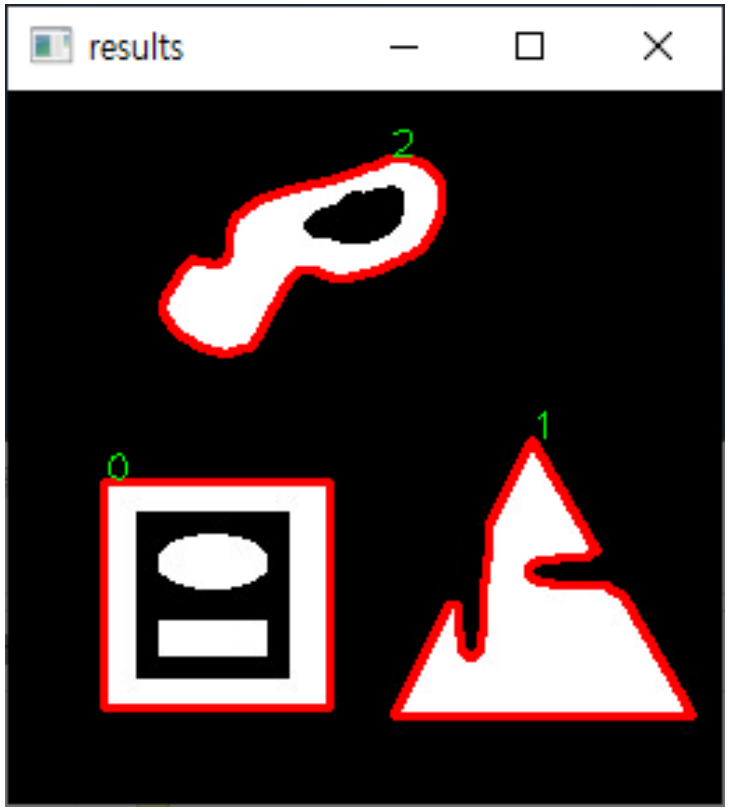
- image : 원본 이미지
- contours : contours 정보 리스트
- contourIdx : contours list type에서 몇 번째 contours line을 그릴 것인지 지정(-1 이면 전체)
- color : contours line의 색상
- thickness : contours line의 두께(음수이면 contours line의 내부를 채움)
- lineType : 선의 형태
  - cv2.FILLED, cv2.LINE_4, cv2.LINE_8, cv2.LINE_AA

```
import cv2
import numpy as np
import random

img = cv2.imread('poly.jpg', cv2.IMREAD_COLOR)
cv2.imshow('image', img)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binImg = cv2.threshold(grayImg, 127, 255, cv2.THRESH_OTSU)

contours, hierarchy = cv2.findContours(
            binImg, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0,0,255), 2)

cv2.imshow('results', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

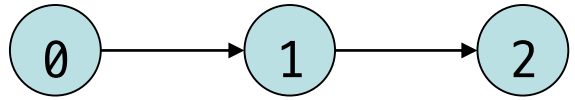


hierarchy(RETR_EXTERNAL)

