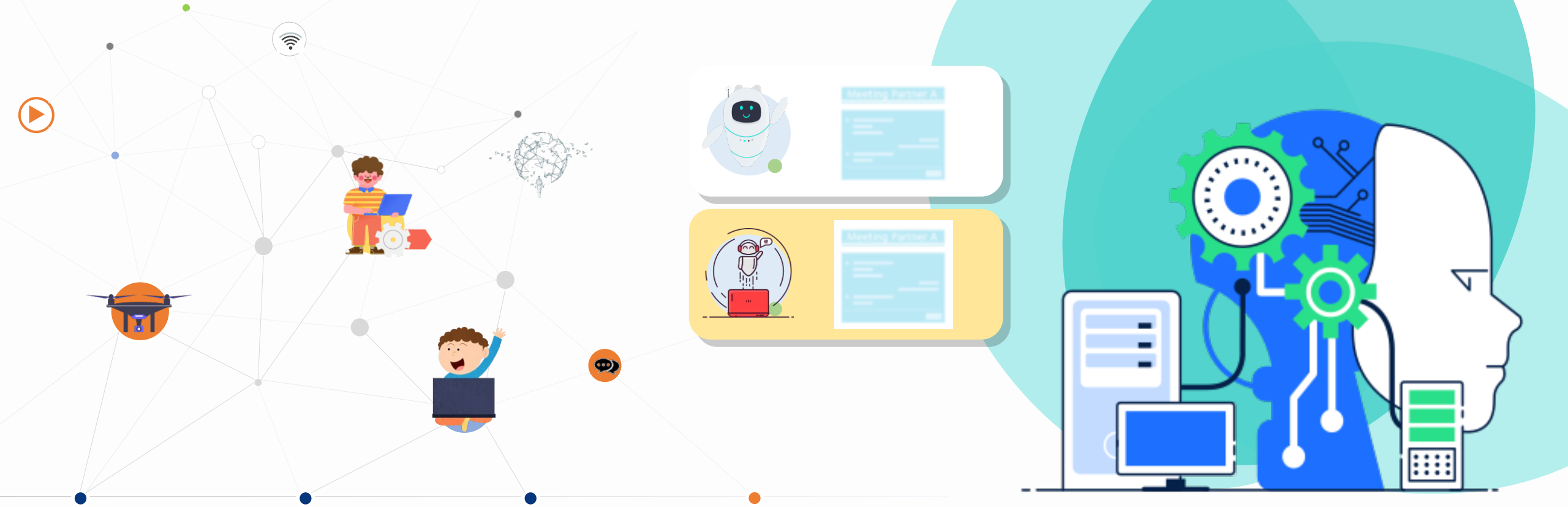
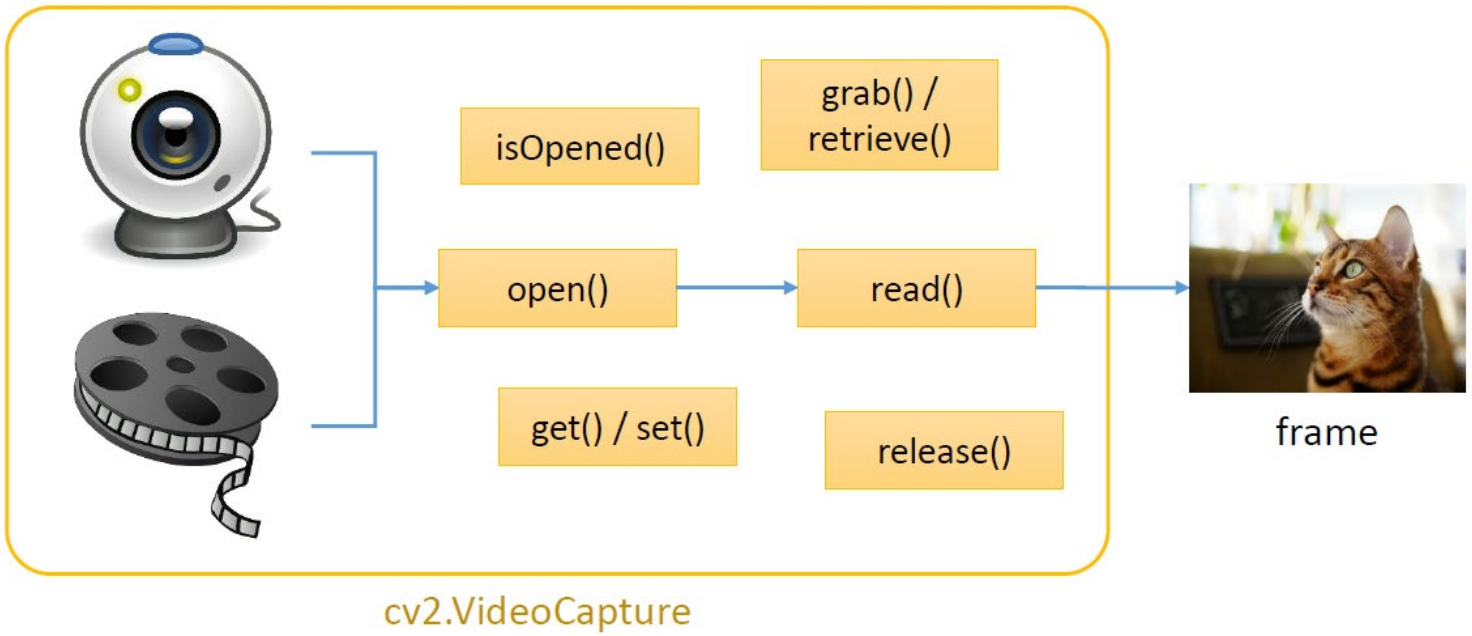


OpenCV-Python
특징 추출

멀티미디어

동영상 처리

- cv2.VideoCapture 클래스
  - OpenCV에서는 카메라와 동영상으로부터 프레임(frame)을 받아오는 작업을 cv2.VideoCapture 클래스 하나로 처리함



동영상 처리

- **카메라 열기**

```
cv2.VideoCapture(index, apiPreference=None) -> retval
```

- index: camera_id + domain_offset_id
  시스템 기본 카메라를 기본 방법으로 열려면 index에 0을 전달
- apiPreference: 선호하는 카메라 처리 방법을 지정
- retval: cv2.VideoCapture 객체

```
cv2.VideoCapture.open(index, apiPreference=None) -> retval
```

- retval: 성공하면 True, 실패하면 False.

동영상 처리

- **카메라 열기**
  - 비디오 캡쳐가 준비되었는지 확인

```
cv2.VideoCapture.isOpened() -> retval
```

  - retval: 성공하면 True, 실패하면 False.

  - 프레임 받아오기

```
cv2.VideoCapture.read(image=None) -> retval, image
```

  - retval: 성공하면 True, 실패하면 False.
  - image: 현재 프레임 (numpy.ndarray)

동영상 처리

- **카메라 열기**

```
cv2.VideoCapture.get(propId) -> retval
```

- propId: 속성 상수. (OpenCV 문서 참조)

CAP_PROP_FRAME_WIDTH	프레임 가로 크기
CAP_PROP_FRAME_HEIGHT	프레임 세로 크기
CAP_PROP_FPS	초당 프레임 수
CAP_PROP_FRAME_COUNT	비디오 파일의 총 프레임 수
CAP_PROP_POS_MSEC	밀리초 단위로 현재 위치
CAP_PROP_POS_FRAMES	현재 프레임 번호
CAP_PROP_EXPOSURE	노출

- retval: 성공하면 해당 속성 값, 실패하면 0.

