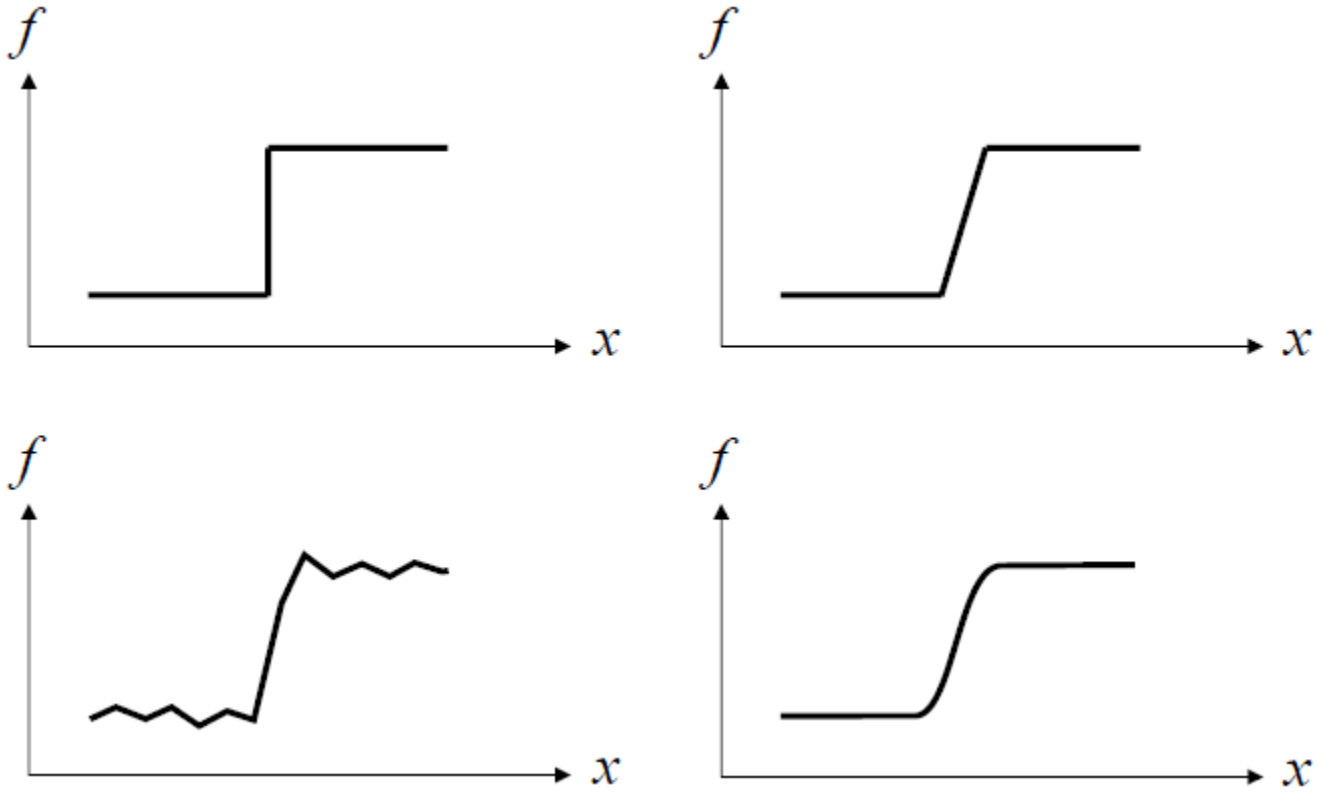
영상의 특징 추출

에지 검출과 미분

에지(edge)

- 영상에서 픽셀의 밝기 값이 급격하게 변하는 부분
- 일반적으로 배경과 객체, 또는 객체와 객체의 경계



영상의 미분과 소벨 필터

1차 미분의 근사화(approximation)

- 전진차분(Forward difference):

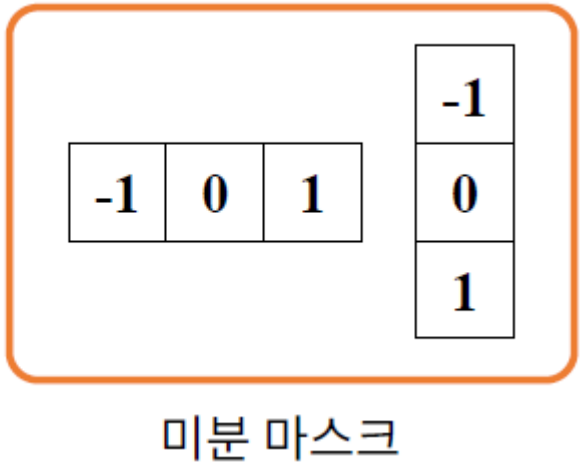
$$\frac{\partial I}{\partial x} \cong \frac{I(x+h)-I(x)}{h}$$

- 후진차분(Backward difference):

$$\frac{\partial I}{\partial x} \cong \frac{I(x)-I(x-h)}{h}$$

- 중앙차분(Centered difference):

$$\frac{\partial I}{\partial x} \cong \frac{I(x+h)-I(x-h)}{2h}$$



미분 마스크

영상의 미분과 소벨 필터

다양한 미분 마스크

가로 방향:

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

Prewitt

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

Sobel

-3	0	3
-10	0	10
-3	0	3

Scharr

세로 방향:

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

Prewitt

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

Sobel

-3	-10	-3
0	0	0
3	10	3

Scharr

영상의 미분과 소벨 필터

소벨 필터를 이용한 미분 함수

cv2.Sobel(src, ddepth, dx, dy, dst=None, ksize=None, scale=None, delta=None, borderType=None) -> dst

- src: 입력 영상
- ddepth: 출력 영상 데이터 타입. -1이면 입력 영상과 같은 데이터 타입을 사용.
- dx: x 방향 미분 차수.
- dy: y 방향 미분 차수.
- dst: 출력 영상(행렬)
- ksize: 커널 크기. 기본값은 3.
- scale: 연산 결과에 추가적으로 곱할 값. 기본값은 1.
- delta: 연산 결과에 추가적으로 더할 값. 기본값은 0.
- borderType: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER_DEFAULT.

대부분 dx=1, dy=0, ksize=3 또는
dx=0, dy=1, ksize=3 으로 지정.

영상의 미분과 소벨 필터

샤르 필터를 이용한 미분 함수

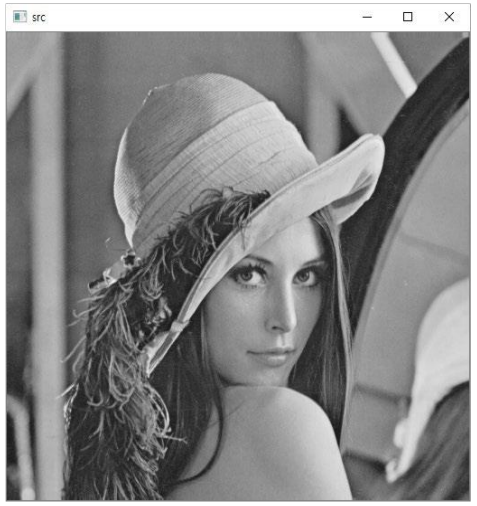
cv2.Scharr(src, ddepth, dx, dy, dst=None, scale=None, delta=None, borderType=None)-> dst

- src: 입력 영상
- ddepth: 출력 영상 데이터 타입. -1이면 입력 영상과 같은 데이터 타입을 사용.
- dx: x 방향 미분 차수.
- dy: y 방향 미분 차수.
- dst: 출력 영상(행렬)
- ksize: 커널 크기. 기본값은 3.
- scale: 연산 결과에 추가적으로 곱할 값. 기본값은 1.
- delta: 연산 결과에 추가적으로 더할 값. 기본값은 0.
- borderType: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER_DEFAULT.

영상의 미분과 소벨 필터

소벨 필터를 이용한 영상의 미분 예제

```python
import sys
import numpy as np
import cv2

src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

if src is None:
    print('Image load failed!')
    sys.exit()

dx = cv2.Sobel(src, -1, dx: 1, dy: 0, delta=128)
dy = cv2.Sobel(src, -1, dx: 0, dy: 1, delta=128)

cv2.imshow( winname: 'src', src)
cv2.imshow( winname: 'dx', dx)
cv2.imshow( winname: 'dy', dy)
cv2.waitKey()

cv2.destroyAllWindows()
```



그래디언트

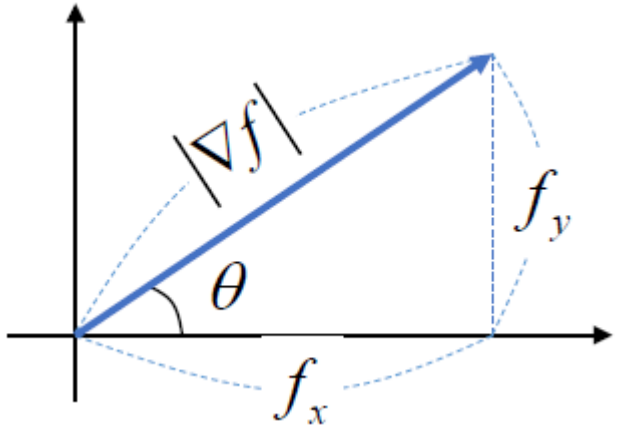
영상의 그래디언트(gradient)

- 함수 f(x, y)를 x축과 y축으로 각각 편미분(partial derivative)하여 벡터 형태로 표현한 것

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = f_x \mathbf{i} + f_y \mathbf{j}$$

- 그래디언트 크기: $\left|\nabla f\right| = \sqrt{f_x^{\,2} + f_y^{\,2}}$

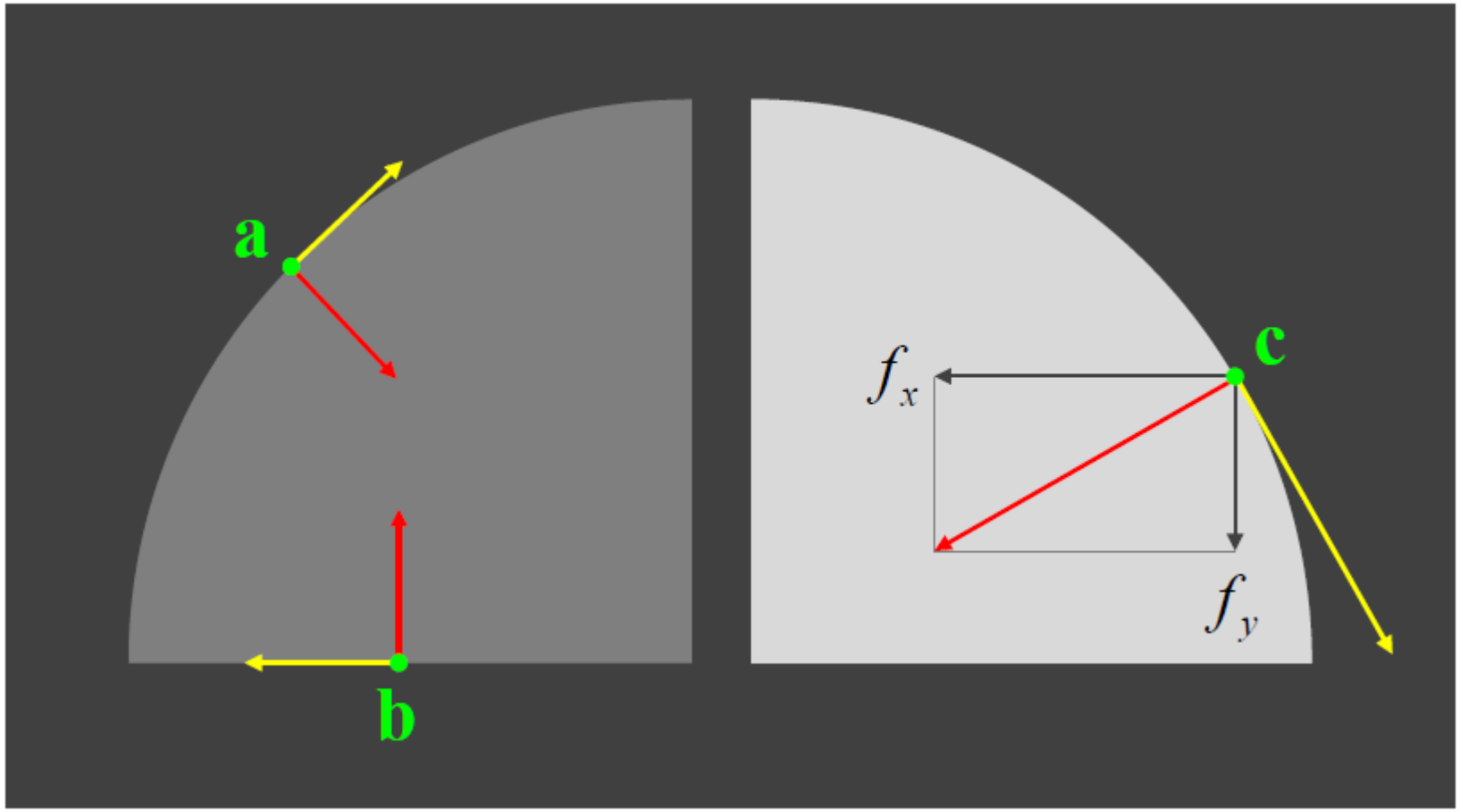
- 그래디언트 방향: $\theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$