```python
contours, hierarchy = cv2.findContours(binImg, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for i, contour in enumerate(contours):
    cv2.drawContours(img, [contour], -1, (0,0,255), 2)
    cv2.putText(img, str(i), tuple(contour[0][0]), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 1)
print(f"# of contours : {len(contours)}")
print(f"hierarchy(cv2.RETR_EXTERNAL)\n{hierarchy}")
```



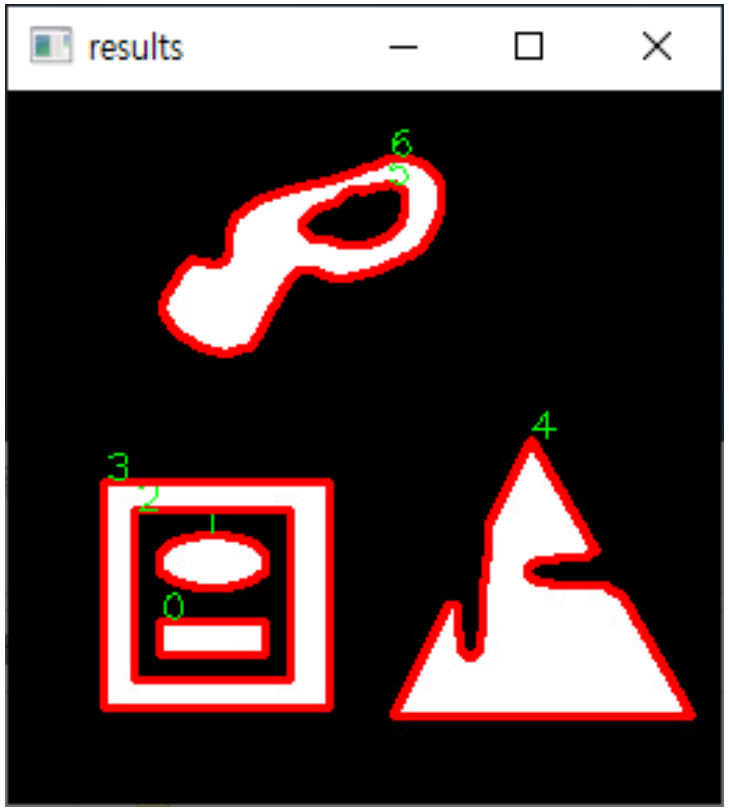

```
# of contours : 3
hierarchy(cv2.RETR_EXTERNAL)
[[[ 1 -1 -1 -1]
  [ 2  0 -1 -1]
  [-1  1 -1 -1]]]
```

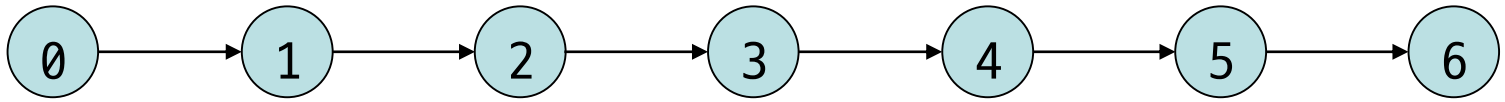


[Next, Previous, FirstChild, Parent]

hierarchy(RETR_LIST)

```python
contours, hierarchy = cv2.findContours(binImg, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
for i, contour in enumerate(contours):
    cv2.drawContours(img, [contour], -1, (0,0,255), 2)
    cv2.putText(img, str(i), tuple(contour[0][0]), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 1)
print(f"# of contours : {len(contours)}")
print(f"hierarchy(cv2.RETR_LIST)\n{hierarchy}")
```



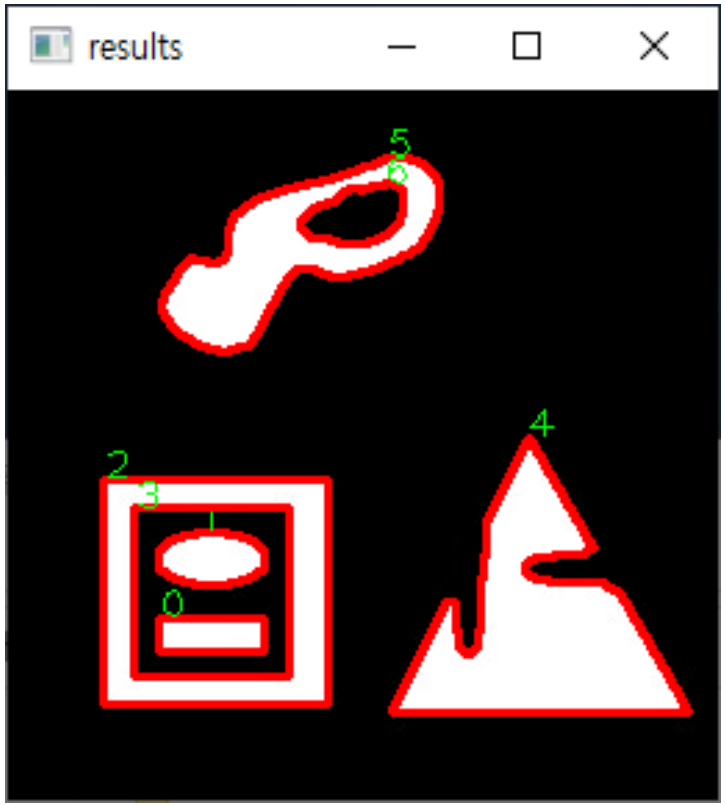