동영상 처리

- 비디오 저장

- cv2.VideoWriter 클래스
  - OpenCV에서는 cv2.VideoWriter클래스를 이용하여 일련의 프레임을 동영상 파일로 저장할 수 있음
  - 일련의 프레임은 모두 크기와 데이터 타입이 같아야 함

- Fourcc
  - 동영상 파일의 코덱, 압축 방식, 색상, 픽셀 포맷 등을 정의하는 정수 값

cv2.VideoWriter_fourcc(*'DIVX')	DIVX MPEG-4 코덱
cv2.VideoWriter_fourcc(*'XVID')	XVID MPEG-4 코덱
cv2.VideoWriter_fourcc(*'FMP4')	FFMPEG MPEG-4 코덱
cv2.VideoWriter_fourcc(*'X264')	H.264/AVC 코덱
cv2.VideoWriter_fourcc(*'MJPG')	Motion-JPEG 코덱

동영상 처리

- **비디오 저장**
- **저장을 위한 동영상 파일 열기**

```
cv2.VideoWriter(filename, fourcc, fps, frameSize, isColor=None) -> retval
```

- filename: 비디오 파일 이름 (e.g. 'video.mp4')
- fourcc: fourcc (e.g. cv2.VideoWriter_fourcc(*'DIVX'))
- fps: 초당 프레임 수 (e.g. 30)
- frameSize: 프레임 크기. (width, height) 튜플.
- isColor: 컬러 영상이면 True, 그렇지않으면 False.
- retval: cv2.VideoWriter 객체

```
cv2.VideoWriter.open(filename, fourcc, fps, frameSize, isColor=None) -> retval
```

- retval: 성공하면 True, 실패하면 False.

동영상 처리

- **비디오 저장**
- **비디오 파일이 준비되었는지 확인**

```
cv2.VideoWriter.isOpened() -> retval
```

  - retval: 성공하면 True, 실패하면 False.

- **프레임 저장**

```
cv2.VideoWriter.write(image) -> None
```

  - image: 저장할 프레임 (numpy.ndarray)

트랙바 사용하기

- 트랙바란?
  - 프로그램 동작 중 사용자가 지정한 범위 안의 값을 선택할 수 있는 컨트롤 바
  - OpenCV에서 제공하는 그래픽 사용자 인터페이스

트랙바 사용하기

- **트랙바 생성 함수**

```
cv2.createTrackbar(trackbarName, windowName, value, count, onChange) -> None
```

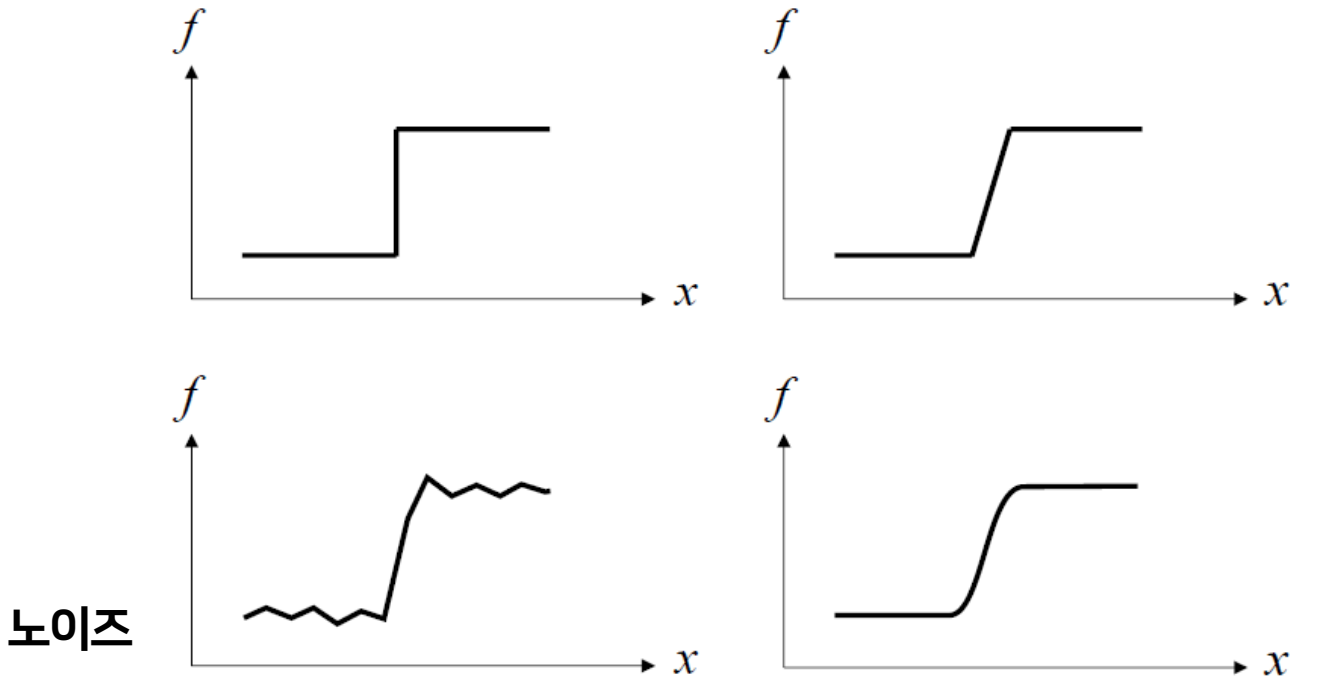
- trackbarName: 트랙바 이름
- windowName: 트랙바를 생성할 창 이름.
- value: 트랙바 위치 초기값
- count: 트랙바 최댓값. 최솟값은 항상 0.
- onChange: 트랙바 위치가 변경될 때마다 호출할 콜백 함수 이름

