AI based Computer Vision

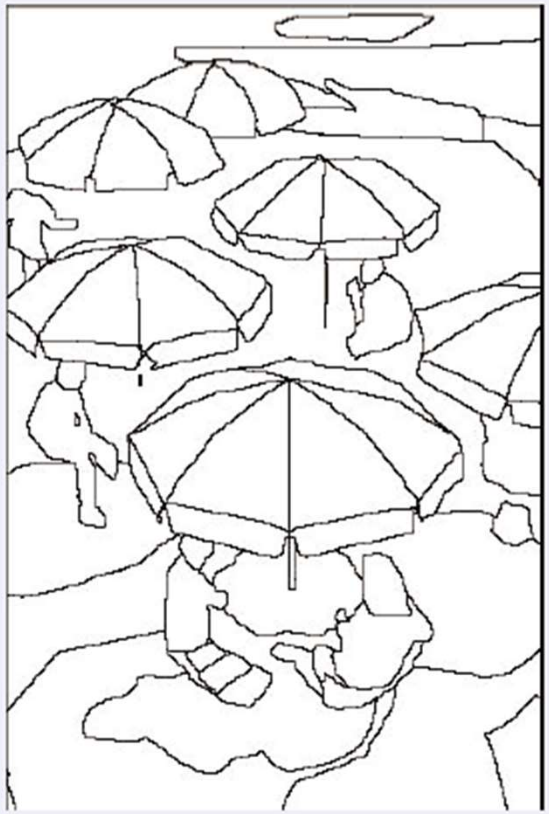
Edge Detection

Division of Computer Engineering
Byunghwan Jeon, PhD

■ 에지 검출과 영역 분할은 컴퓨터 비전 초창기부터 중요한 연구 주제

- 컴퓨터 비전 알고리즘이 사람 수준으로 분할할 수 있을까?

(a) 원래 영상



(b) 영역 영상(사람이 분할)

그림 4-1 영역 분할을 위한 BSDS 데이터셋

■ 에지와 영역은 쌍대 문제지만 다른 접근방법 사용

- 에지는 특성이 다른 곳을 검출하지만 영역은 유사한 화소를 묶는 방법 사용

■ 사람은 의미 분할semantic segmentation에 능숙

- 사람은 머리 속에 기억된 물체 모델을 활용하여 의미 분할
- 이 장에서 공부하는 고전적인 방법은 의미 분할 불가능
- 딥러닝은 의미 분할이 가능해져 혁신을 일으킴(9장)