```
# of contours : 7
hierarchy(cv2.RETR_LIST)
[[[ 1 -1 -1 -1]
  [ 2  0 -1 -1]
  [ 3  1 -1 -1]
  [ 4  2 -1 -1]
  [ 5  3 -1 -1]
  [ 6  4 -1 -1]
  [-1  5 -1 -1]]]
```

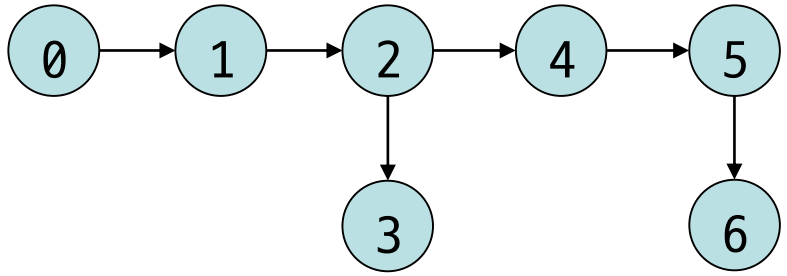
[Next, Previous, FirstChild, Parent]

hierarchy(RETR_CCOMP)

```python
contours, hierarchy = cv2.findContours(binImg, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
for i, contour in enumerate(contours):
    cv2.drawContours(img, [contour], -1, (0,0,255), 2)
    cv2.putText(img, str(i), tuple(contour[0][0]), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 1)
print(f"# of contours : {len(contours)}")
print(f"hierarchy(cv2.RETR_CCOMP)\n{hierarchy}")
```



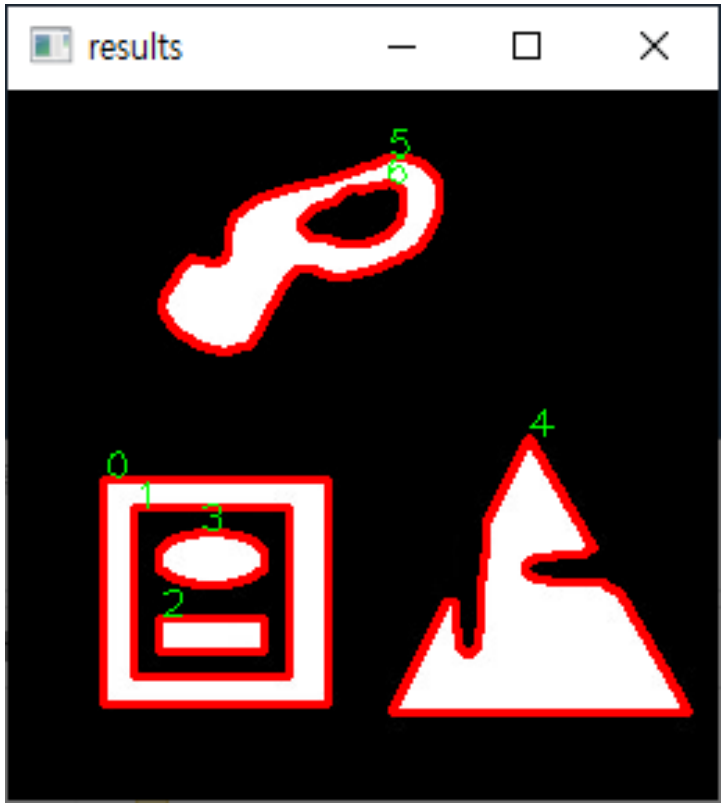