트랙바 이벤트 콜백 함수는 다음 형식을 따름.

```
onChange(pos) -> None
```

엣지 검출과 미분

- 엣지(edge)
  - 영상에서 픽셀의 값이 급격하게 변하는 부분
  - 일반적으로 배경과 객체, 또는 객체와 객체의 경계

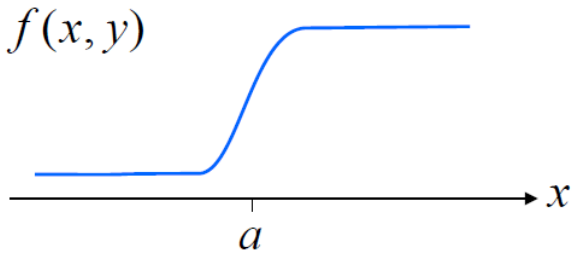


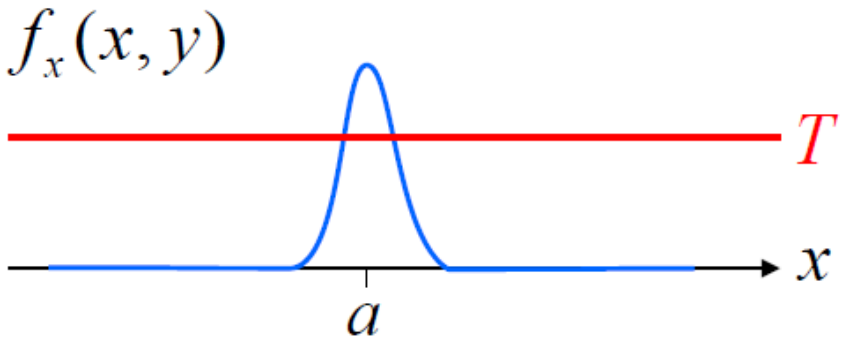
노이즈

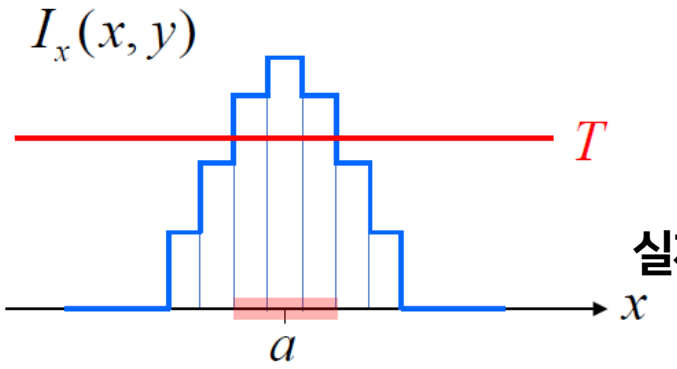
블러

엣지 검출과 미분

- 기본적인 엣지 검출 방법
  - 이미지를 (x,y)변수의 함수로 간주했을 때, 이 함수의 1차 미분 값이 크게 나타나는 부분을 검출







실제이미지

엣지 검출과 소벨 필터

- **1차 미분의 근사화**

$$f' = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x = 1\text{pix}}$$

- 전진 차분 (Forward difference):

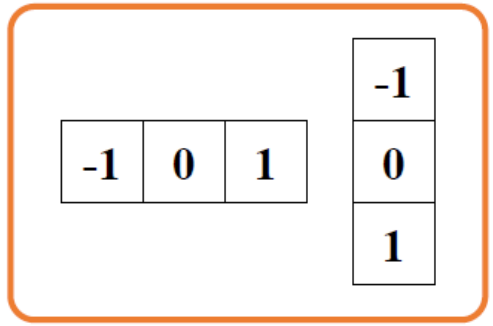
$$\frac{\partial I}{\partial x} \cong \frac{I(x+h) - I(x)}{h}$$

- 후진 차분 (Backward difference):

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

- 중앙 차분 (Centered difference):

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



미분 마스크

엣지 검출과 소벨 필터

- **다양한 미분 마스크**

	평균			가중치2배			3:10 가중치		
가로 방향:	-1	0	1	-1	0	1	-3	0	3
	-1	0	1	-2	0	2	-10	0	10
	-1	0	1	-1	0	1	-3	0	3
세로 방향:	-1	-1	-1	-1	-2	-1	-3	-10	-3
	0	0	0	0	0	0	0	0	0
	1	1	1	1	2	1	3	10	3
	Prewitt			Sobel			Scharr		

엣지 검출과 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.

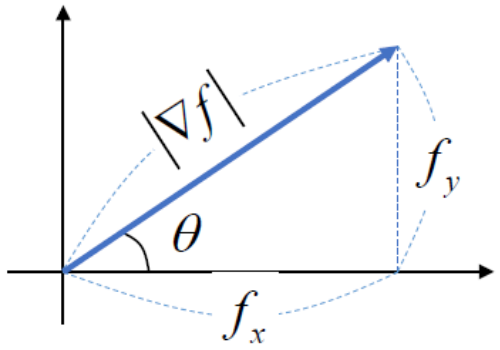
대부분 dx=1, dy=0, ksize=3 또는
dx=0, dy=1, ksize=3 으로 지정.

그래디언트

- 이미지의 그래디언트(gradient)
  - 함수 f(x,y)를 x축과 y축으로 미분하여 벡터형태로 표현한 것

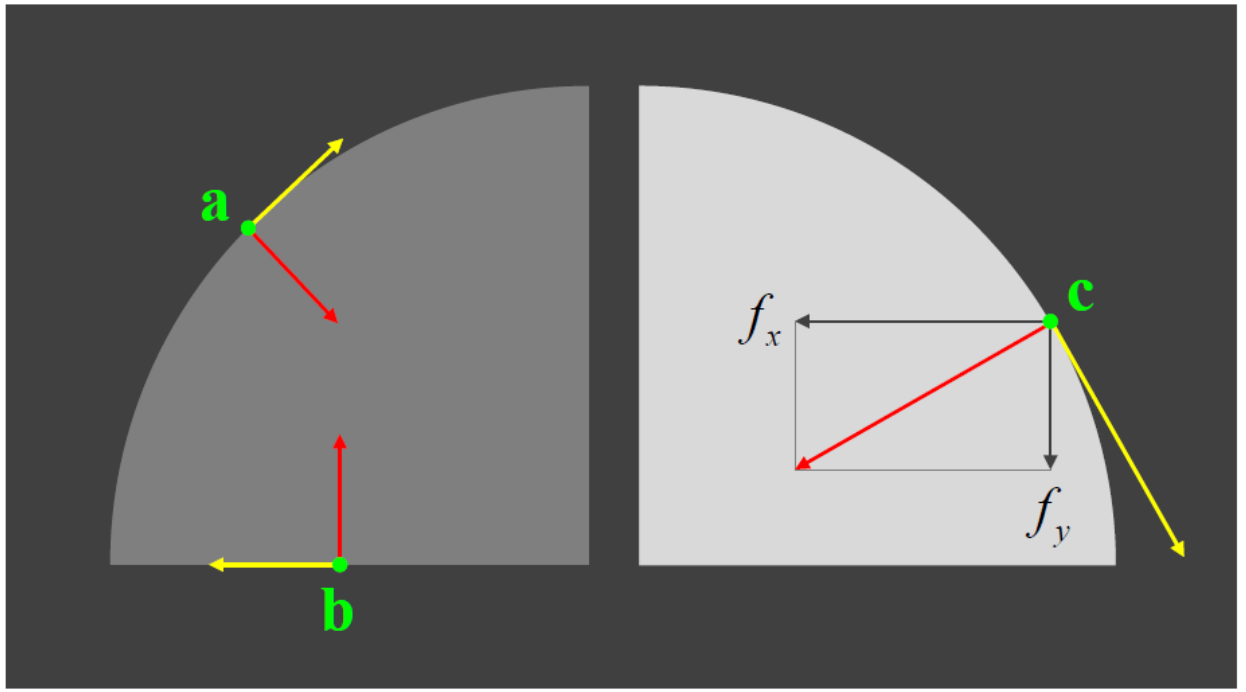
$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = f_x \mathbf{i} + f_y \mathbf{j}$$

- 그래디언트 크기: $|\nabla f| = \sqrt{f_x^2 + f_y^2}$
- 그래디언트 방향: $\theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$



그래디언트

- 실제 이미지에서 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 벡터의 크기 행렬. x와 같은 크기, 같은 타입.
  $\text{angle}(I) = \text{atan2}(\text{y}(I), \text{x}(I))$
  만약 x(I)=y(I)=0이면 angle은 0으로 설정됨.
- angleInDegrees: True이면 각도 단위, False이면 래디언 단위.