그래디언트

실제 영상에서 구한 그래디언트 크기와 방향

- 그래디언트 크기: 픽셀 값의 차이 정도, 변화량
- 그래디언트 방향: 픽셀 값이 가장 급격하게 증가하는 방향



그래디언트

2D 벡터의 크기 계산 함수

cv2.magnitude(x, y, magnitude=None) -> magnitude

- x: 2D 벡터의 x 좌표 행렬. 실수형.
- y: 2D 벡터의 y 좌표 행렬. x와 같은 크기. 실수형.
- magnitude: 2D 벡터의 크기 행렬. x와 같은 크기, 같은 타입.

$$\text{magnitude}(I) = \sqrt{\text{x}(I)^2 + \text{y}(I)^2}$$

그래디언트

2D 벡터의 방향 계산 함수

cv2.phase(x, y, angle=None, angleInDegrees=None)-> angle

- x: 2D 벡터의 x 좌표 행렬. 실수형.
- y: 2D 벡터의 y 좌표 행렬. x와 같은 크기. 실수형.
- angle: 2D 벡터의크기행렬. x와 같은 크기, 같은 타입.

  $\text{angle}(I)=\text{atan2}(y(I), x(I))$

  만약x(I)=y(I)=0이면 angle은 0으로 설정됨.
- angleInDegrees: True이면 각도 단위, False이면 래디언 단위.

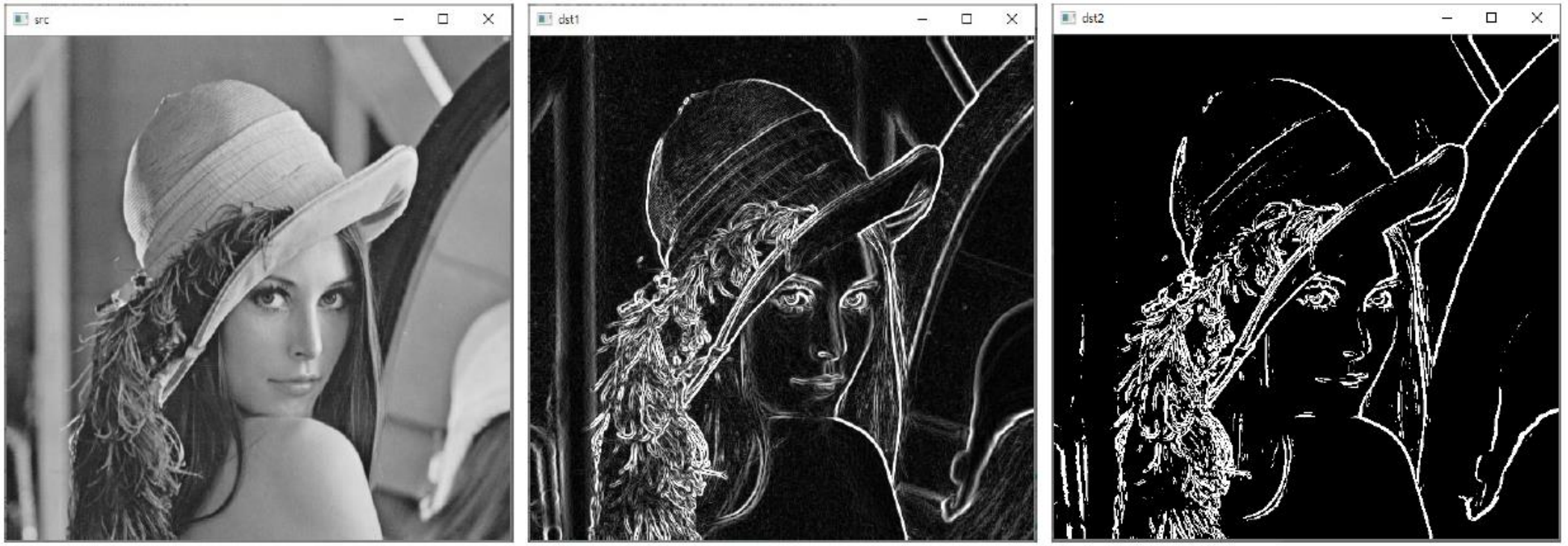
그래디언트와 에지 검출

소벨 필터를 이용한 에지 검출 예제

```python
import sys
import numpy as np
import cv2

src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

if src is None:
    print('Image load failed!')
    sys.exit()

dx = cv2.Sobel(src, cv2.CV_32F, dx: 1, dy: 0)
dy = cv2.Sobel(src, cv2.CV_32F, dx: 0, dy: 1)

mag = cv2.magnitude(dx, dy)
mag = np.clip(mag, a_min: 0, a_max: 255).astype(np.uint8)

dst = np.zeros(src.shape[:2], np.uint8)
dst[mag > 120] = 255
#_, dst = cv2.threshold(mag, 120, 255, cv2.THRESH_BINARY)

cv2.imshow( winname: 'src', src)
cv2.imshow( winname: 'mag', mag)
cv2.imshow( winname: 'dst', dst)
cv2.waitKey()

cv2.destroyAllWindows()
```

그래디언트와 에지 검출

소벨 필터를 이용한 에지 검출 예제 실행 결과



캐니 에지 검출

좋은 에지 검출기의 조건(J. Canny)

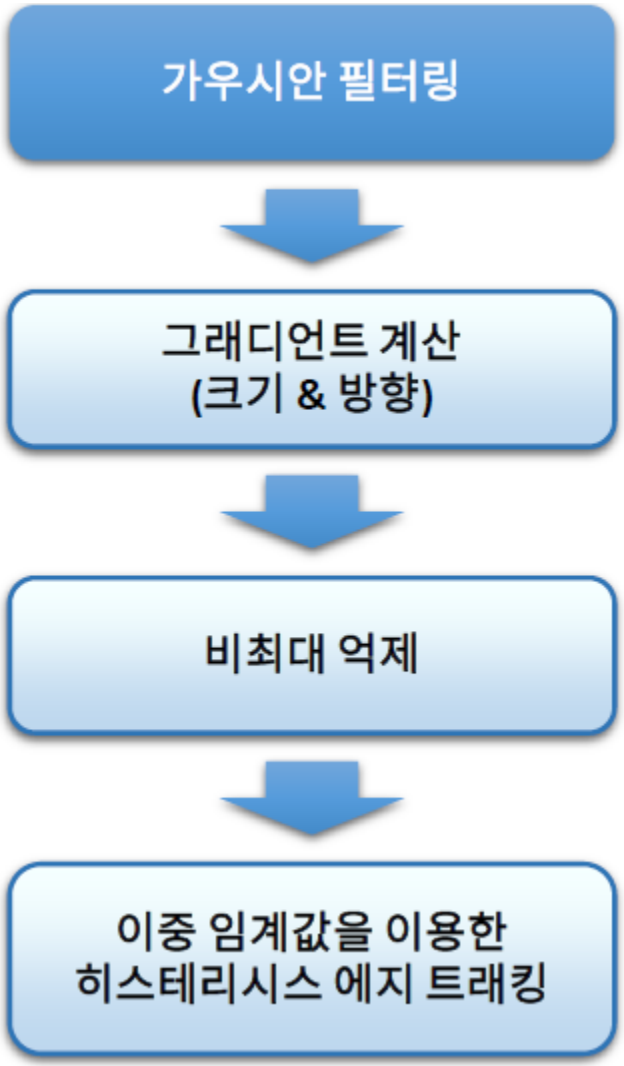
- 정확한 검출(Good detection): 에지가 아닌 점을 에지로 찾거나 또는 에지인데 에지로 찾지 못하는 확률을 최소화
- 정확한 위치(Good localization): 실제 에지의 중심을 검출
- 단일 에지(Single edge): 하나의 에지는 하나의 점으로 표현

캐니 에지 검출

캐니 에지 검출 1단계

- 가우시안 필터링
  - (Optional) 잡음 제거 목적

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$



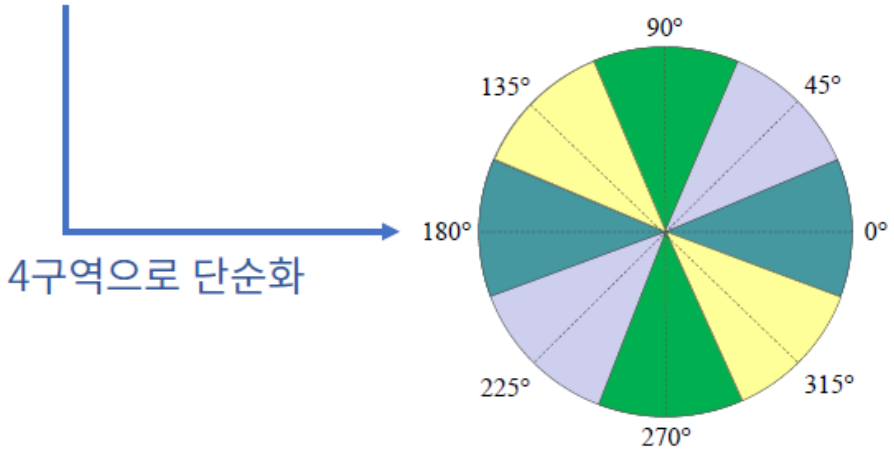
캐니 에지 검출

캐니 에지 검출 2단계

- 그래디언트 계산
- 주로 소벨 마스크를 사용

- 크기: $\|f\| = \sqrt{f_x^2 + f_y^2}$

- 방향: $\theta = \tan^{-1}\left(f_y / f_x\right)$



가우시안 필터링

그래디언트 계산
(크기 & 방향)