```
# of contours : 7
hierarchy(cv2.RETR_CCOMP)
[[[ 1 -1 -1 -1]
  [ 2  0 -1 -1]
  [ 4  1  3 -1]
  [-1 -1 -1  2]
  [ 5  2 -1 -1]
  [-1  4  6 -1]
  [-1 -1 -1  5]]]
```



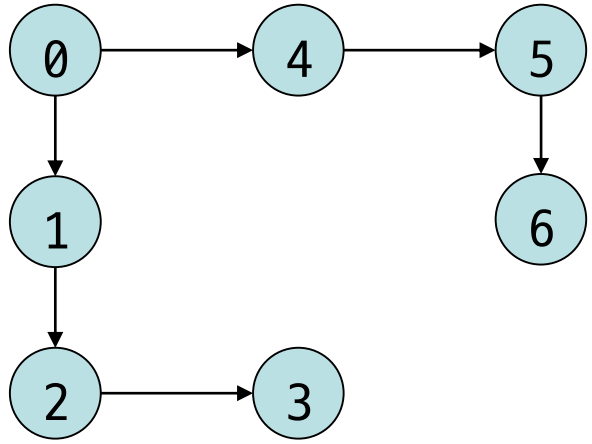
[Next, Previous, FirstChild, Parent]

hierarchy(RETR_TREE)

```python
contours, hierarchy = cv2.findContours(binImg, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
for i, contour in enumerate(contours):
    cv2.drawContours(img, [contour], -1, (0,0,255), 2)
    cv2.putText(img, str(i), tuple(contour[0][0]), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 1)
print(f"# of contours : {len(contours)}")
print(f"hierarchy(cv2.RETR_CCOMP)\n{hierarchy}")
```

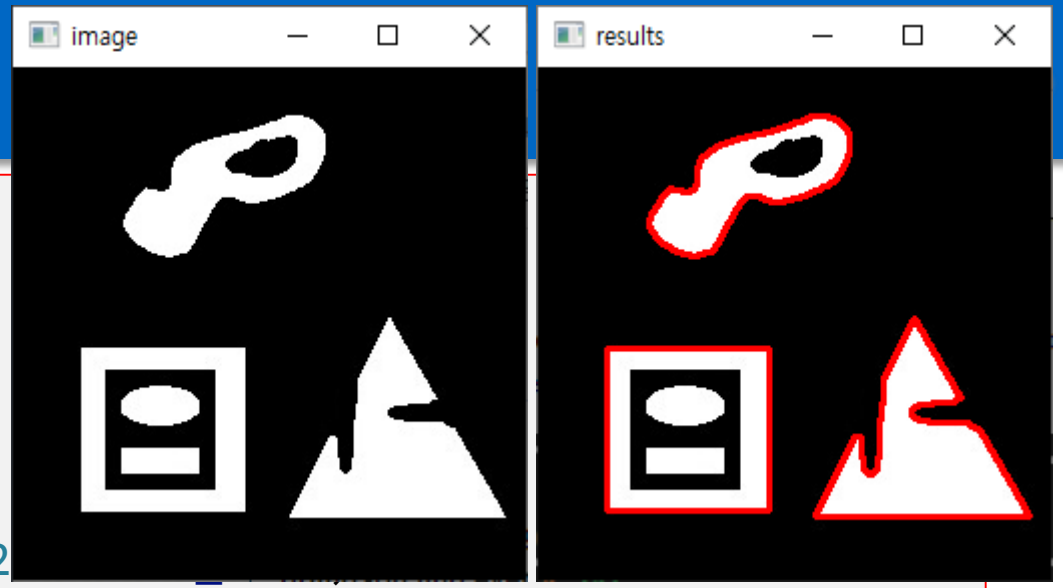