캐니 엣지 검출

- 좋은 엣지 검출기의 조건(J.Canny)

- 정확한 검출 : 엣지가 아닌 점을 엣지로 찾거나 또는 엣지인데 엣지로 찾지 못하는 확률을 최소화

- 정확한 위치 : 실제 엣지의 중심을 검출

- 단일 엣지 : 하나의 엣지는 하나의 점으로 표현

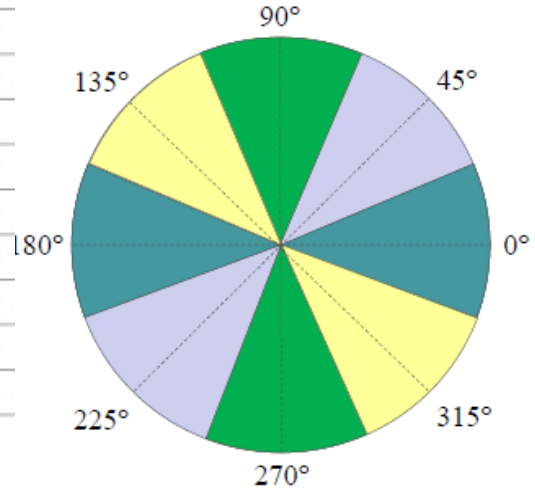
캐니 엣지 검출

- 캐니 엣지 검출 단계

1. 가우시안 필터링
   - 잡음제거목적

2. 그래디언트 계산
   - 주로 소벨 마스크를 사용

- 크기: $\|f\| = \sqrt{f_x^2 + f_y^2}$
- 방향: $\theta = \tan^{-1}\left(f_y / f_x\right)$

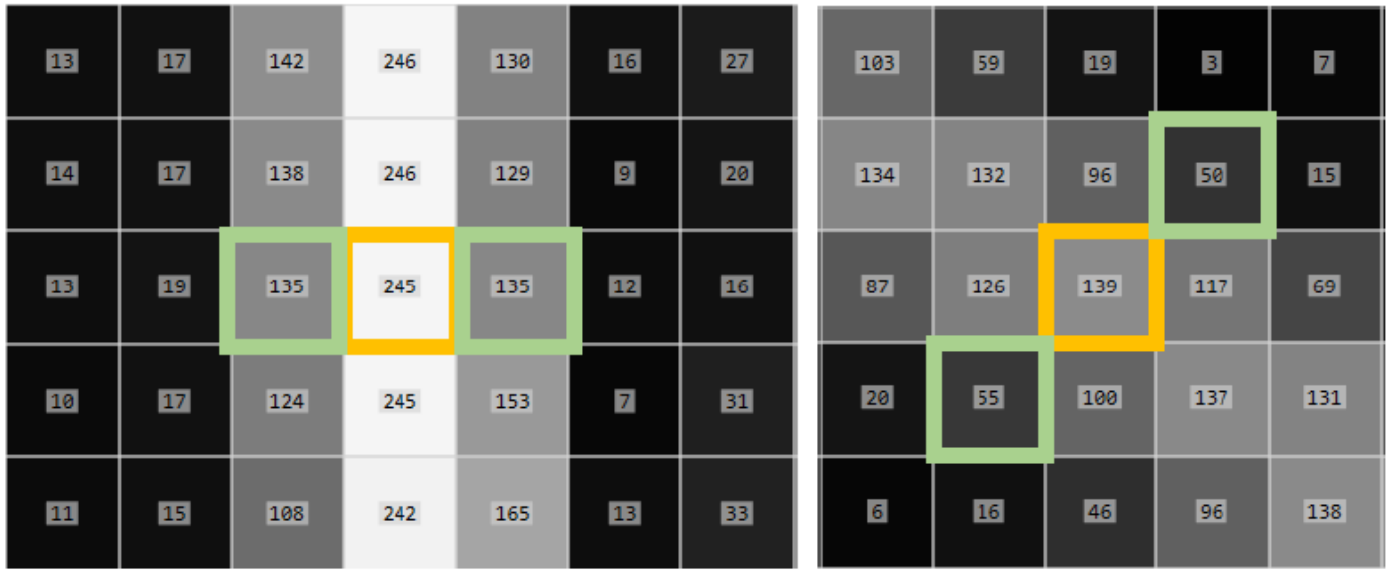


캐니 엣지 검출

- 캐니 엣지 검출 단계

3. 최대비 억제

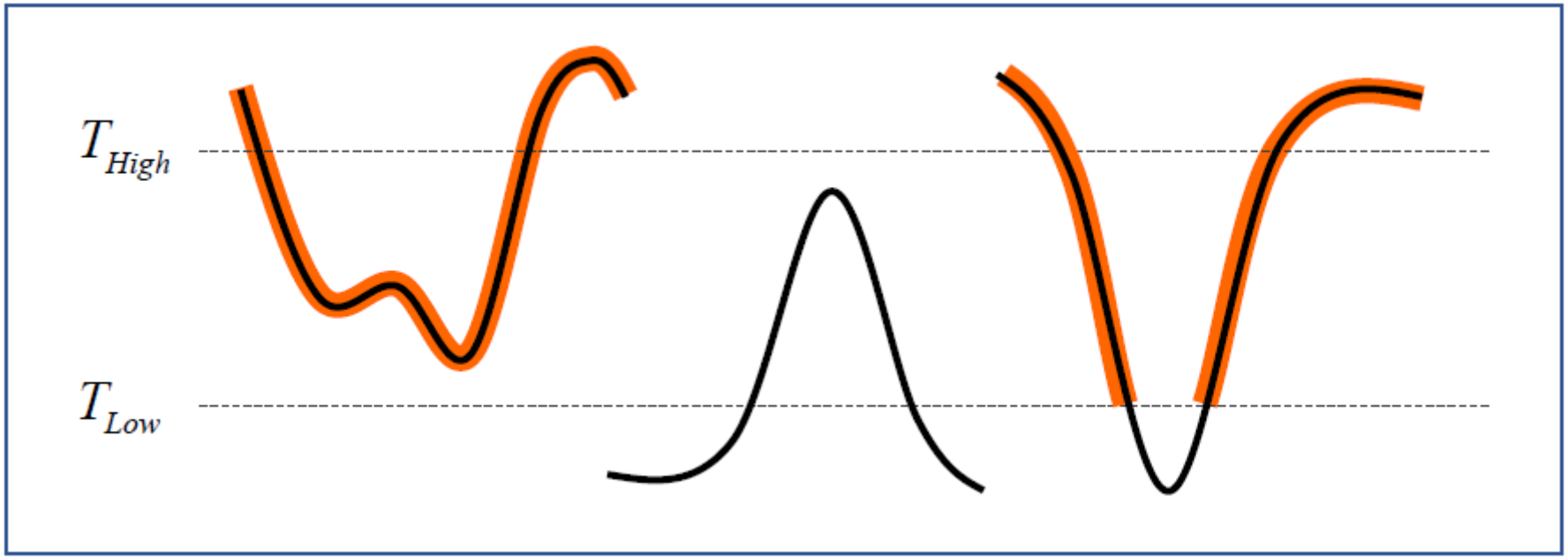
- 하나의 엣지가 여러 개의 픽셀로 표현되는 현상을 없애기 위하여 그래디언트 크기가 국지적 최대인 픽셀만을 엣지 픽셀로 설정
- 그래디언트 방향에 위치한 두개의 픽셀을 조사하여 국지적 최대를 검사