■ 에지 검출 알고리즘

- 물체 내부는 명암이 서서히 변하고 경계는 급격히 변하는 특성을 활용

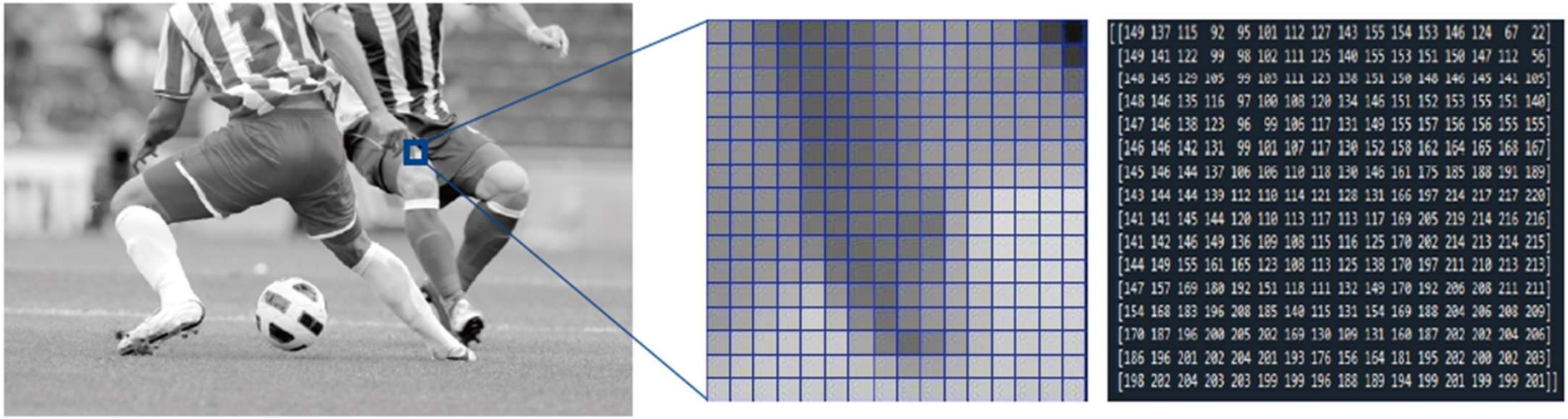


그림 4-2 명암 변화를 확인하기 위해 영상 일부를 확대

4.1.1 영상의 미분

■ 미분

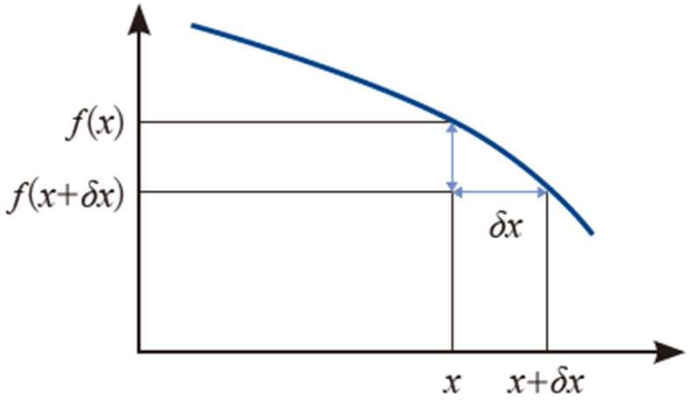
- 변수 x가 미세하게 증가했을 때 함수 변화량을 측정하는 수학 기법

$$f'(x)=\lim_{\delta x\to 0}\frac{f(x+\delta x)-f(x)}{\delta x} \qquad (4.1)$$

- 미분을 디지털 영상에 적용하면,

$$f'(x)=\frac{f(x+\delta x)-f(x)}{\delta x}=f(x+1)-f(x) \qquad (4.2)$$

- 실제 구현은 필터 u로 컨볼루션 (u를 에지 연산자라 부름)



(a) 연속 함수의 미분

(b) 디지털 영상의 미분(필터 u로 컨볼루션)

그림 4-3 연속 함수와 디지털 영상의 미분

4.1.2 에지 연산자

■ 현실 세계의 램프 에지

- 명암이 몇 화소에 걸쳐 변함

■ 1차 미분과 2차 미분

그림 4-4 현실 세계에서 발생하는 램프 에지

4.1.2 에지 연산자

■ 2차 미분

$$
\begin{aligned}
f''(x) &= \frac{f'(x)-f'(x-\delta)}{\delta} = f'(x)-f'(x-1) \\
&= \big(f(x+1)-f(x)\big)-\big(f(x)-f(x-1)\big) \\
&= f(x+1)-2f(x)+f(x-1)
\end{aligned}
\qquad (4.3)
$$