```
# of contours : 7
hierarchy(cv2.RETR_TREE)
[[[ 4 -1  1 -1]
  [-1 -1  2  0]
  [ 3 -1 -1  1]
  [-1  2 -1  1]
  [ 5  0 -1 -1]
  [-1  4  6 -1]
  [-1 -1 -1  5]]]
```



[Next, Previous, FirstChild, Parent]

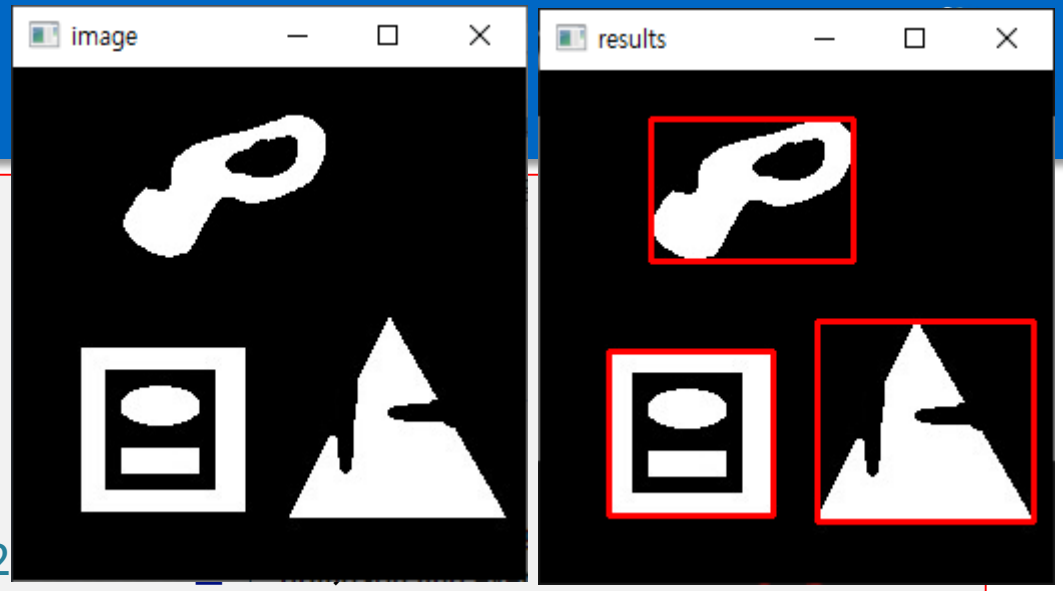
면적 계산




```
import cv2
import numpy as np

img = cv2.imread('poly.jpg', cv2.IMREAD_COLOR)
cv2.imshow('image', img)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binImg = cv2.threshold(grayImg, 127, 255, cv2

contours, hierarchy = cv2.findContours(binImg, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
print(f"# of contours : {len(contours)}")
for contour in contours:
    cv2.drawContours(img, [contour], -1, (0,0,255), 2)
    area = cv2.contourArea(contour)
    print(f"area = {area}")