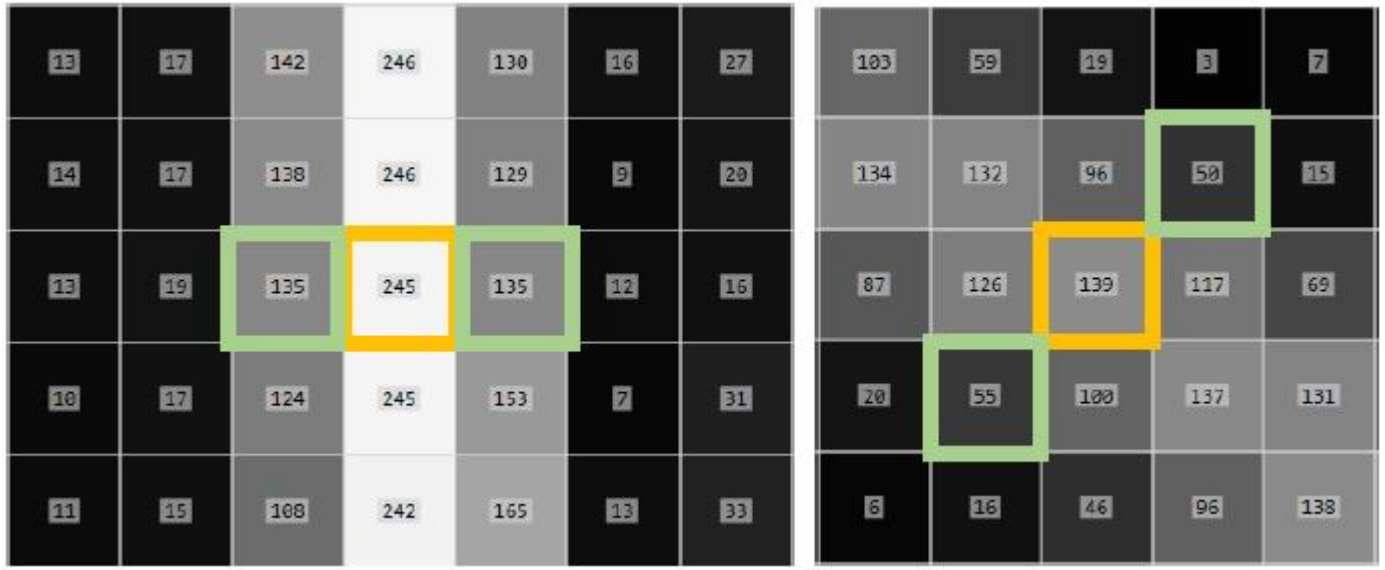
비최대 억제

이중 임계값을 이용한
히스테리시스 에지 트래킹

캐니 에지 검출

캐니 에지 검출 3단계

- 비최대 억제(Non-maximum suppression)
- 하나의 에지가 여러 개의 픽셀로 표현되는 현상을 없애기 위하여
  그래디언트 크기가 국지적 최대(local maximum)인 픽셀만을 에지 픽셀로 설정
- 그래디언트 방향에 위치한 두 개의 픽셀을 조사하여 국지적 최대를 검사



가우시안 필터링

그래디언트 계산
(크기 & 방향)

비최대 억제

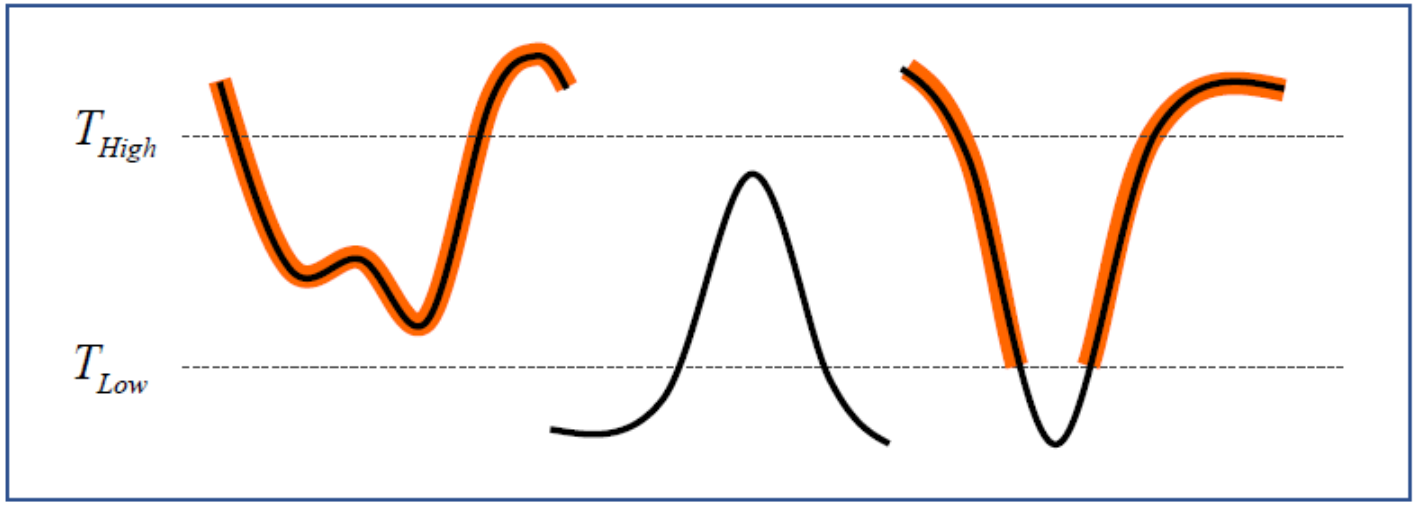
이중 임계값을 이용한
히스테리시스 에지 트래킹

캐니 에지 검출

캐니 에지 검출 4단계

- 히스테리시스 에지 트래킹(Hysteresis edge tracking)
  - 두 개의 임계값을 사용: T_{Low}, T_{High}
  - 강한 에지: $\|f\| \geq T_{High}$ → 최종 에지로 선정
  - 약한 에지: $T_{Low} \leq \|f\| < T_{High}$

    → 강한 에지와 연결되어 있는 픽셀만 최종 에지로 선정



가우시안 필터링

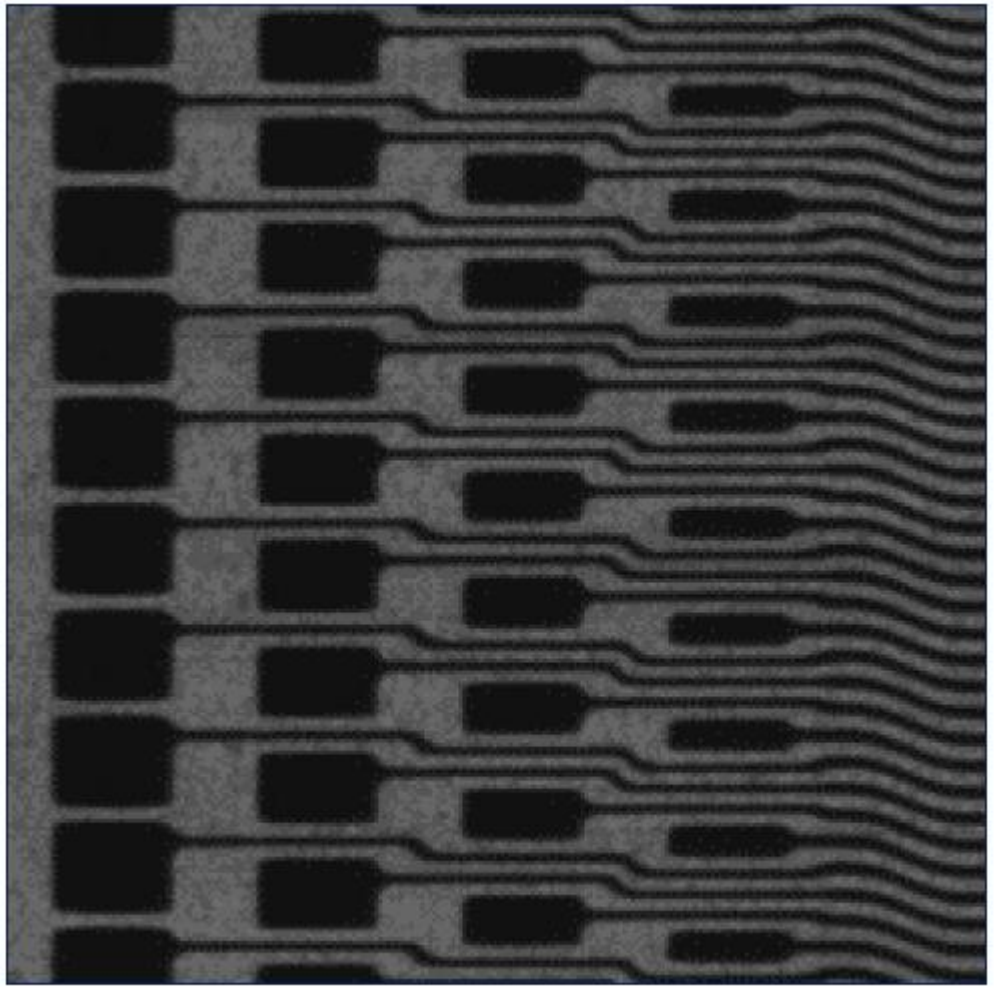
그래디언트 계산
(크기 & 방향)