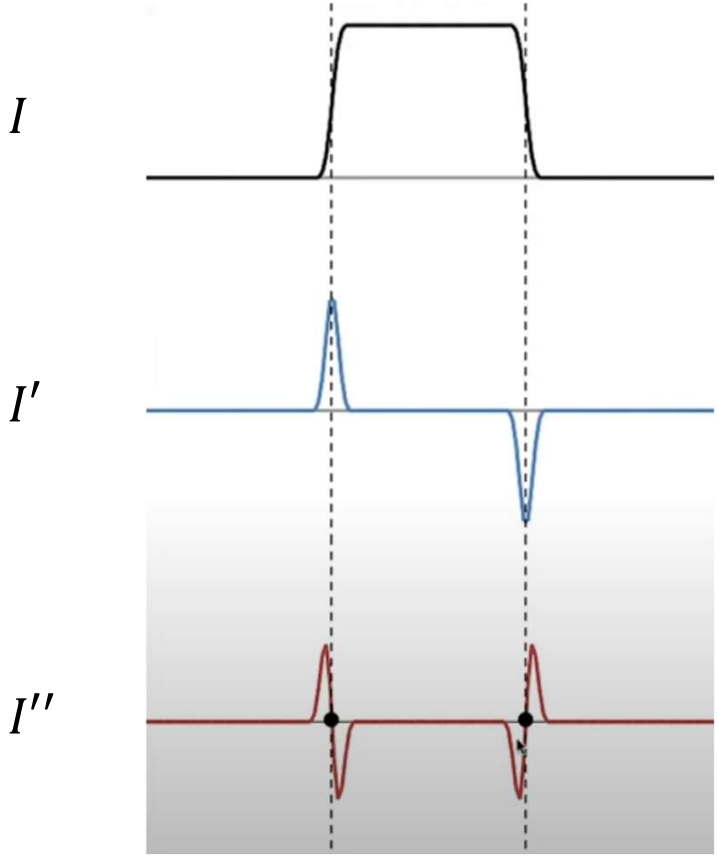
이 식을 구현하는 필터는

1	-2	1

4.1.2 에지 연산자

■ 에지 검출

- 1차 미분에서 봉우리 찾기
- 2차 미분에서 영교차$_{\text{zero-crossing}}$ 찾기



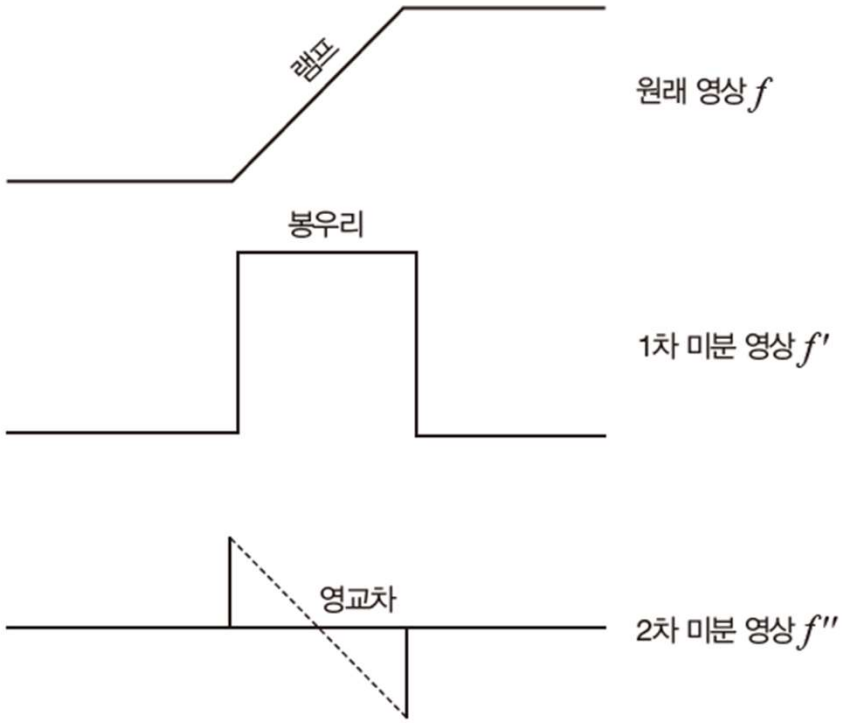


그림 4-5 램프 에지에서 발생하는 봉우리와 영교차

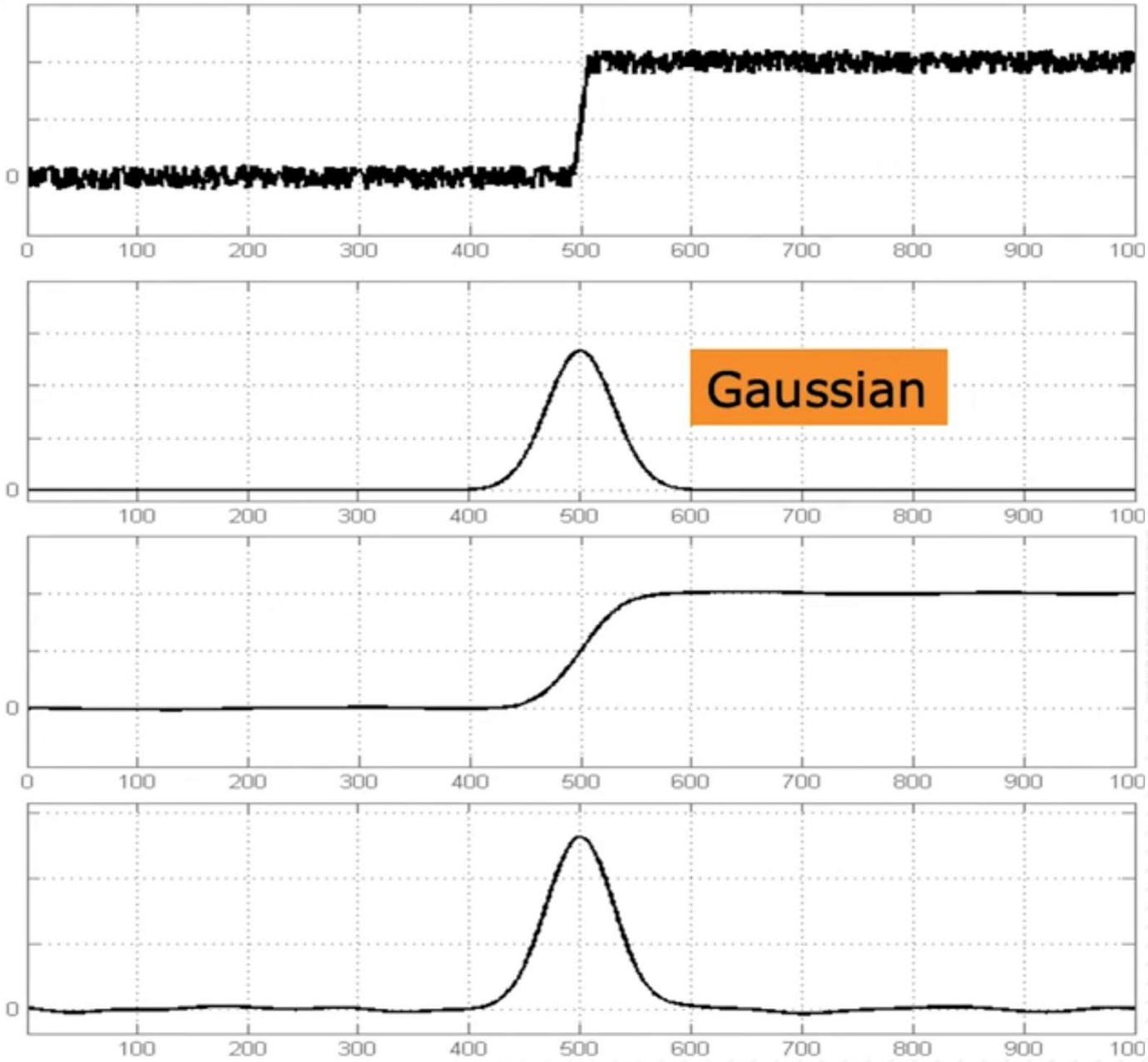
Edge Detection 과정

I

n_σ

$n_\sigma * I$

$(n_\sigma * I)'$



4.1.2 에지 연산자

■ 1차 미분에 기반한 에지 연산자

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

$$f_x'(y,x) = f(y,x+1) - f(y,x-1)$$
$$f_y'(y,x) = f(y+1,x) - f(y-1,x) \qquad (4.4)$$

이 식을 구현하는 필터는 u_x= | -1 | 0 | 1 | 와 u_y= | -1 | 0 | 1 |

- 또한 1차원을 2차원으로 확장

(a) 프레윗(Prewitt) 연산자

u_x=

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

u_y=

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

(b) 소벨(Sobel) 연산자

u_x=

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

u_y=

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

그림 4-6 에지 연산자

■ 에지 강도와 에지 방향

에지 강도: $s(y,x)=\sqrt{f_x^{'}(y,x)^2+f_y^{'}(y,x)^2}$