cv2.imshow('results', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
# of contours : 3
area = 6561.0
area = 4734.5
area = 3372.5
```

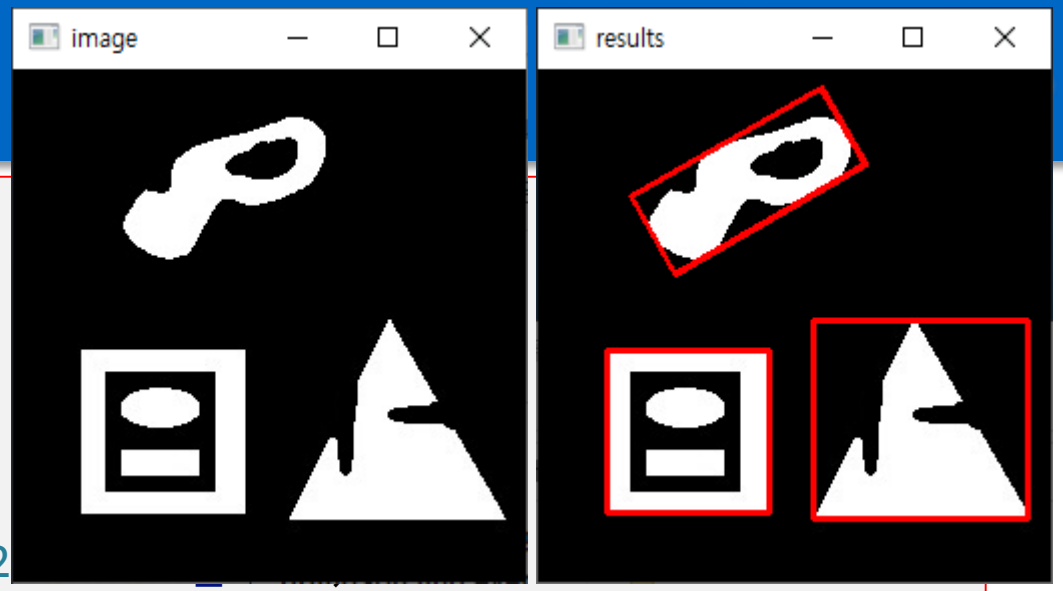
경계사각형 그리기




```
import cv2
import numpy as np

img = cv2.imread('poly.jpg', cv2.IMREAD_COLOR)
cv2.imshow('image', img)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binImg = cv2.threshold(grayImg, 127, 255, cv2

contours, hierarchy = cv2.findContours(binImg, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
print(f"# of contours : {len(contours)}")
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    cv2.rectangle(img, (x, y), (x+w, y+h), (0,0,255), 2)

cv2.imshow('results', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
# of contours : 3
```

최소인접사각형 그리기

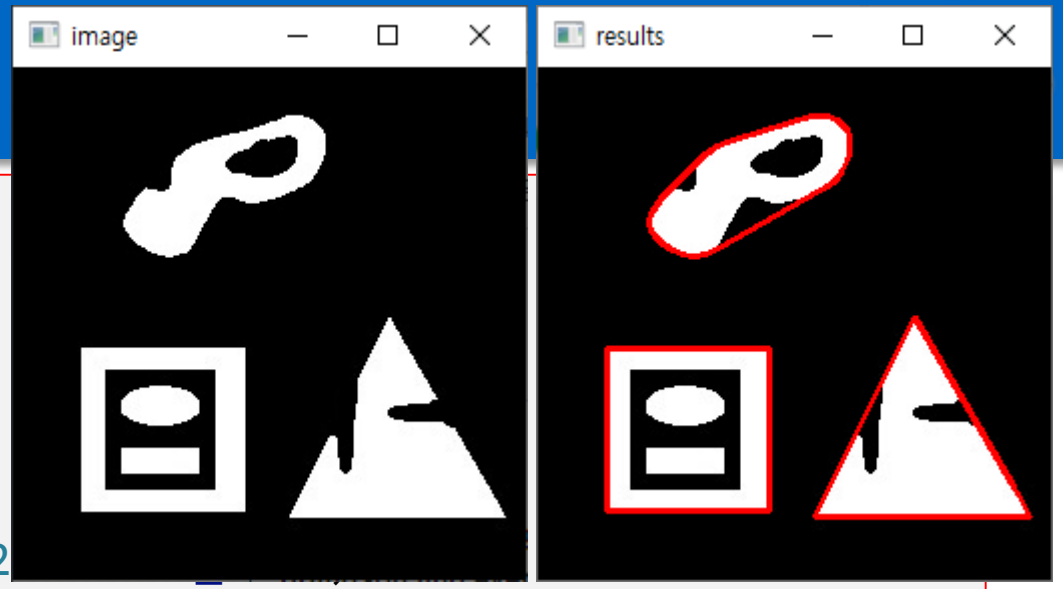