캐니 엣지 검출

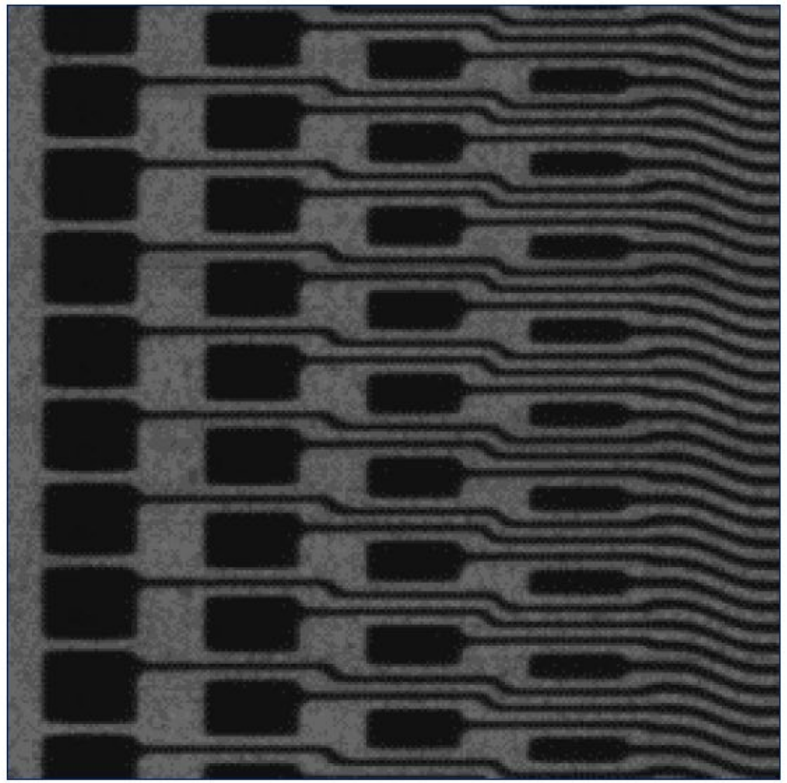
- 캐니 엣지 검출 단계

4. 히스테리시스 엣지 트래킹
   - 두개의 임계값을 사용 (강한엣지, 약한엣지)