그레이디언트 방향: $d(y,x)=\arctan\left(\frac{f_y^{'}(y,x)}{f_x^{'}(y,x)}\right)$ (4.5)

4.1.2 에지 연산자

프로그램 4-1 소벨 에지 검출(Sobel 함수 사용)하기

```
01  import cv2 as cv
02
03  img=cv.imread('soccer.jpg')
04  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
05
06  grad_x=cv.Sobel(gray,cv.CV_32F,1,0,ksize=3)      # 소벨 연산자 적용
07  grad_y=cv.Sobel(gray,cv.CV_32F,0,1,ksize=3)
08
09  sobel_x=cv.convertScaleAbs(grad_x)               # 절댓값을 취해 양수 영상으로 변환
10  sobel_y=cv.convertScaleAbs(grad_y)
11
12  edge_strength=cv.addWeighted(sobel_x,0.5,sobel_y,0.5,0)   # 에지 강도 계산
13
14  cv.imshow('Original',gray)
15  cv.imshow('sobelx',sobel_x)
16  cv.imshow('sobely',sobel_y)
17  cv.imshow('edge strength',edge_strength)
18
19  cv.waitKey()
20  cv.destroyAllWindows()
```

cv.CV_8U(numpy의 uint8)로 변환
0보다 작으면 0, 255보다 크면 255로 바꿈

addWeighted(i1,a,i2,b,c)는 i1*a+i2*b+c를 계산
i1과 i2가 같은 데이터 형이면 결과는 같은 데이터 형, 다르면 오류 발생
i1과 i2가 CV_8U인데 계산 결과가 255를 넘으면 255를 기록