비최대 억제

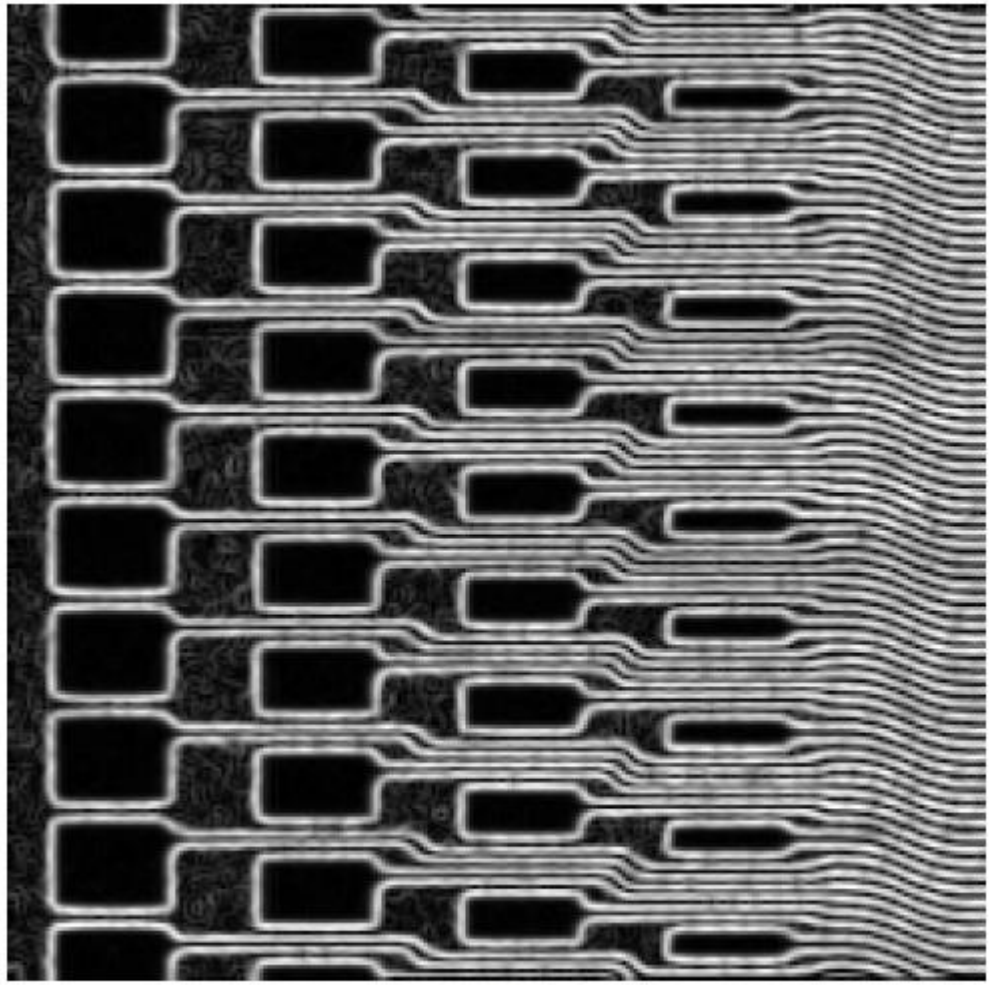
이중 임계값을 이용한
히스테리시스 에지 트래킹

캐니 에지 검출

캐니 에지 검출 과정



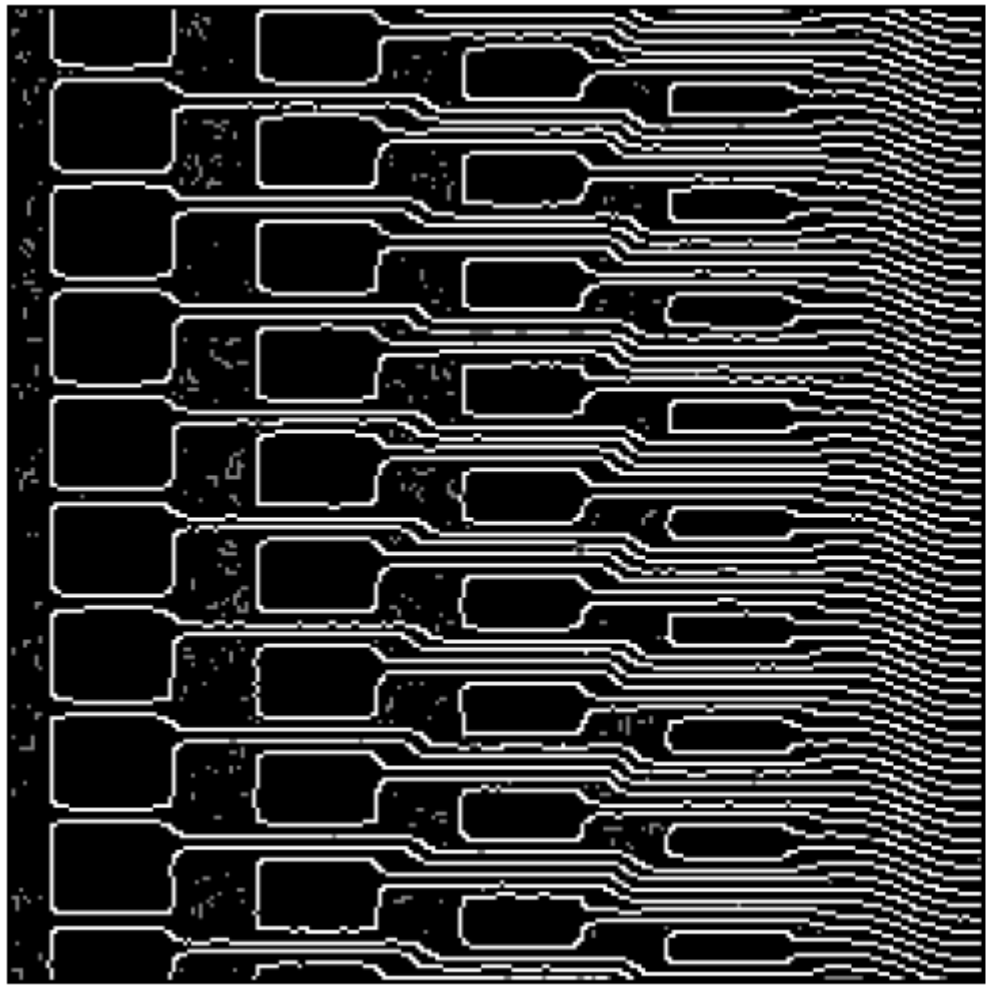
입력 영상



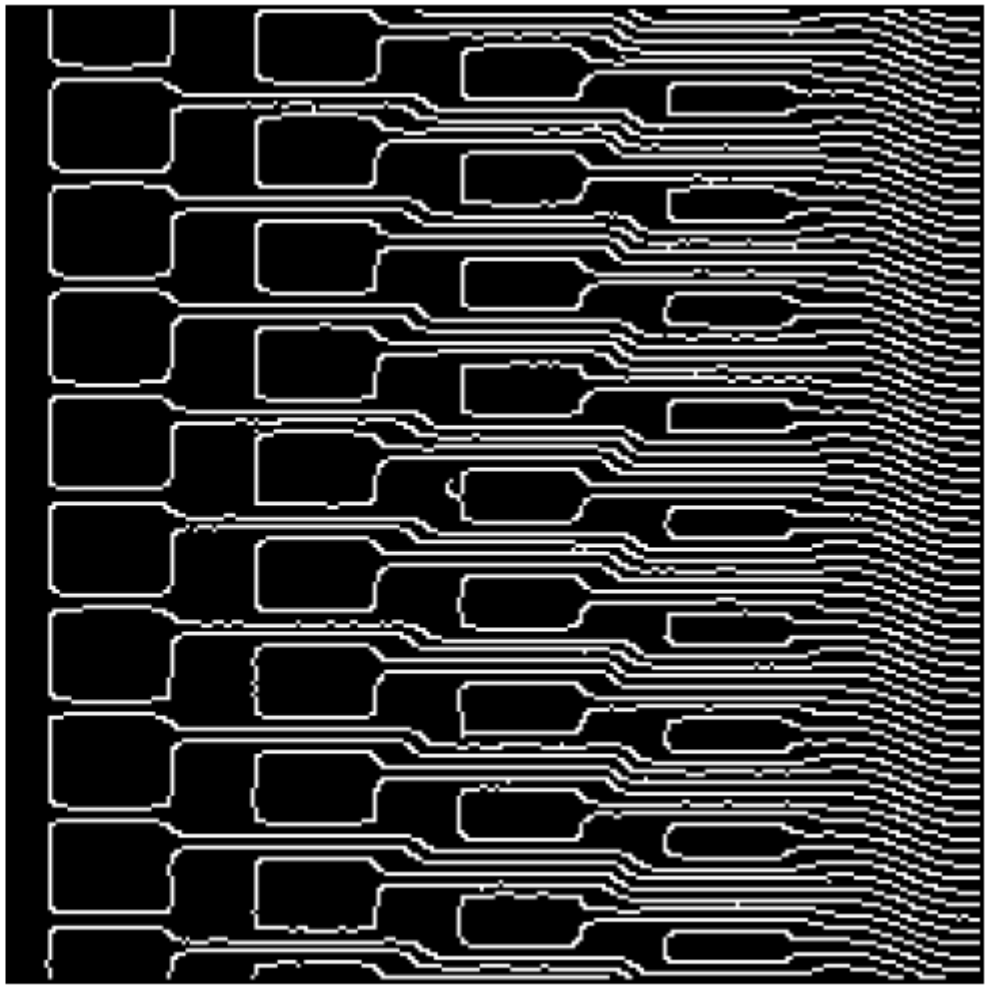
그래디언트 크기

캐니 에지 검출

캐니 에지 검출 과정



비최대 억제



히스테리시스 에지 트래킹

캐니 에지 검출

캐니 에지 검출 함수

cv2.Canny(image, threshold1, threshold2, edges=None, apertureSize=None, L2gradient=None) -> edges

- image: 입력 영상
- threshold1: 하단 임계값
- threshold2: 상단 임계값

} threshold1:threshold2 = 1:2 또는 1:3

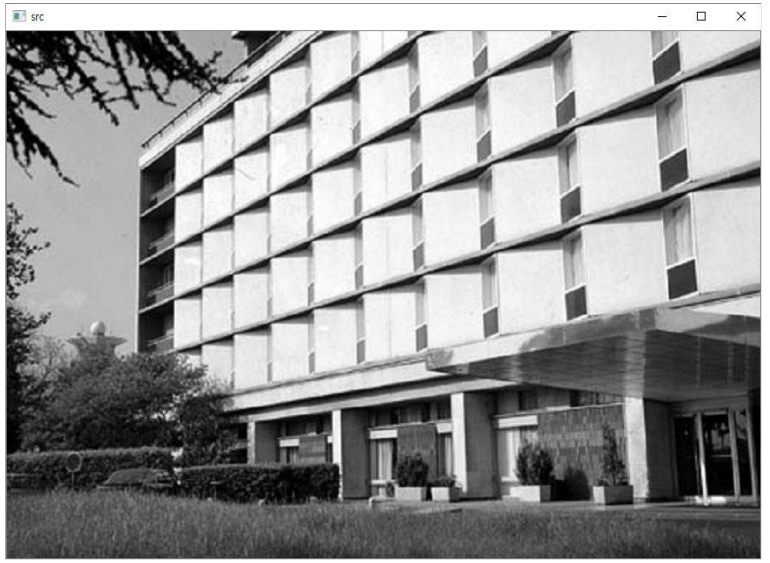
- edges: 에지 영상
- apertureSize: 소벨 연산을 위한 커널 크기. 기본값은 3.
- L2gradient: True이면 L2 norm 사용, False이면 L1 norm 사용. 기본값은 False.

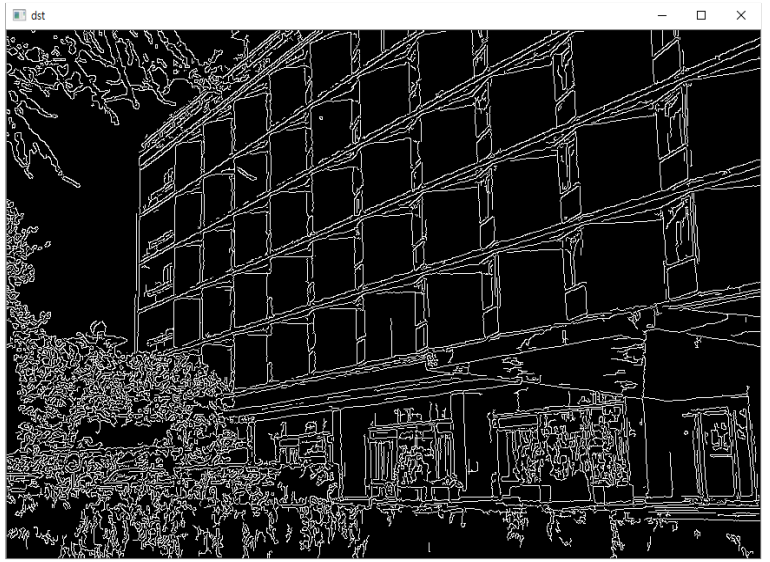
$$L_2 \text{ norm} = \sqrt{(dI/dx)^2 + (dI/dy)^2}\ ,\ L_1 \text{ norm} = |dI/dx| + |dI/dy|$$

캐니 에지 검출

캐니 에지 검출 예제

```python
import sys
import numpy as np
import cv2

src = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE)

if src is None:
    print('Image load failed!')
    sys.exit()

dst = cv2.Canny(src, 50, 150)

cv2.imshow( winname: 'src', src)
cv2.imshow( winname: 'dst', dst)
cv2.waitKey()

cv2.destroyAllWindows()
```



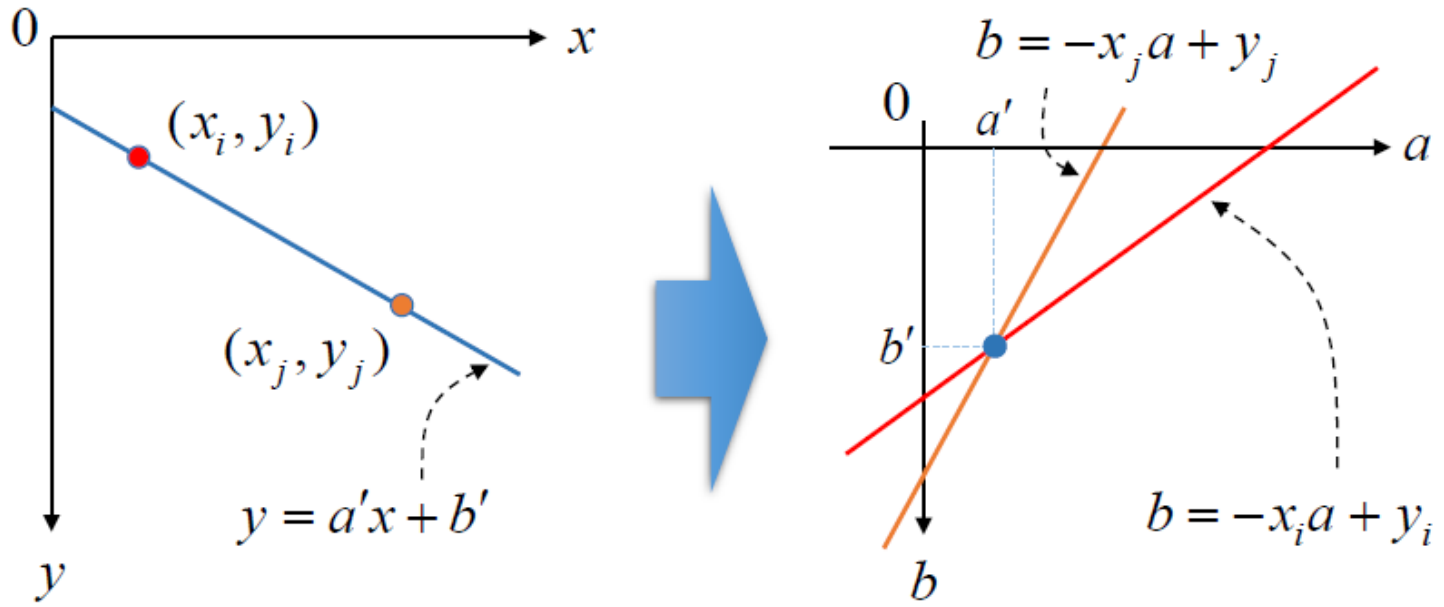


허프 변환: 직선 검출

허프 변환(Hough transform) 직선 검출이란?