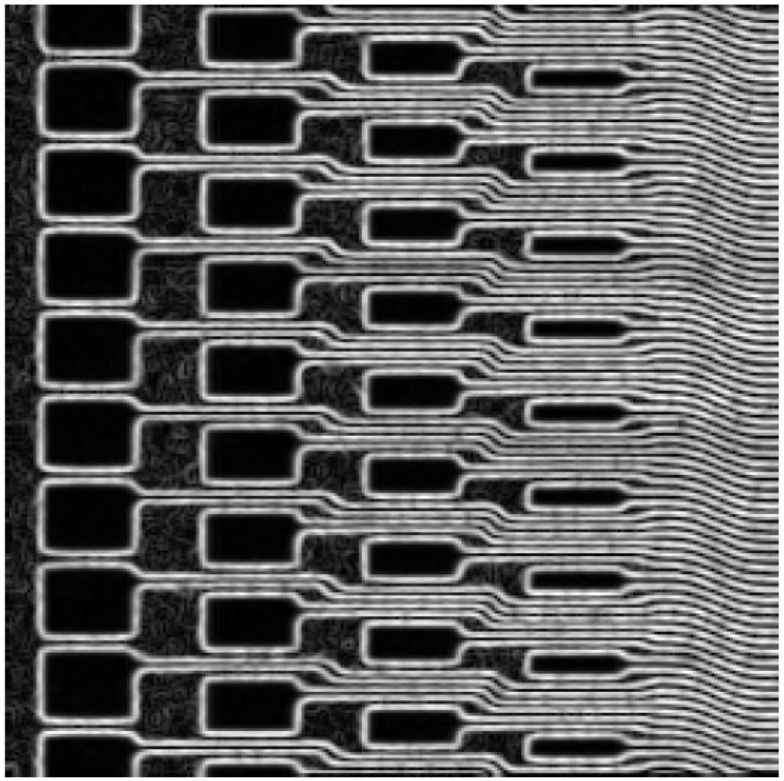


캐니 엣지 검출

- 캐니 엣지 검출 과정



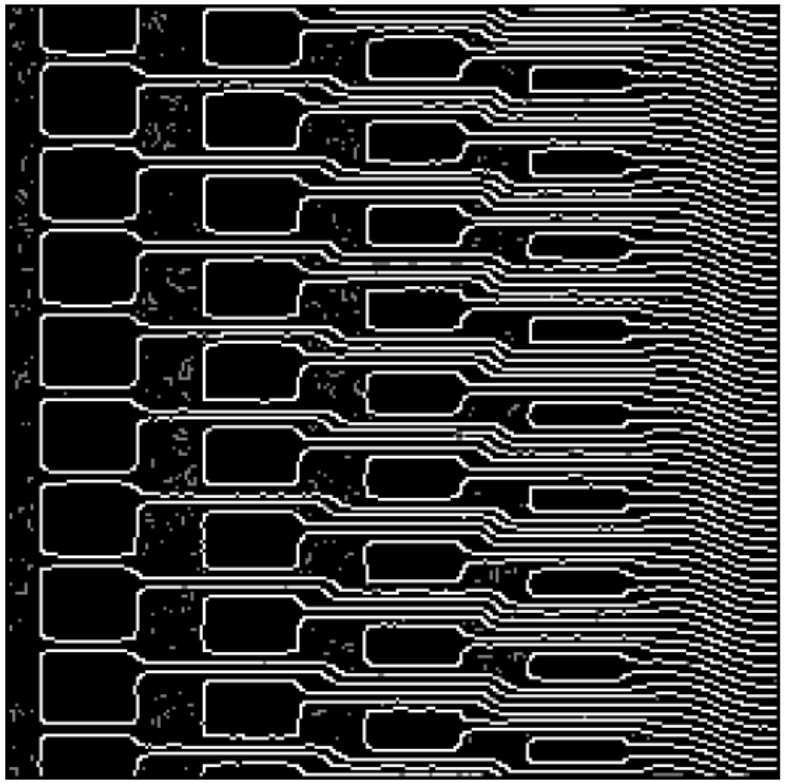
입력 영상



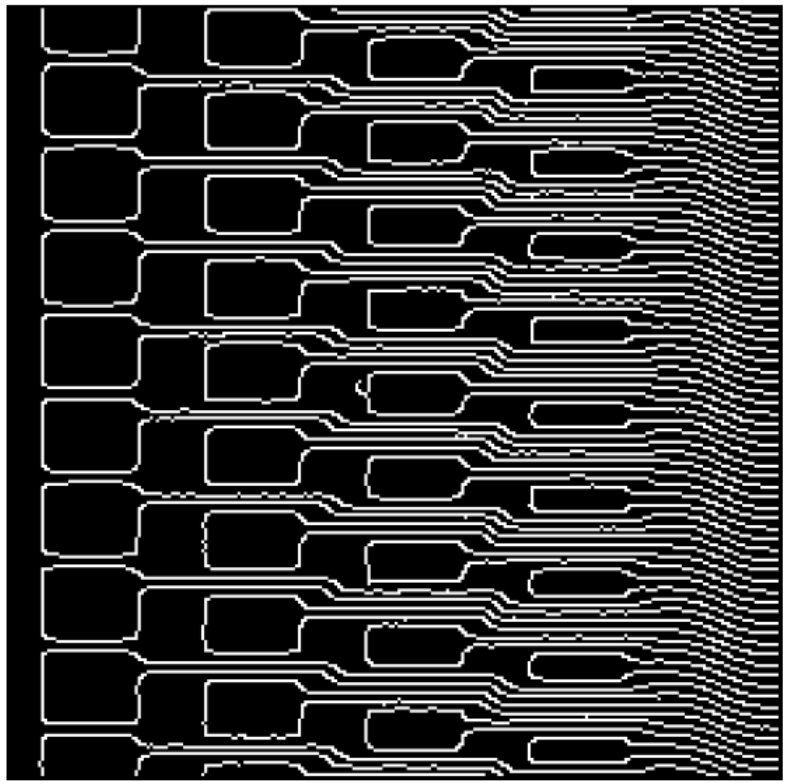
그래디언트 크기

캐니 엣지 검출

- 캐니 엣지 검출 과정



비최대 억제



히스테리시스 에지 트래킹

캐니 엣지 검출

- **캐니 엣지 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.

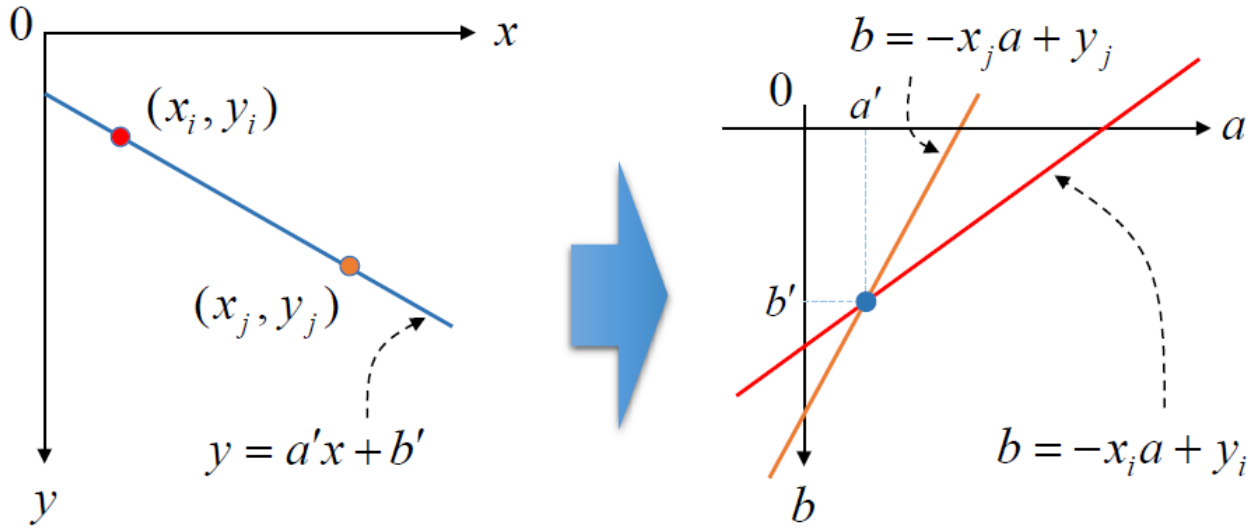
$$L_2 \text{ norm} = \sqrt{(dI/dx)^2 + (dI/dy)^2}\ ,\ L_1 \text{ norm} = |dI/dx| + |dI/dy|$$

L2norm이 정확하지만, 시간이 오래걸림

허프 변환 : 직선 검출

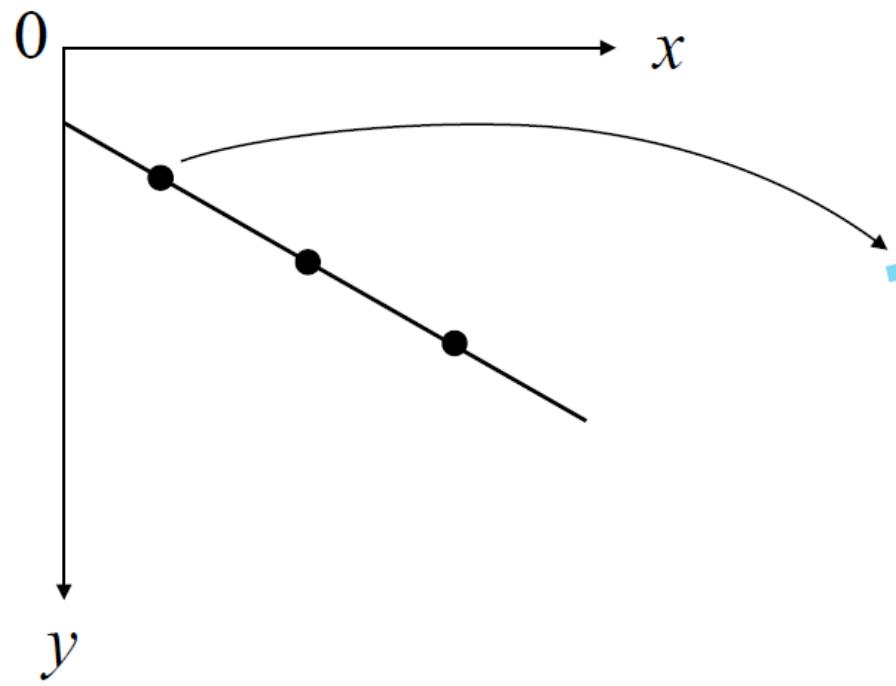
- 허프 변환 (Hough transform) 직선 검출
  - 2차원 좌표에서 직선의 방정식을 파라미터 공간으로 변환하여 직선을 찾는 알고리즘