Code: convolution operator

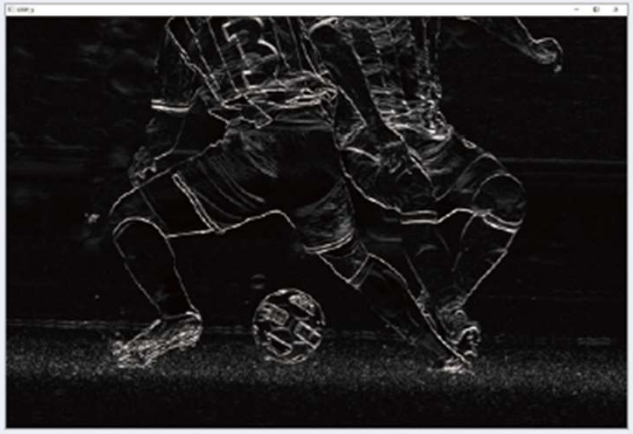

```python
def convolution(self, img, filter):

    #demension
    nY, nX = img.shape[0], img.shape[1]
    fnY, fnX = filter.shape[0], filter.shape[1]

    #j, i -> y, x
    #k, l -> fy, fx
    halfSize = fnY//2

    target_img = np.zeros_like(img, dtype='float32')
    #0으로 채워진 원본 소스와 동일한 크기의 2차원 배열을 만든다. #, dtype='int16'
    #image
    for j in range(halfSize, nY-halfSize): #Y
        for i in range(halfSize, nX-halfSize): #X
            conv_value = 0
            #filter
            for k in range(-halfSize, halfSize+1):
                for l in range(-halfSize, halfSize+1):
                    conv_value += (img[j+k][i+l] * filter[k+halfSize][l+halfSize])
                    ## 이 부분을 완성하면 컨볼루션 연산 완료

            target_img[j][i] = conv_value

    return target_img
```

Code


```python
def edgeDetection(self, img):

    gauss = np.array([[1, 2, 1],
                      [2, 4, 2],
                      [1, 2, 1]], dtype='float32')
    gauss = gauss/np.sum(gauss)

    dx = np.array([[-1, 0, 1],
                   [-2, 0, 2],
                   [-1, 0, 1]], dtype='float32')

    dy = np.array([[-1, -2, -1],
                   [0, 0, 0],
                   [1, 2, 1]], dtype='float32')

    smoothed = self.convolution(img, gauss)
    img_dx = self.convolution(smoothed, dx)
    img_dy = self.convolution(smoothed, dy)
    magnitude = np.sqrt(img_dx*img_dx + img_dy*img_dy)

    magnitude = self.clipping(magnitude) #미분결과가 음수~양수의 범위를 가지므로 절대값 + [0, 255]범위로 클립핑

    self.targetImg = magnitude

    return self.targetImg
```