- 2차원 영상 좌표에서의 직선의 방정식을 파라미터(parameter) 공간으로 변환하여 직선을 찾는 알고리즘

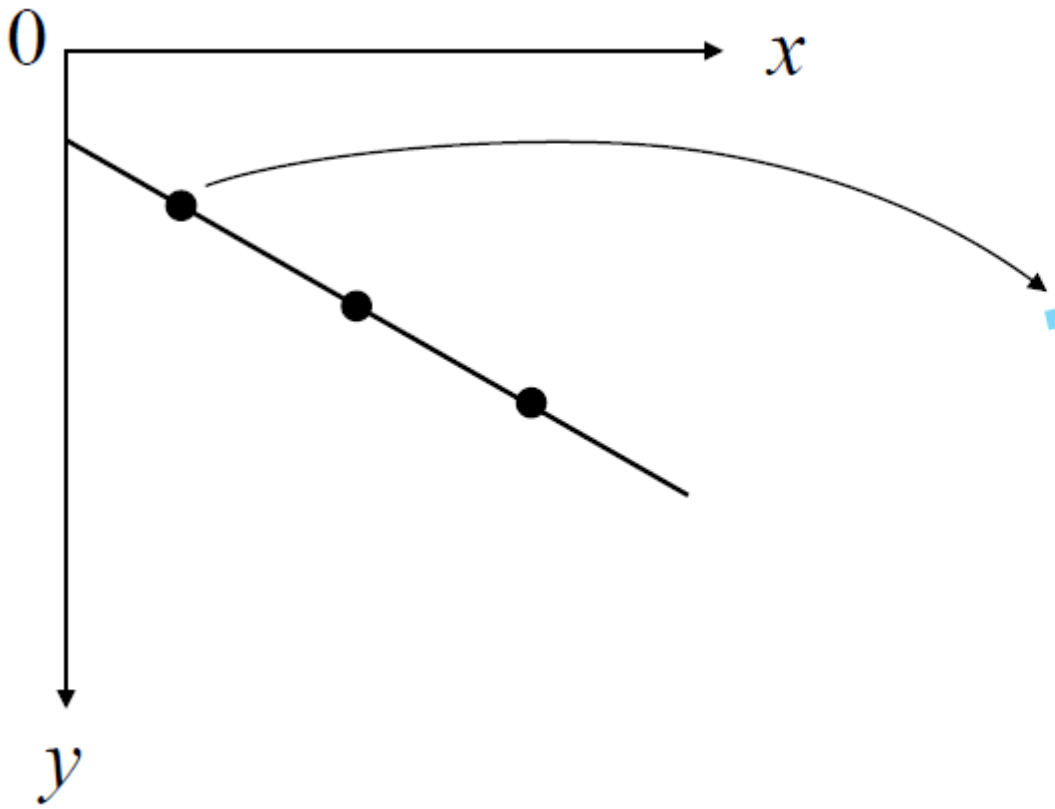
$$y = ax + b \quad \Leftrightarrow \quad b = -xa + y$$



허프 변환: 직선 검출

축적 배열(accumulation array)

- 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열

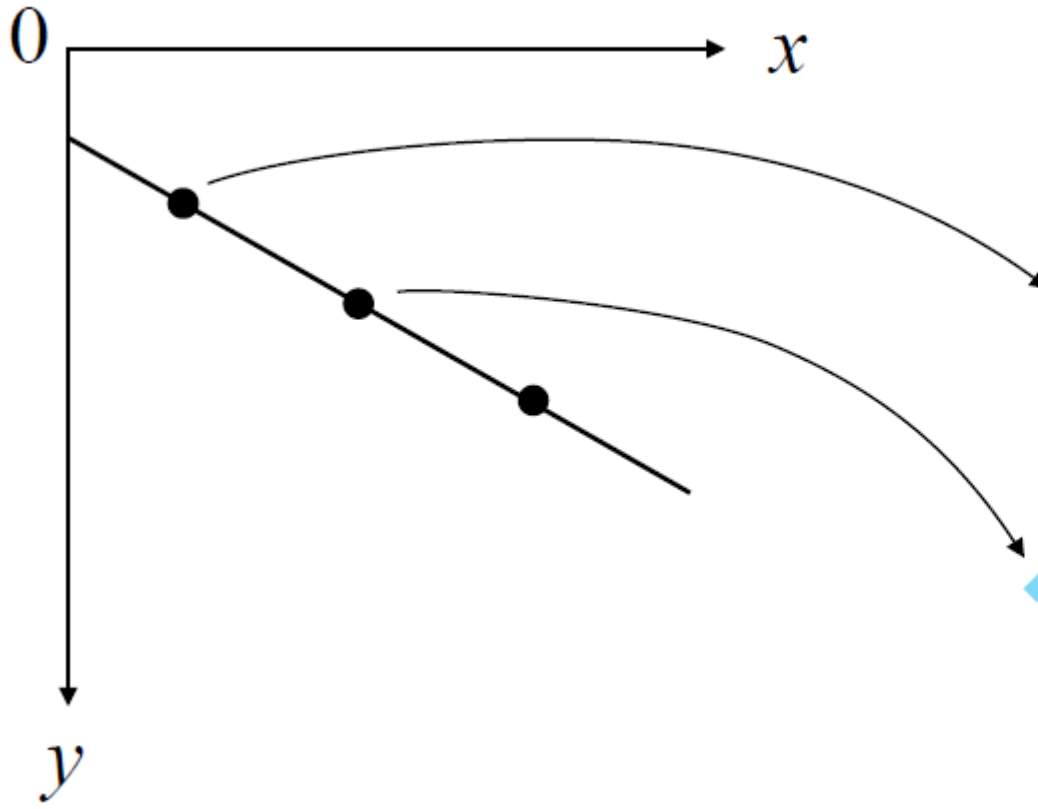


0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

허프 변환: 직선 검출

축적 배열(accumulation array)

- 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열

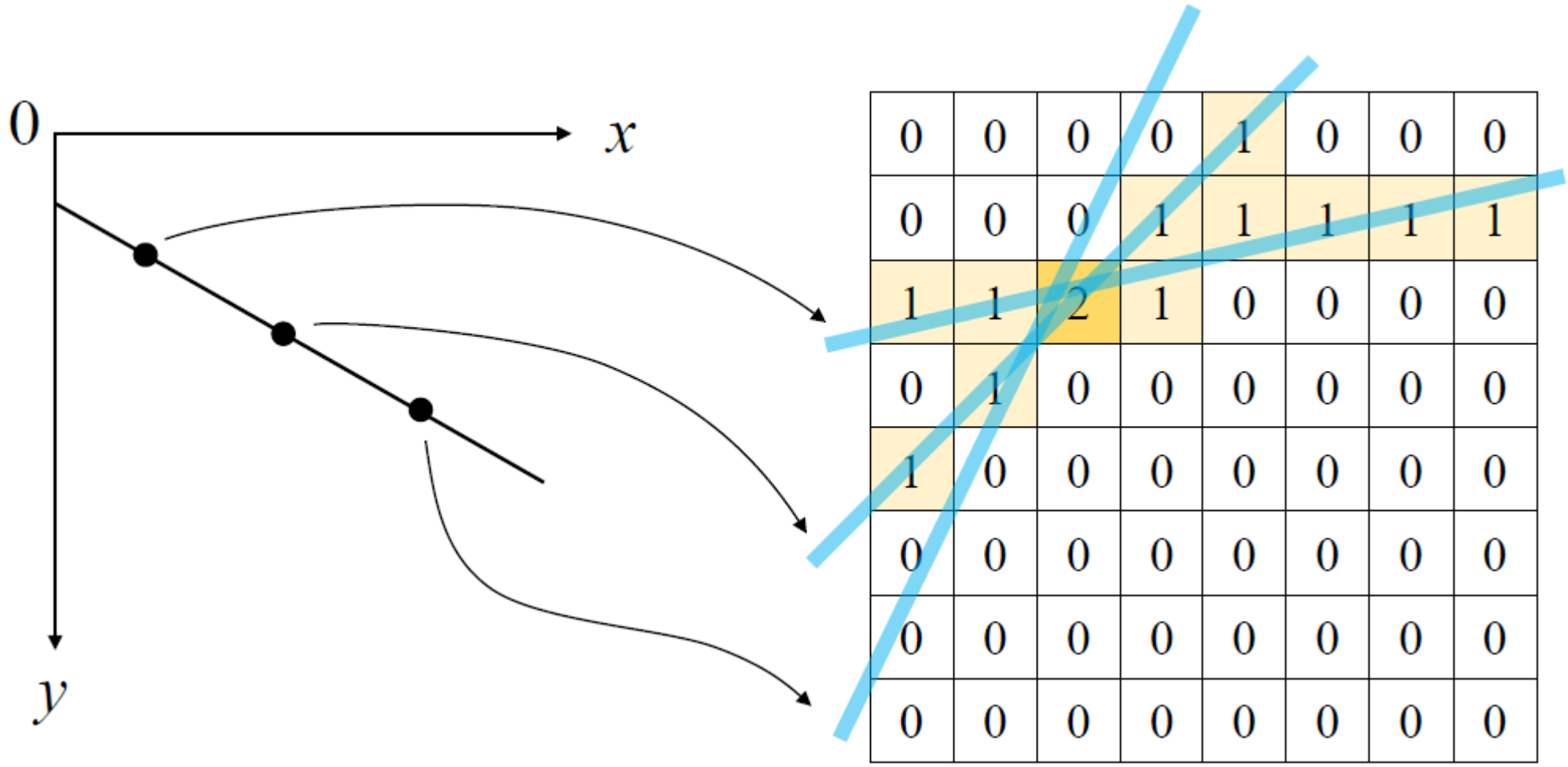


0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

허프 변환: 직선 검출

축적 배열(accumulation array)

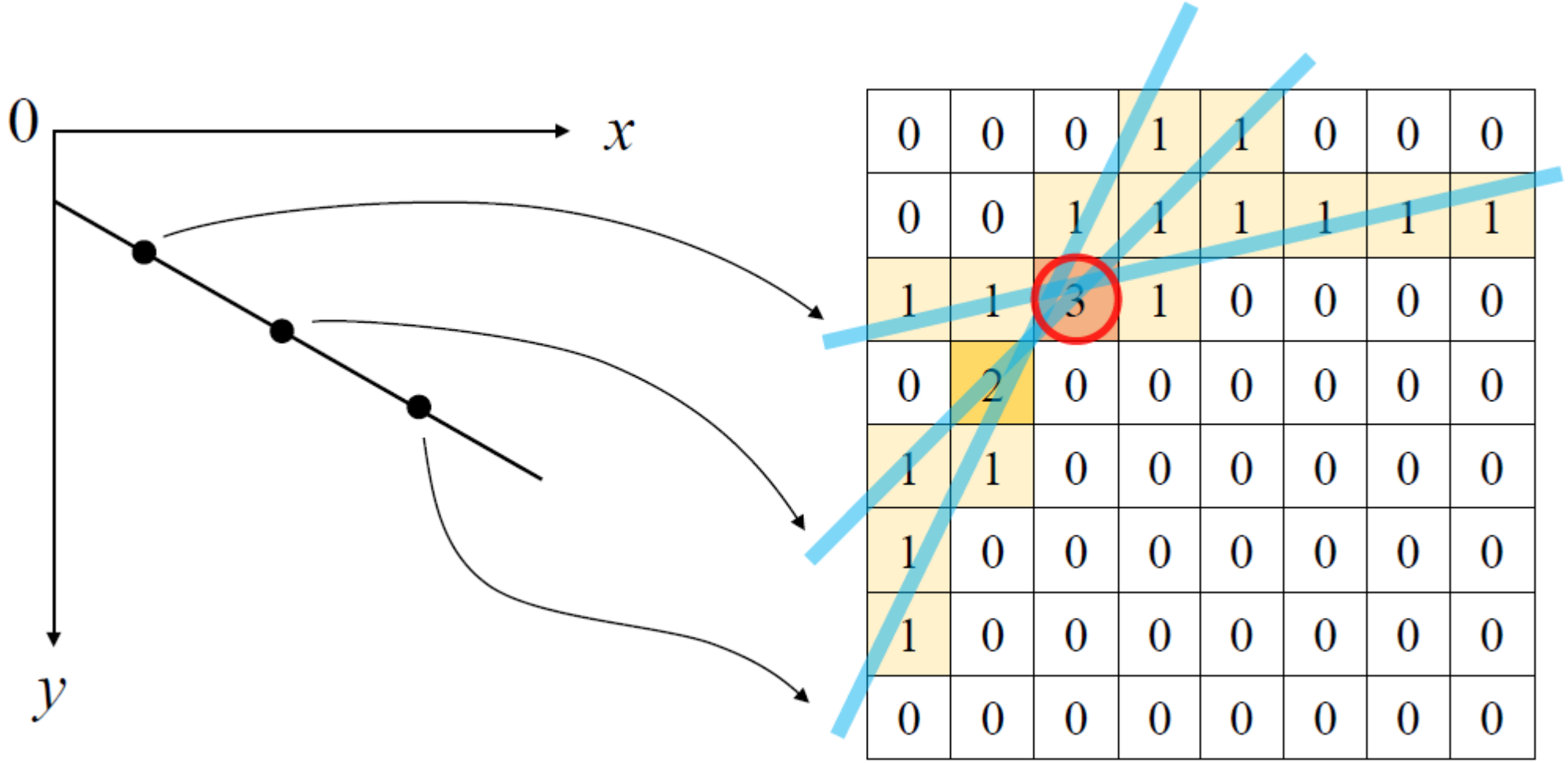
- 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