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



허프 변환 : 직선 검출

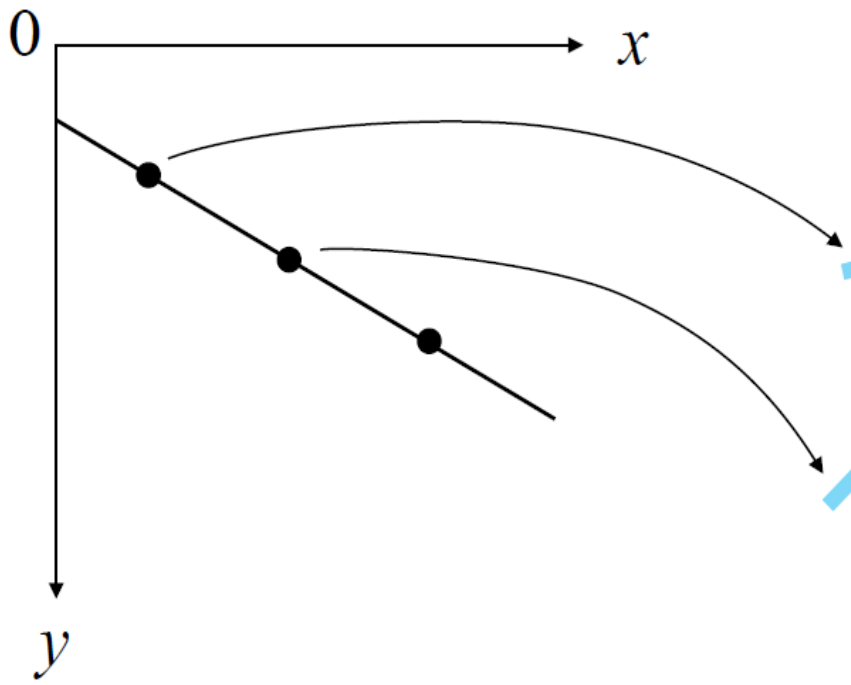
- 축적 배열
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



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

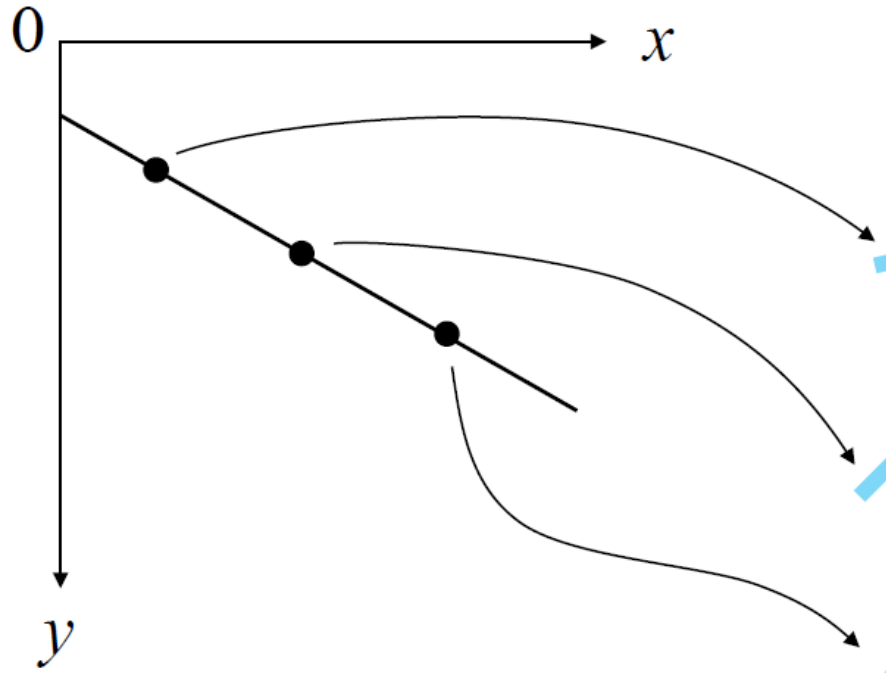
- 축적 배열
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



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

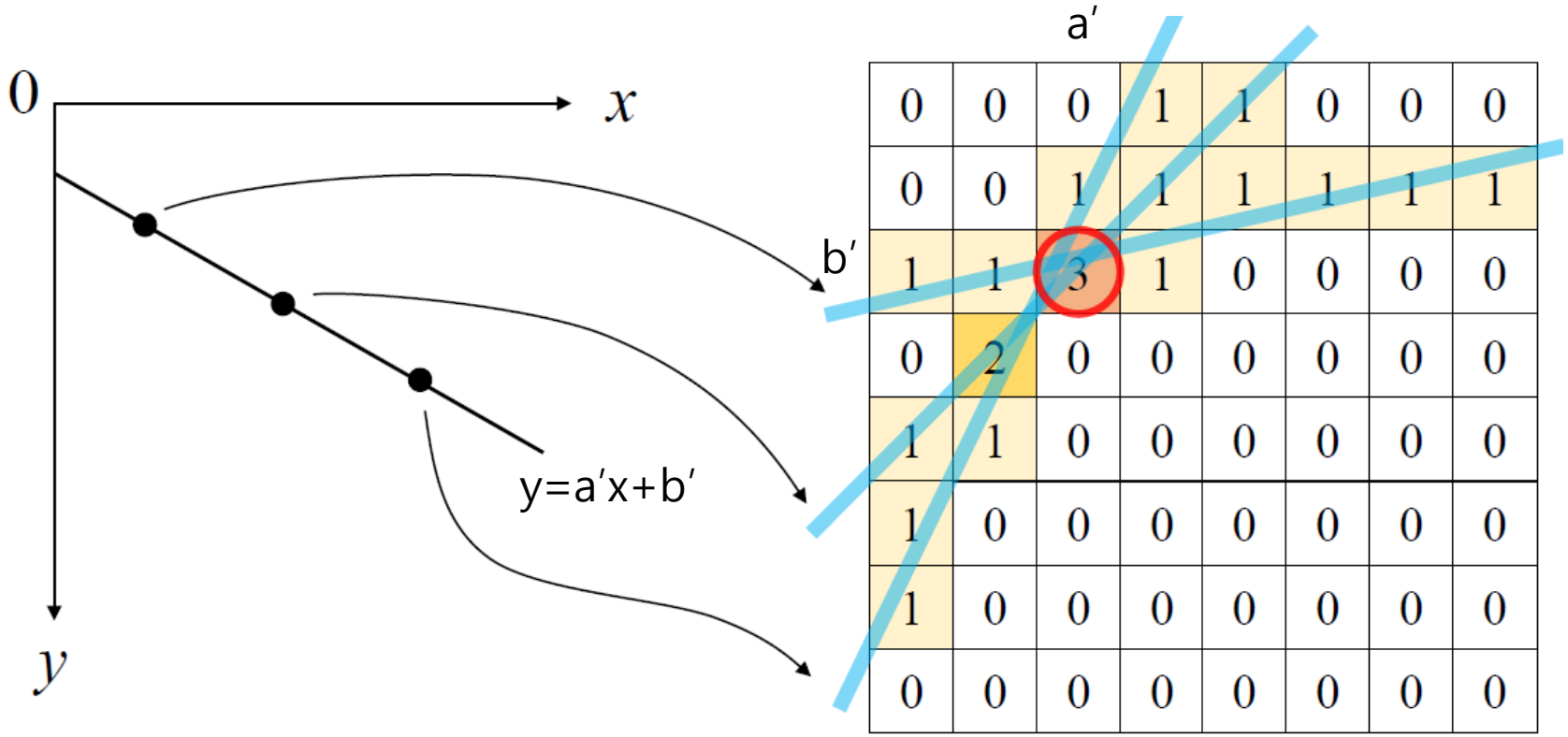
- 축적 배열
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