4.1.2 에지 연산자

sobel_x



sobel_y

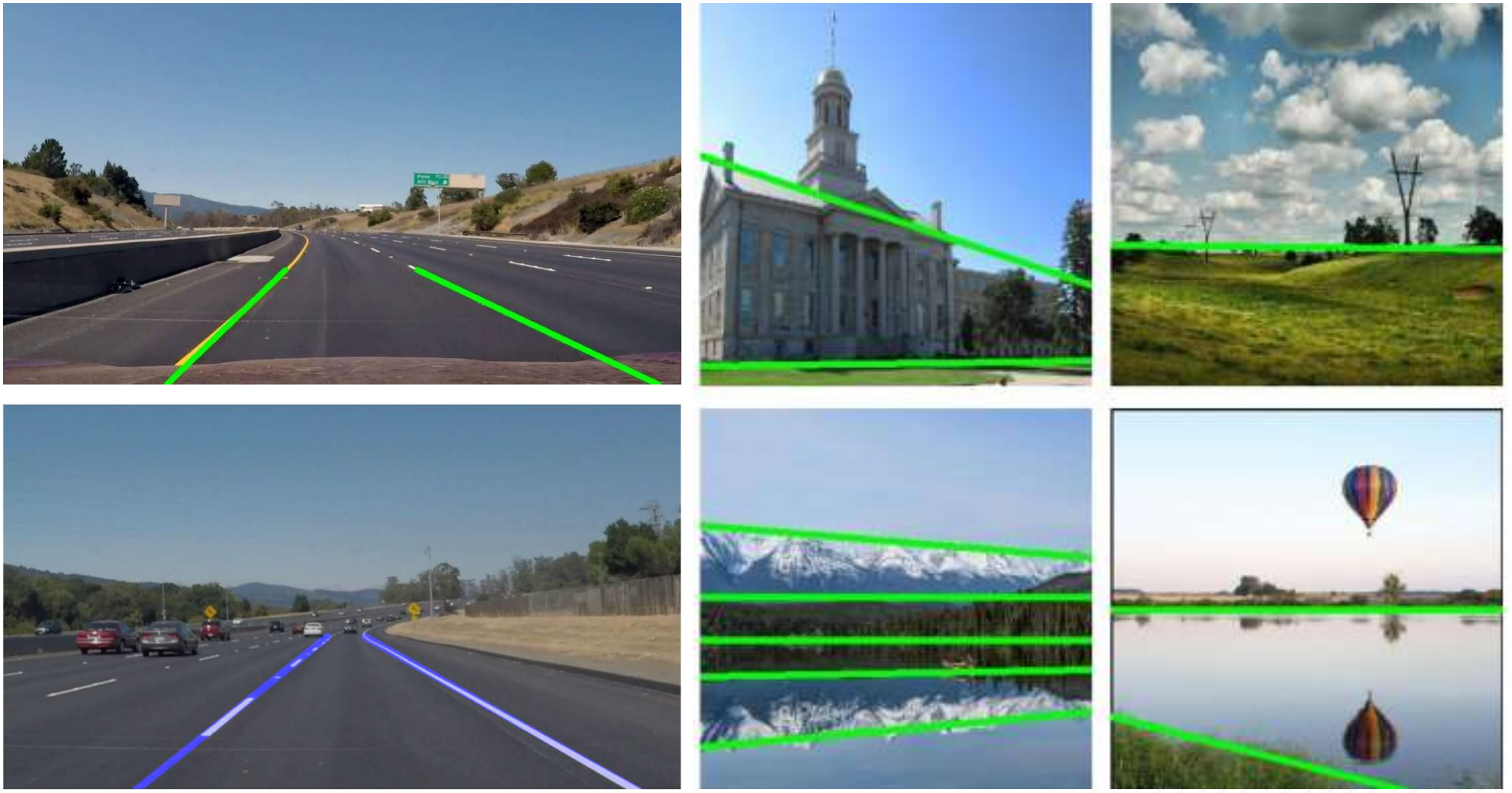
edge_strength

AI based Computer Vision

Hough Line Transform

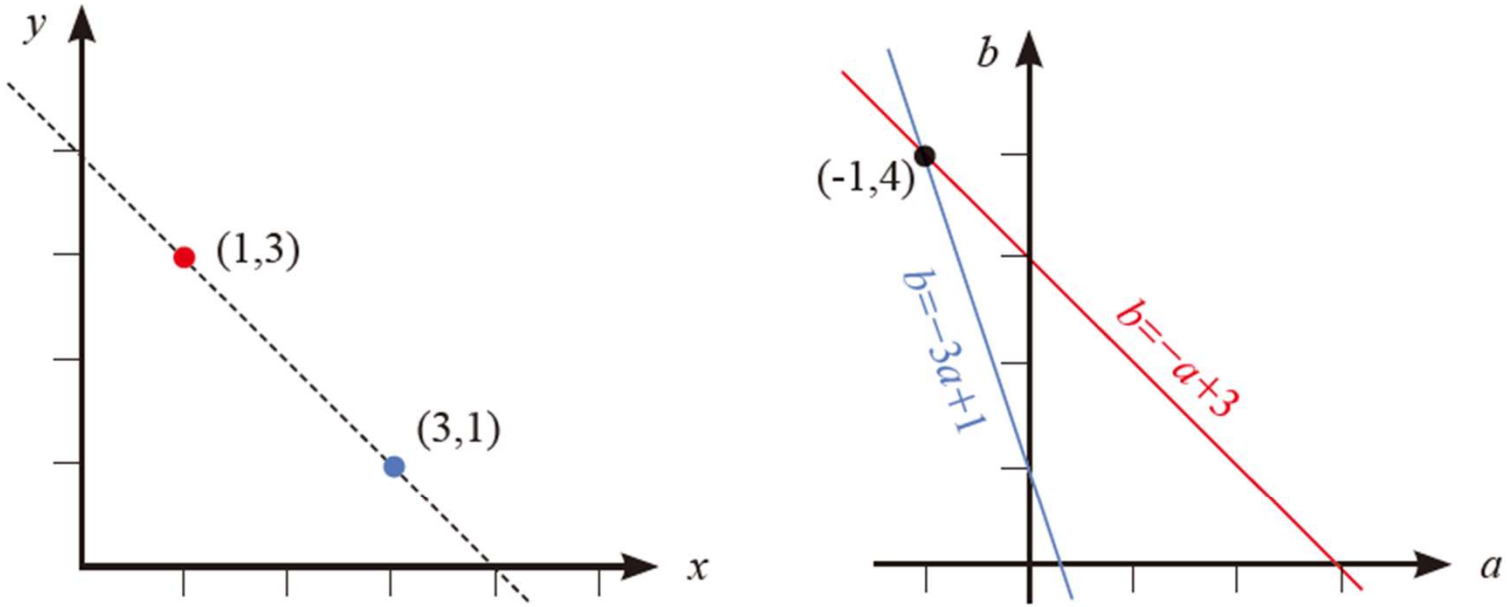
Division of Computer Engineering
Byunghwan Jeon, PhD

Hough Line Transform



Hough Line Transform