허프 변환: 직선 검출

축적 배열(accumulation array)

- 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열

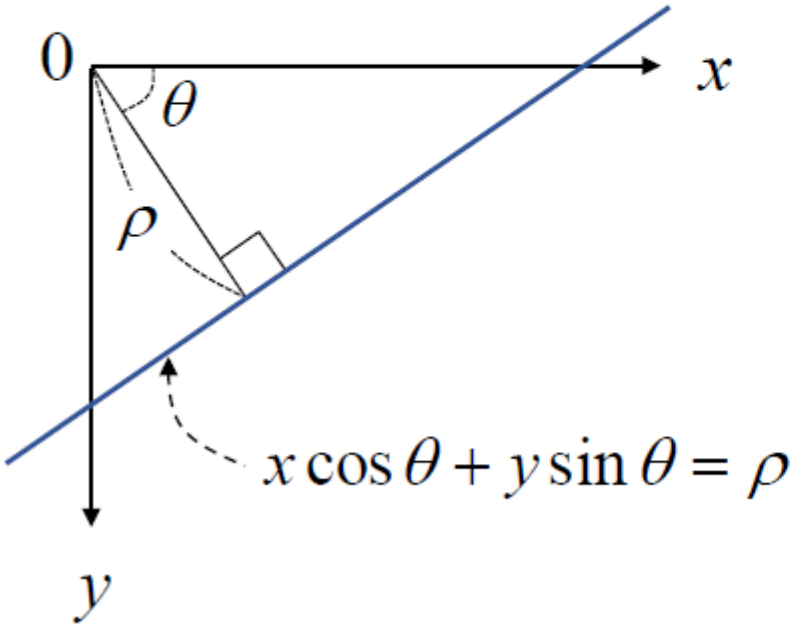


허프 변환: 직선 검출

직선의 방정식 y= ax+ b를 사용할 때의 문제점

- Y축과 평행한 수직선을 표현하지 못함 → 극좌표계 직선의 방정식을 사용

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



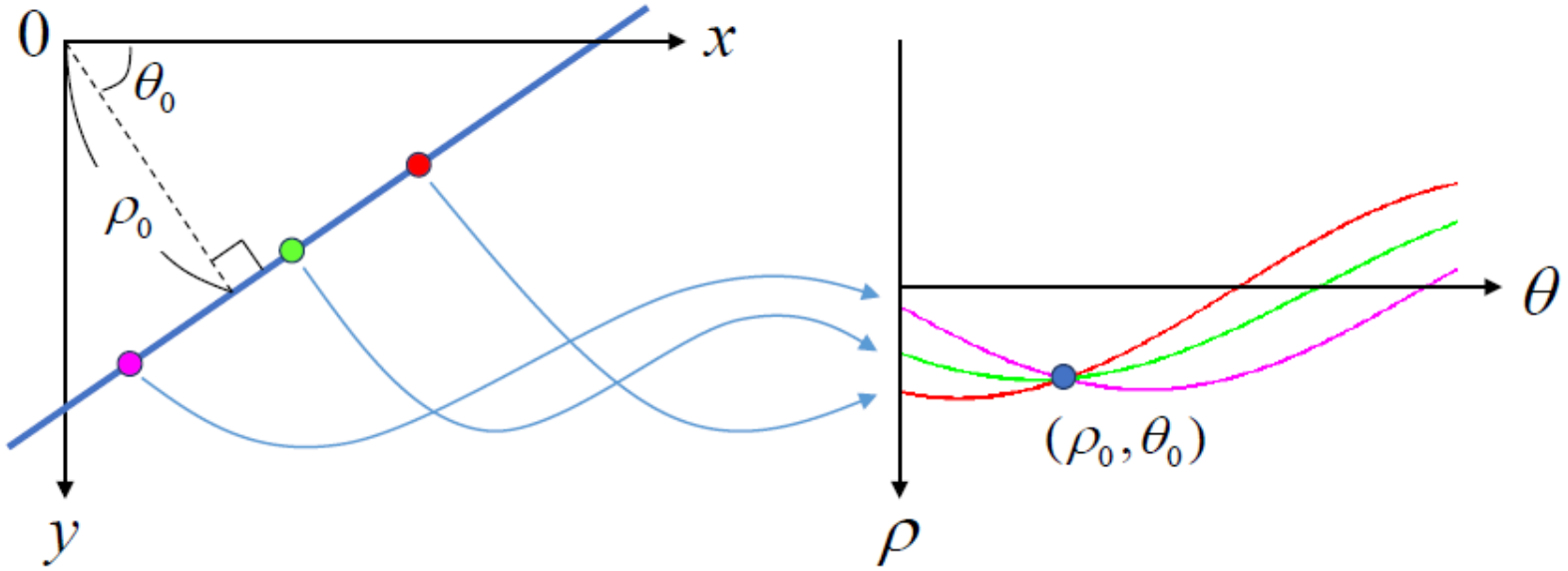
$$\begin{cases} \text{기울기} = -\dfrac{\cos\theta}{\sin\theta} \\ \text{y절편} = \dfrac{\rho}{\sin\theta} \end{cases}$$

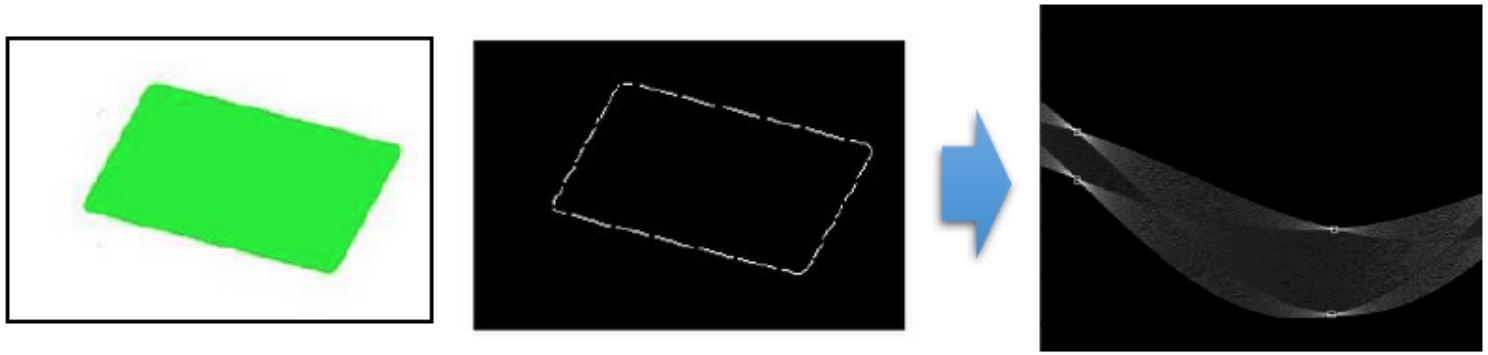
$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$

➔ $x\cos\theta + y\sin\theta = \rho$

허프 변환: 직선 검출

- **xcosθ+ ysinθ = ρ 방정식에 의한 파라미터 공간으로의 변환**





허프 변환: 직선 검출

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.) np.pi/180 → 1°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

허프 변환: 직선 검출

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.) np.pi/180 → 1°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