```
import cv2
import numpy as np

img = cv2.imread('poly.jpg', cv2.IMREAD_COLOR)
cv2.imshow('image', img)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binImg = cv2.threshold(grayImg, 127, 255, cv2

contours, hierarchy = cv2.findContours(binImg, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
print(f"# of contours : {len(contours)}")
for contour in contours:
    rect = cv2.minAreaRect(contour)
    box = cv2.boxPoints(rect)
    box = box.astype("intp") # box = np.int0(box) # int0: integer for index pointer
    cv2.drawContours(img, [box], 0, (0,0,255), 2)

cv2.imshow('results', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
# of contours : 3
```

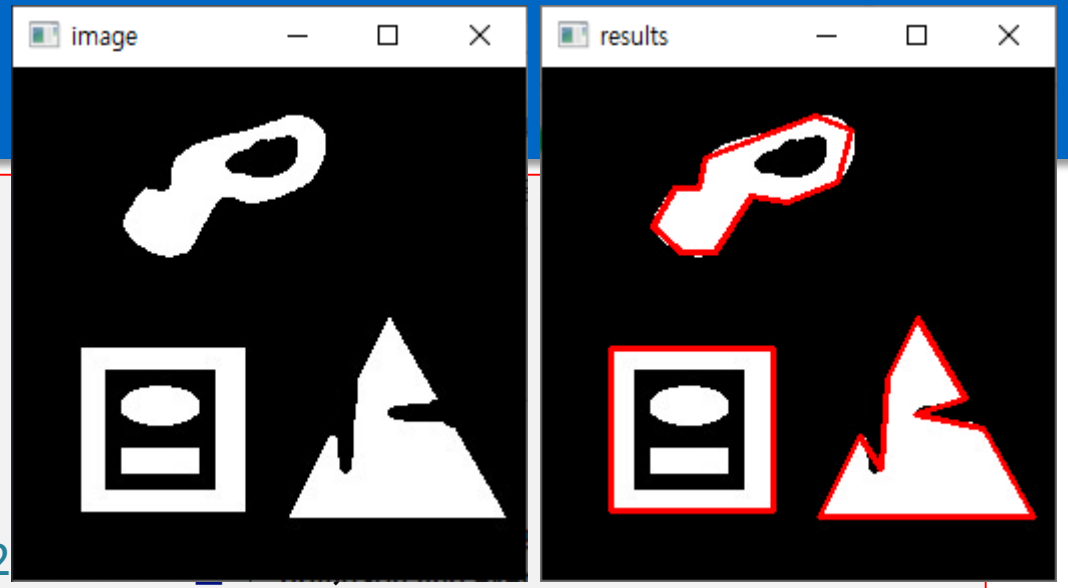
Convex Hull

```
import cv2
import numpy as np

img = cv2.imread('poly.jpg', cv2.IMREAD_COLOR)
cv2.imshow('image', img)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binImg = cv2.threshold(grayImg, 127, 255, cv2

contours, hierarchy = cv2.findContours(binImg, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
print(f"# of contours : {len(contours)}")
for contour in contours:
    hull = cv2.convexHull(contour)
    cv2.drawContours(img, [hull], -1, (0,0,255), 2)

cv2.imshow('results', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




```
# of contours : 3
```

근사화




```
import cv2
import numpy as np

img = cv2.imread('poly.jpg', cv2.IMREAD_COLOR)
cv2.imshow('image', img)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binImg = cv2.threshold(grayImg, 127, 255, cv2

contours, hierarchy = cv2.findContours(binImg, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
print(f"# of contours : {len(contours)}")
for contour in contours:
    epsilon = 0.015 * cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True)
    print(f"approx length = {len(approx)}")
    cv2.drawContours(img, [approx], -1, (0,0,255), 2)

cv2.imshow('results', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
# of contours : 3
approx length = 4
approx length = 9
approx length = 11
```

4

k-NN 분류기

k-NN 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
- k-NN은 기계학습 알고리즘 중 하나
  - 기계학습과 관련된 대부분의 함수들은 **cv2.ml** 모듈에 구현되어 있음