- **허프 변환은 끊긴 에지를 모아 직선 또는 원 등을 검출**
- **직선 검출의 원리**
  - 각각의 점 (y_i, x_i)에 대해 (b, a) 공간에 직선 $b = -ax_i + y_i$를 그림
  - (b, a) 공간에서 직선이 만나는 점을 절편과 기울기로 취함. 만나는 점은 투표로 알아냄



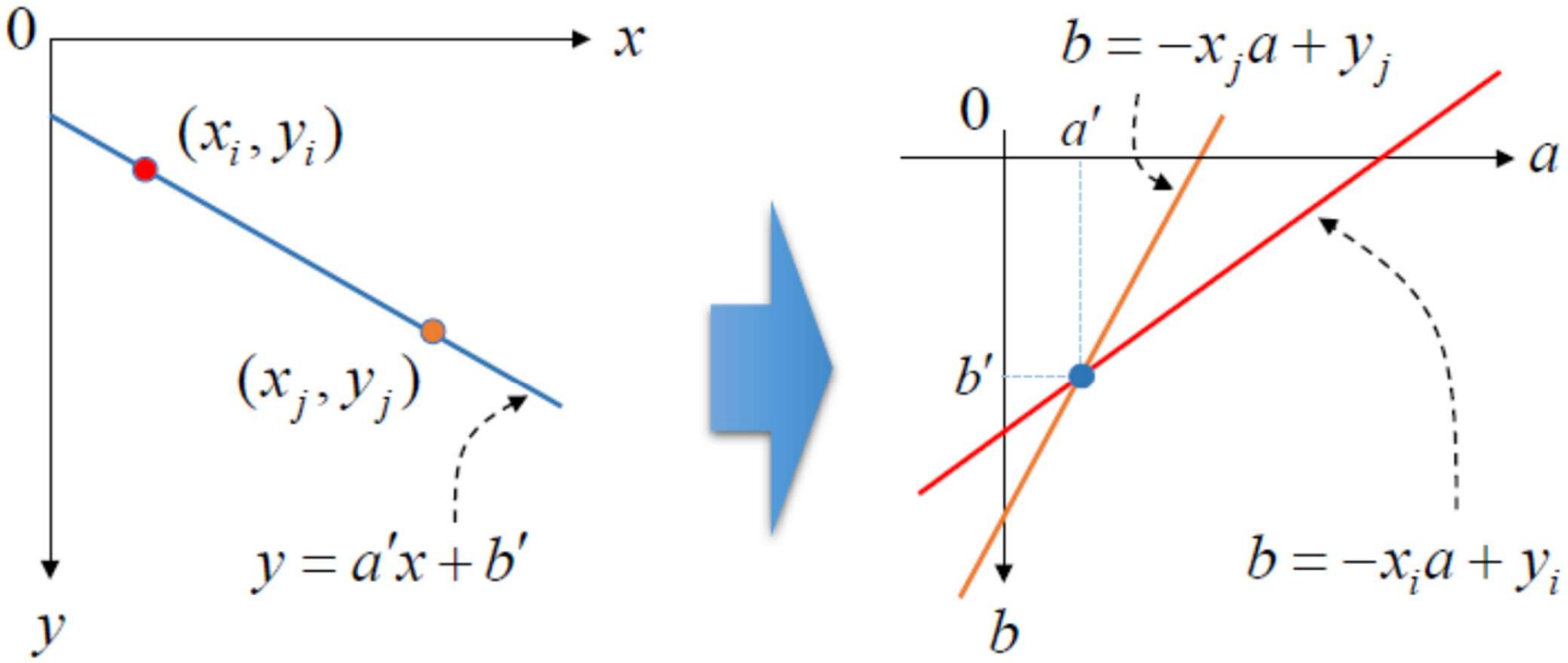
(a) (y, x)로 표현되는 영상 좌표 (b) (b, a)로 표현되는 공간으로 매핑