0	0	0	0	1	0	0	0
0	0	0	1	1	1	1	1
1	1	2	1	0	0	0	0
0	1	0	0	0	0	0	0
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

허프 변환 : 직선 검출

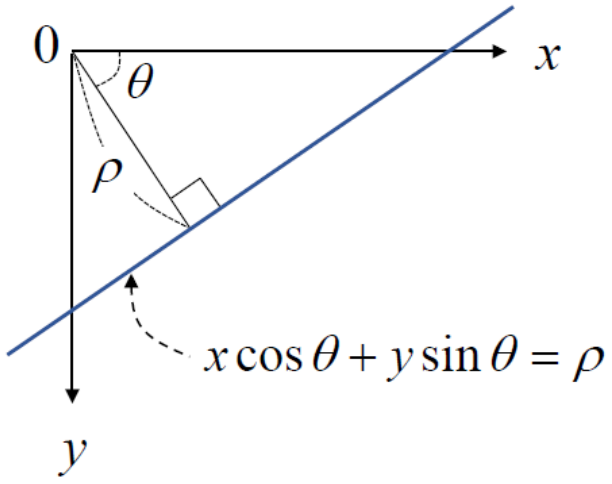
- 축적 배열
  - 직선 성분과 관련된 원소 값을 1씩 증가시키는 배열



허프 변환 : 직선 검출

- 직선의 방정식 $y = ax + b$ 를 사용할 때의 문제점
  - y 축과 평행한 수직선을 표현하지 못함 ➔ 극좌표계 직선의 방정식을 사용

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



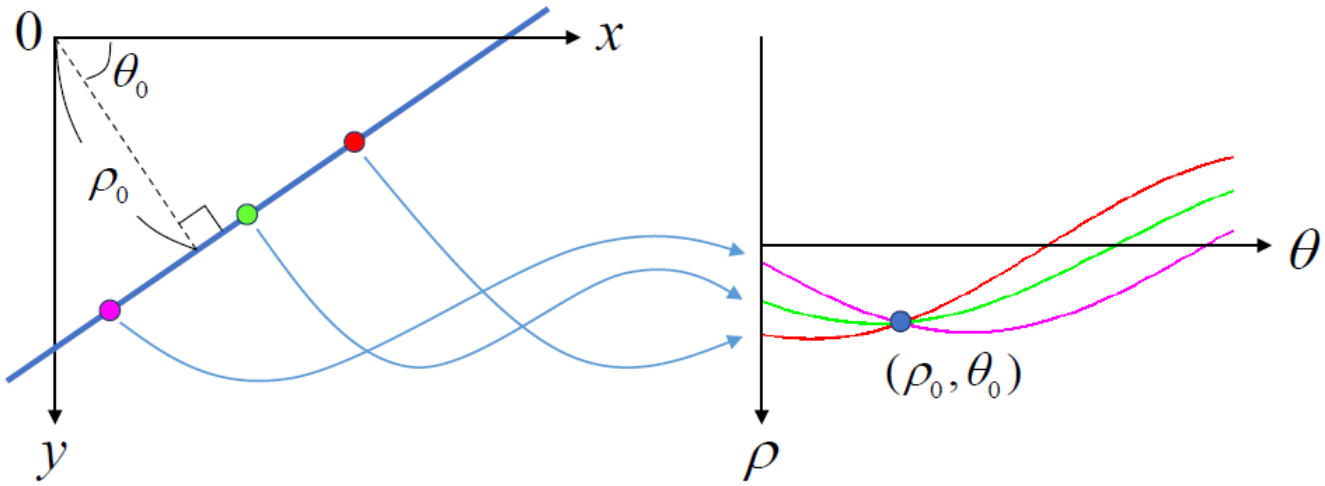
$$\begin{cases} \text{기울기} = -\dfrac{\cos\theta}{\sin\theta} \\ \text{y절편} = \dfrac{\rho}{\sin\theta} \end{cases}$$

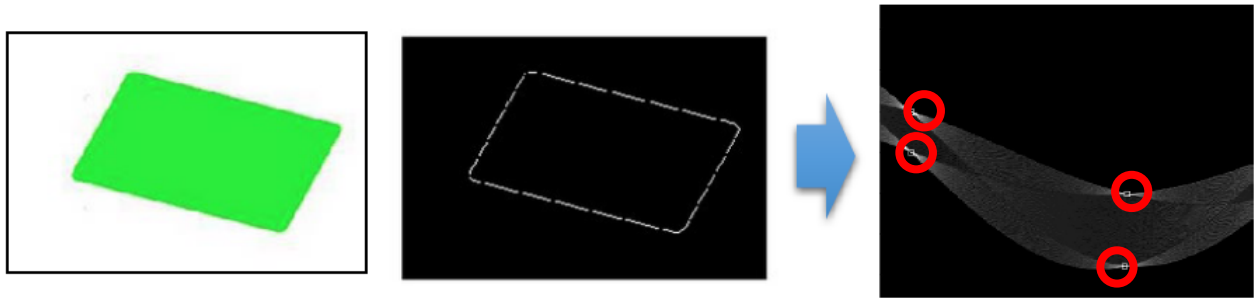
$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$

➔ $x\cos\theta + y\sin\theta = \rho$

허프 변환 : 직선 검출

- $x\cos\theta + y\sin\theta = \rho$ 방정식에 의한 파라미터 공간으로의 변환





허프 변환 : 직선 검출

- **허프변환에 의한 선분 검출**

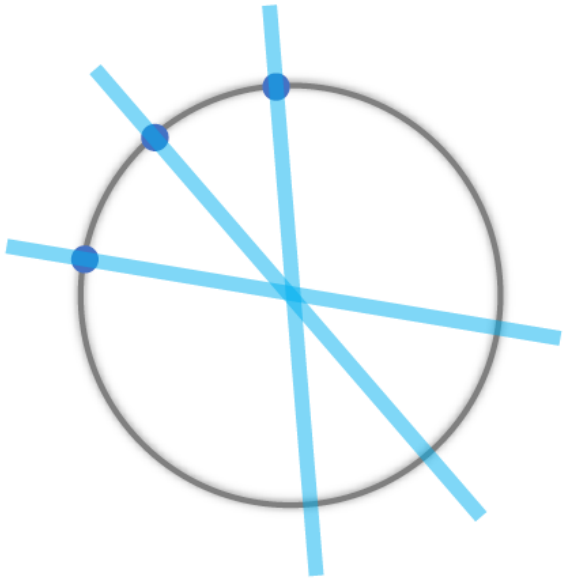
```
cv2.HoughLinesP(image, rho, theta, threshold, lines=None,
                minLineLength=None, maxLineGap=None) -> lines
```

- 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 : 원 검출

- 허프 변환을 응용하여 원을 검출할 수 있음
  - 원의 방정식: $(x-a)^2+(y-b)^2=c^2$ → 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

- **허프변환 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) 방향성을 알아야되기 때문, 내부에서 엣지 검출을함
- method: OpenCV 4.2 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