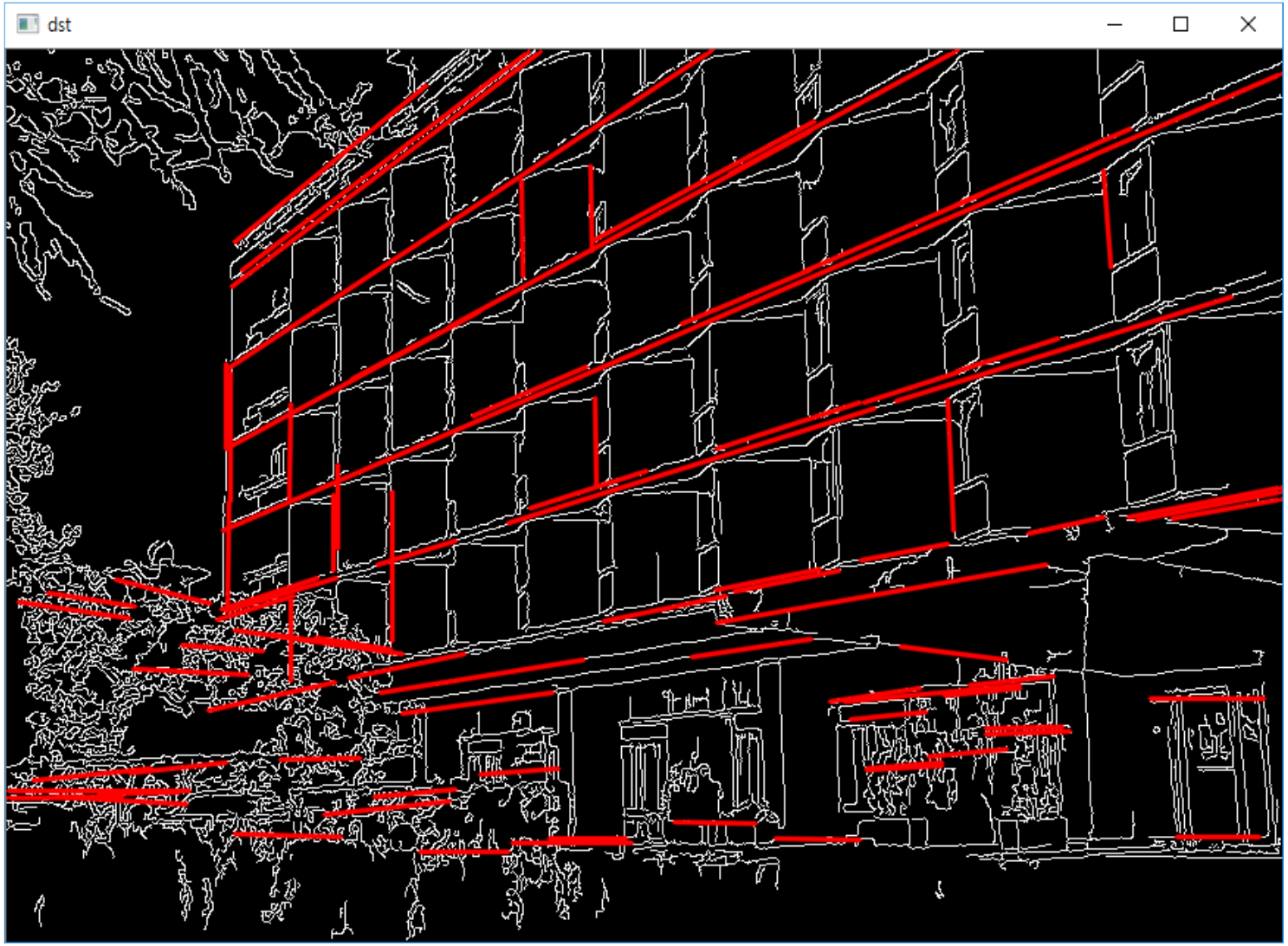
허프 변환: 직선 검출

확률적 허프 변환 직선 검출 예제

```python
import sys
import numpy as np
import cv2

src = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE)

if src is None:
    print('Image load failed!')
    sys.exit()

edges = cv2.Canny(src, 50, 150)

lines = cv2.HoughLinesP(edges, rho: 1, np.pi / 180., threshold: 160,
                        minLineLength=50, maxLineGap=5)

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

if lines is not None:
    for i in range(lines.shape[0]):
        pt1 = (lines[i][0][0], lines[i][0][1])  # 시작점 좌표
        pt2 = (lines[i][0][2], lines[i][0][3])  # 끝점 좌표
        cv2.line(dst, pt1, pt2, color: (0, 0, 255), thickness: 2, cv2.LINE_AA)

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

허프 변환: 직선 검출

확률적 허프 변환 직선 검출 예제 실행 결과



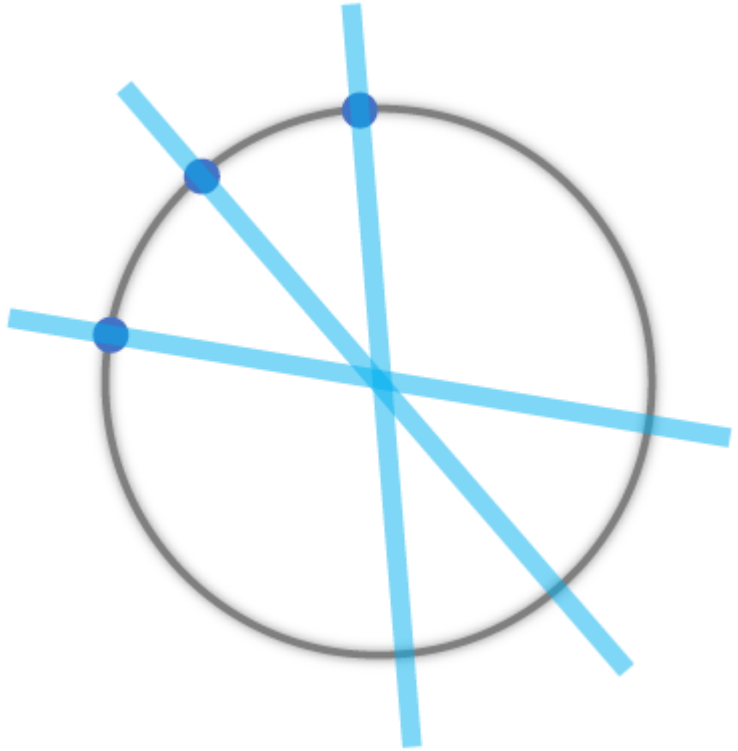
허프 변환: 원 검출

허프 변환을 응용하여 원을 검출할 수 있음

- 원의방정식: $(x - a)^2 + (y - b)^2 = c^2$ → 3차원 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 → 가장 작은 원 하나만 검출됨



허프 변환: 원 검출

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: OpenCV4.2 이하에서는 cv2.HOUGH_GRADIENT만 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

허프 변환: 원 검출

허프 변환 원 검출 예제

```python
import sys
import numpy as np
import cv2

# 입력 이미지 불러오기
src = cv2.imread('dial.jpg')

if src is None:
    print('Image open failed!')
    sys.exit()

gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
blr = cv2.GaussianBlur(gray, ksize: (0, 0), sigmaX: 1.0)

```

허프 변환: 원 검출

허프 변환 원 검출 예제

```
15      def on_trackbar(pos):
16          rmin = cv2.getTrackbarPos( trackbarname: 'minRadius',  winname: 'img')
17          rmax = cv2.getTrackbarPos( trackbarname: 'maxRadius',  winname: 'img')
18          th = cv2.getTrackbarPos( trackbarname: 'threshold',  winname: 'img')
19
20          circles = cv2.HoughCircles(blr, cv2.HOUGH_GRADIENT,  dp: 1,  minDist: 50,
21                                     param1=120, param2=th, minRadius=rmin, maxRadius=rmax)
22
23          dst = src.copy()
24          if circles is not None:
25              for i in range(circles.shape[1]):
26                  cx, cy, radius = np.uint16(circles[0][i])
27                  cv2.circle(dst,  center: (cx, cy), radius,  color: (0, 0, 255),  thickness: 2, cv2.LINE_AA)
28
29          cv2.imshow( winname: 'img', dst)
30
```

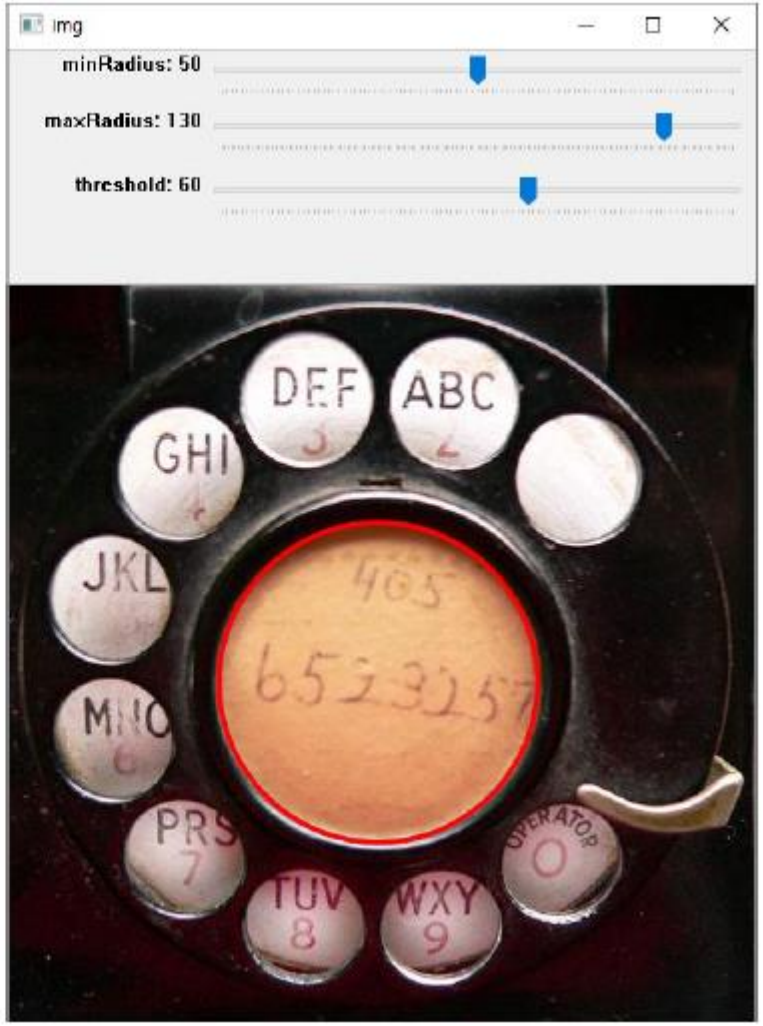
허프 변환: 원 검출

허프 변환 원 검출 예제