❖ '가장 가까운 거리'

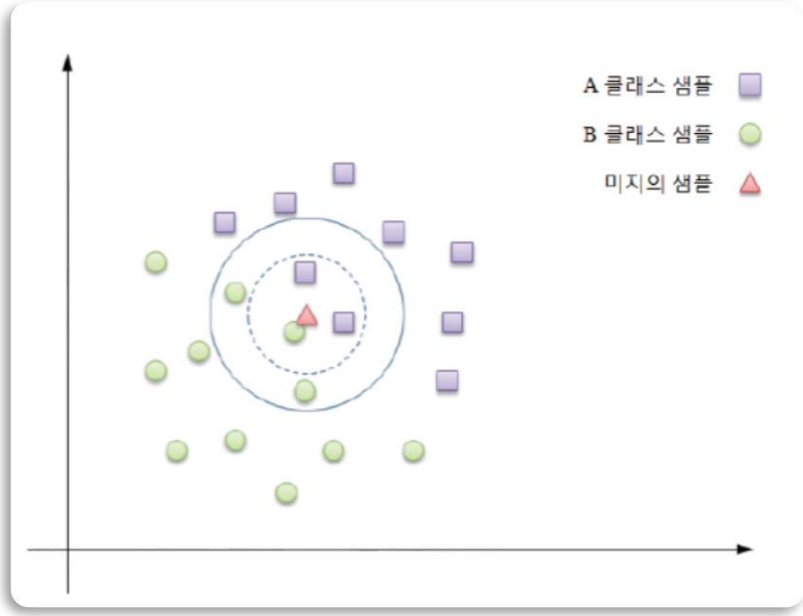
- 미지의 샘플과 학습 클래스 샘플간의 유사도가 가장 높은 것을 의미
- 유클리드 거리(euclidean distance), 해밍 거리(hamming distance), 차분 절대값 등

❖ 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

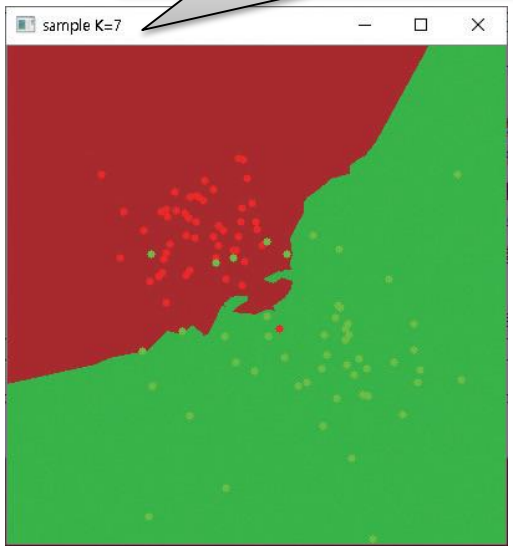


예제 10.3.1 임의 좌표 생성 – 04.kNN_exam.py

```
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))                    # 정수 원소 튜플
06          cv2.circle(image, pt, 3, color, cv2.FILLED)
07
08  nsample = 50                                         # 그룹당 학습 데이터 수
09  traindata = np.zeros((nsample*2, 2), np.float32)     # 전체 학습 데이터 행렬
10  label = np.zeros((nsample*2, 1), np.float32)         # 레이블 행렬 생성
11
12  cv2.randn(traindata[:nsample], 150, 30)              # 정규분포 랜덤 값 생성
13  cv2.randn(traindata[nsample:], 250, 60)
14  label[:nsample], label[nsample:] = 0, 1              # 레이블 기준값 지정
15
16  K = 7
17  knn = cv2.ml.KNearest_create()                       # kNN 클래스로 객체 생성
18  knn.train(traindata, cv2.ml.ROW_SAMPLE, classlable)  # 학습 수행
19
```

```
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)
```

K를 7로 지정하여 분류 수행 결과



K를 15 로 지정하여 분류 수행 결과