```
31      # 트랙바 생성
32      cv2.imshow( winname: 'img', src)
33      cv2.createTrackbar( trackbarName: 'minRadius', windowName: 'img', value: 0, count: 100, on_trackbar)
34      cv2.createTrackbar( trackbarName: 'maxRadius', windowName: 'img', value: 0, count: 150, on_trackbar)
35      cv2.createTrackbar( trackbarName: 'threshold', windowName: 'img', value: 0, count: 100, on_trackbar)
36      cv2.setTrackbarPos( trackbarname: 'minRadius', winname: 'img', pos: 10)
37      cv2.setTrackbarPos( trackbarname: 'maxRadius', winname: 'img', pos: 80)
38      cv2.setTrackbarPos( trackbarname: 'threshold', winname: 'img', pos: 40)
39      cv2.waitKey()
40
41      cv2.destroyAllWindows()
```

허프 변환: 원 검출

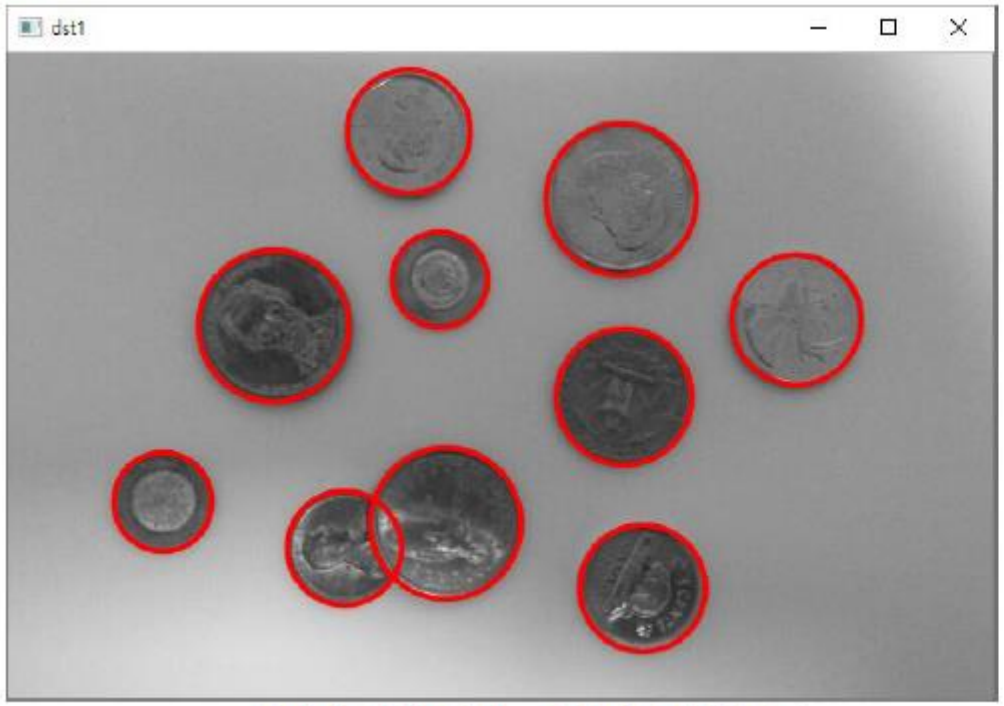
허프 변환 원 검출 예제 실행 결과



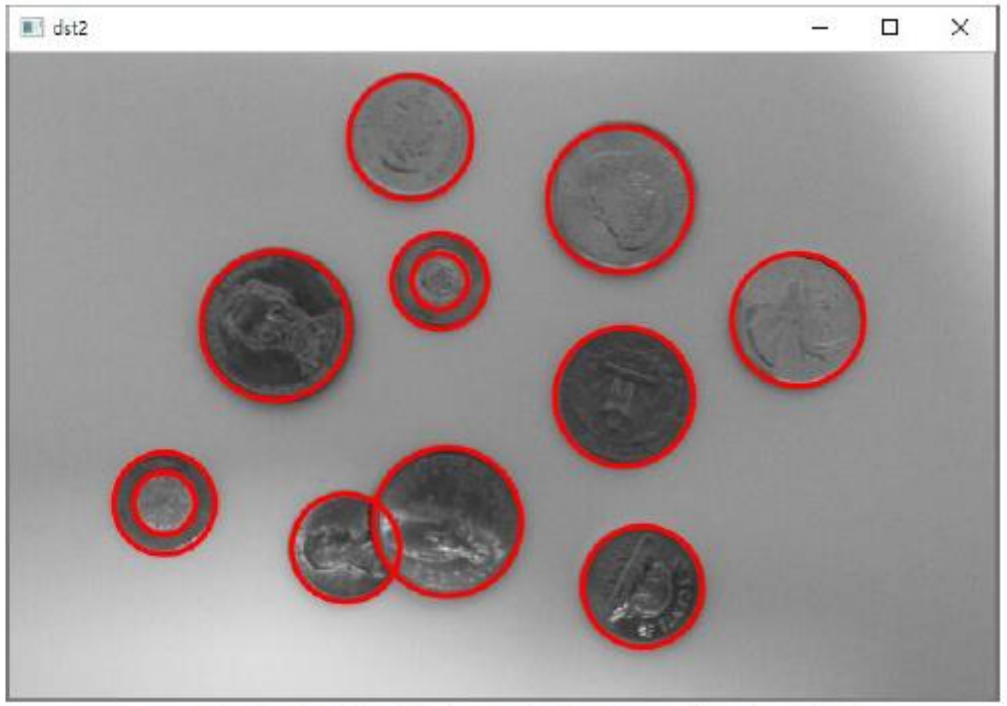
허프 변환: 원 검출

허프 원 검출과 cv2.HOUGH_GRADIENT_ALT 방법

- OpenCV 4.3버전부터 지원
- cv2.HOUGH_GRADIENT 방법보다 정확한 원 검출 가능
- 사용법: OpenCV 4.3.0 HoughCircles() 함수 설명 참고



cv2.HOUGH_GRADIENT



cv2.HOUGH_GRADIENT_ALT

[프로젝트] 동전 카운터

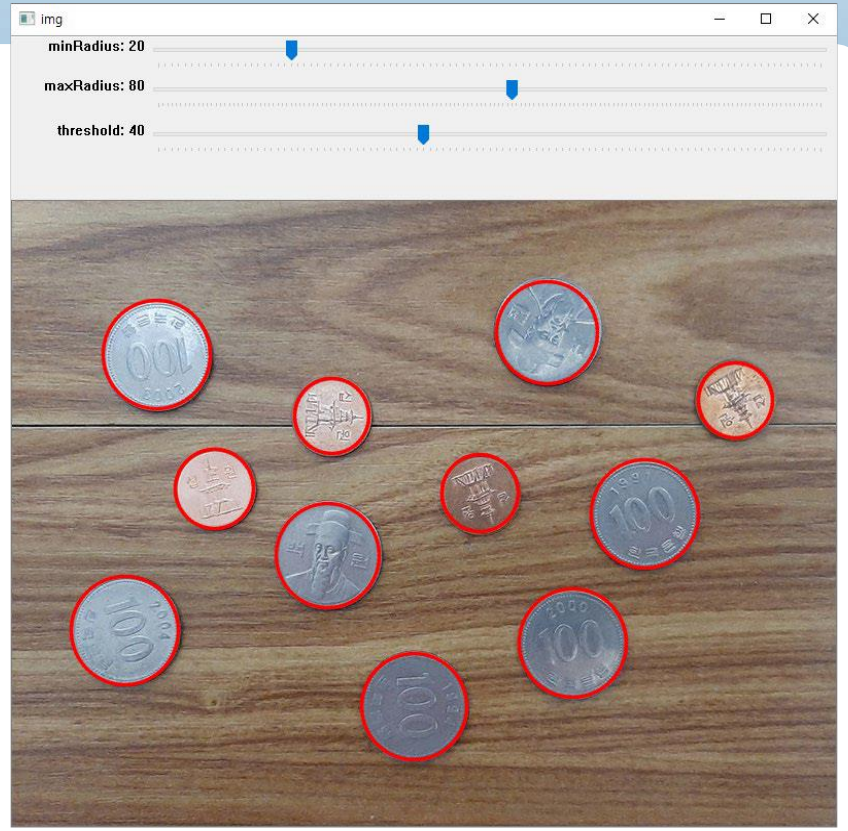
동전 카운터

- 영상의 동전을 검출하여 금액이 얼마인지를 자동으로 계산하는 프로그램
- 편의상 동전은 100원짜리와 10원짜리만 있다고 가정

구현할 기능

- 동전 검출하기 → 허프 원 검출
- 동전 구분하기 → 색상 정보 이용

[프로젝트] 동전 카운터



동전 검출하기

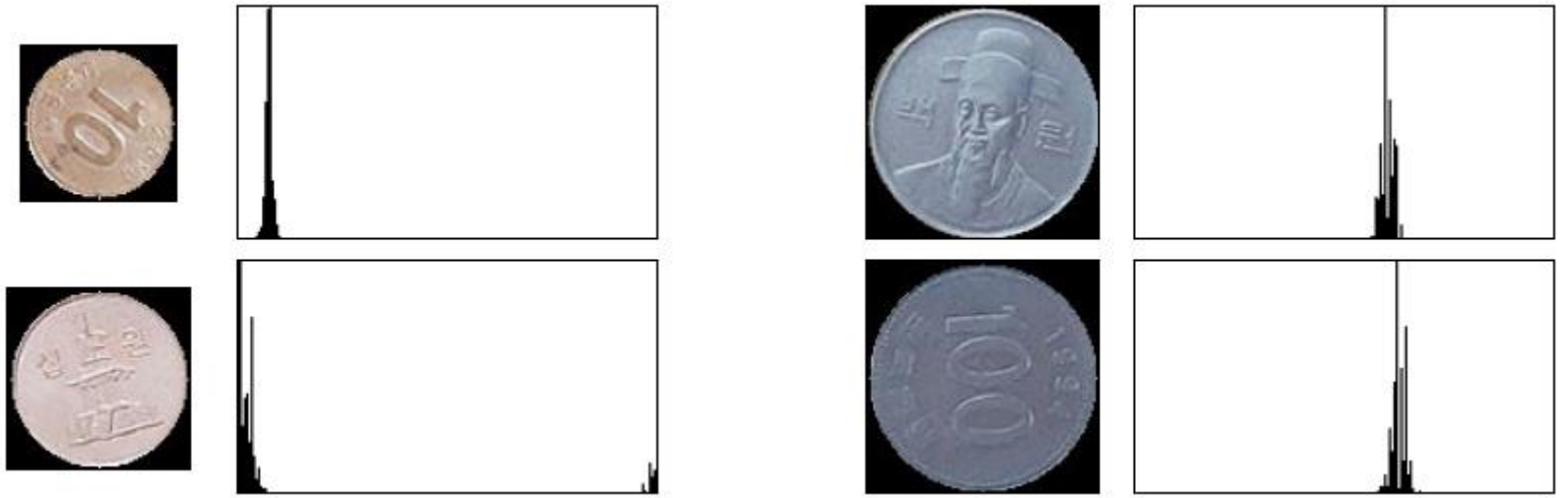
- 동그란 객체는 동전만 있다고 가정 → cv2.HoughCircles() 함수 사용
- 영상크기: 800x600 (px)
- 동전크기
  - 100원: 약100x100(px)
  - 10원: 약80x80(px)

```
src = cv2.imread('coins2.jpg')
gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
blr = cv2.GaussianBlur(gray, (0, 0), 1)

circles = cv2.HoughCircles(blr, cv2.HOUGH_GRADIENT, 1, 50,
                           param1=150, param2=40, minRadius=20, maxRadius=80)
```

[프로젝트] 동전 카운터

동전 구분하기

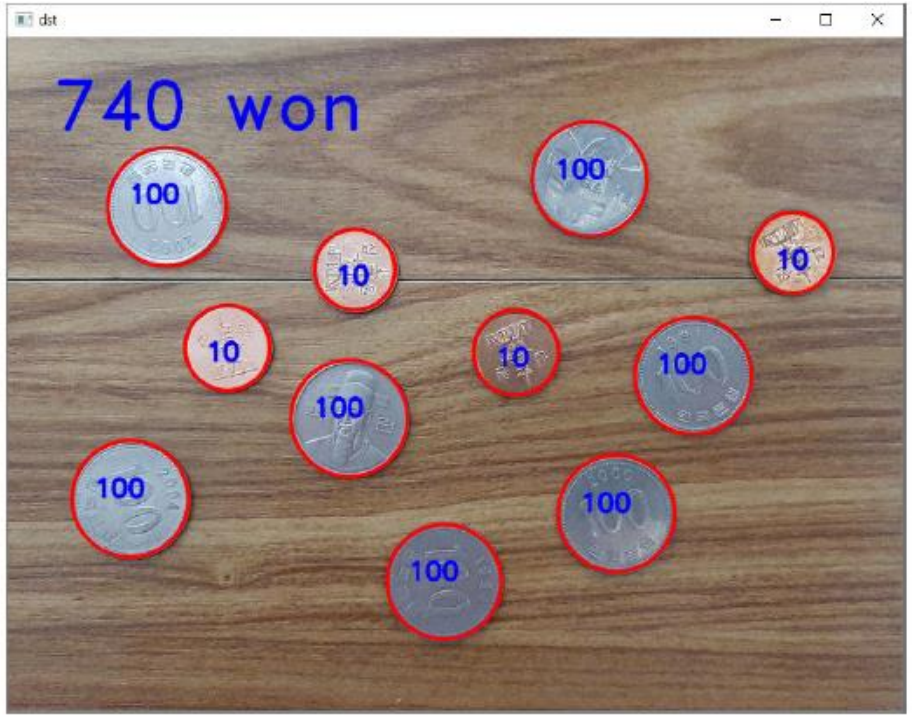
- 동전 영역 부분 영상 추출 → HSV 색 공간으로 변환
- 동전 영역에 대해서만 Hue 색 성분 분포 분석



- 동전 영역 픽셀에 대해 Hue 값을 +40만큼 시프트하고, Hue 평균을 분석
  - Hue 평균이 90보다 작으면 10원
  - Hue 평균이 90보다 크면 100원

[프로젝트] 동전 카운터

동전 구분하기



[프로젝트] 동전 카운터

동전 구분하기

```python
import sys
import numpy as np
import cv2

# 입력 이미지 불러오기
src = cv2.imread('coins1.jpg')

if src is None:
    print('Image open failed!')
    sys.exit()

gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
blr = cv2.GaussianBlur(gray, ksize: (0, 0), sigmaX: 1)

# 허프 변환 원 검출
circles = cv2.HoughCircles(blr, cv2.HOUGH_GRADIENT, dp: 1, minDist: 50,
                           param1=150, param2=40, minRadius=20, maxRadius=80)

```

[프로젝트] 동전 카운터

동전 구분하기

```
19  # 원 검출 결과 및 동전 금액 출력
20  sum_of_money = 0
21  dst = src.copy()
22  if circles is not None:
23      for i in range(circles.shape[1]):
24          cx, cy, radius = np.uint16(circles[0][i])
25          cv2.circle(dst, center: (cx, cy), radius, color: (0, 0, 255), thickness: 2, cv2.LINE_AA)
26
27          # 동전 영역 부분 영상 추출
28          x1 = int(cx - radius)
29          y1 = int(cy - radius)
30          x2 = int(cx + radius)
31          y2 = int(cy + radius)
32          radius = int(radius)
33
34          crop = dst[y1:y2, x1:x2, :]
35          ch, cw = crop.shape[:2]
36
37          # 동전 영역에 대한 ROI 마스크 영상 생성
38          mask = np.zeros( shape: (ch, cw), np.uint8)
39          cv2.circle(mask, center: (cw//2, ch//2), radius, color: 255, -1)
40
```

[프로젝트] 동전 카운터

동전 구분하기

```
41          # 동전 영역 Hue 색 성분을 +40 시프트하고, Hue 평균을 계산
42          hsv = cv2.cvtColor(crop, cv2.COLOR_BGR2HSV)
43          hue, _, _ = cv2.split(hsv)
44          hue_shift = (hue + 40) % 180
45          mean_of_hue = cv2.mean(hue_shift, mask)[0]
46
47          # Hue 평균이 90보다 작으면 10원, 90보다 크면 100원으로 간주
48          won = 100
49          if mean_of_hue < 90:
50              won = 10
51
52          sum_of_money += won
53
54          cv2.putText(crop, str(won), org: (20, 50), cv2.FONT_HERSHEY_SIMPLEX,
55                      fontScale: 0.75, color: (255, 0, 0), thickness: 2, cv2.LINE_AA)
56
57  cv2.putText(dst, str(sum_of_money) + ' won', org: (40, 80),
58              cv2.FONT_HERSHEY_DUPLEX, fontScale: 2, color: (255, 0, 0), thickness: 2, cv2.LINE_AA)
59
60  cv2.imshow( winname: 'src', src)
61  cv2.imshow( winname: 'dst', dst)
62  cv2.waitKey()
63
64  cv2.destroyAllWindows()
```