그림 4-10 허프 변환의 원리

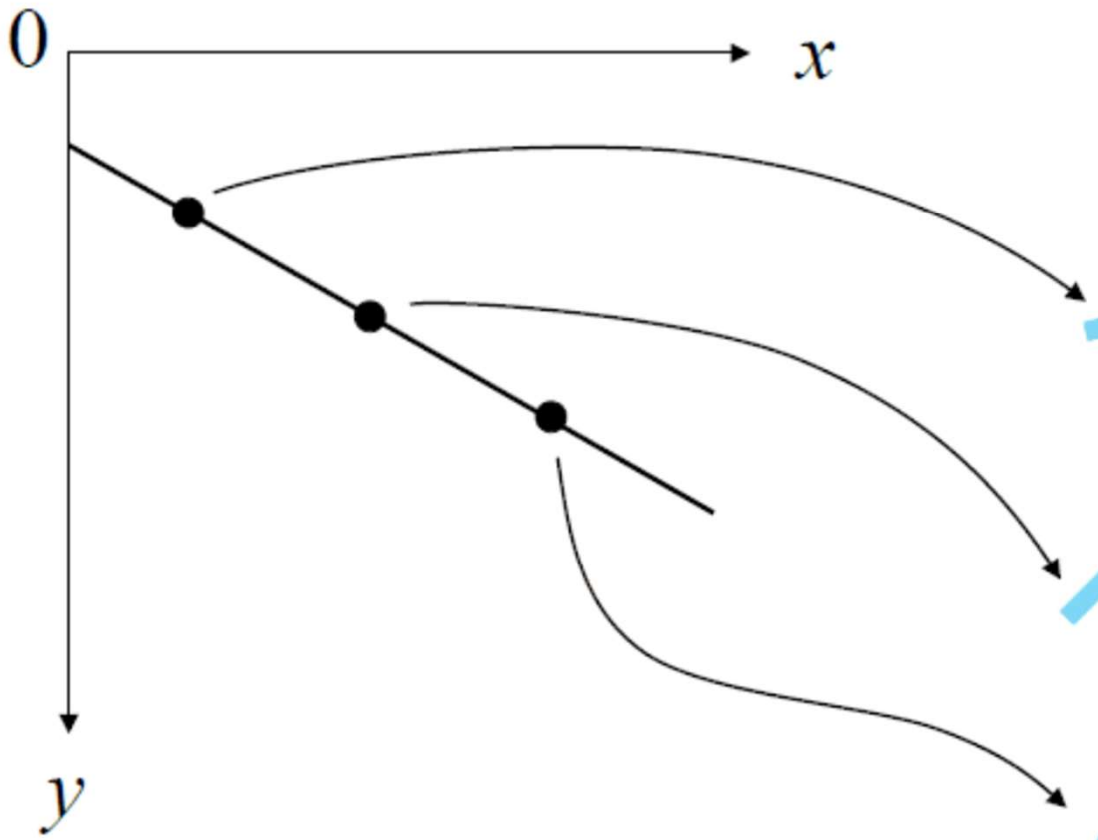
Hough Line Transform

Image space에서 parameter space로 변환

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



Accumulation in parameter space

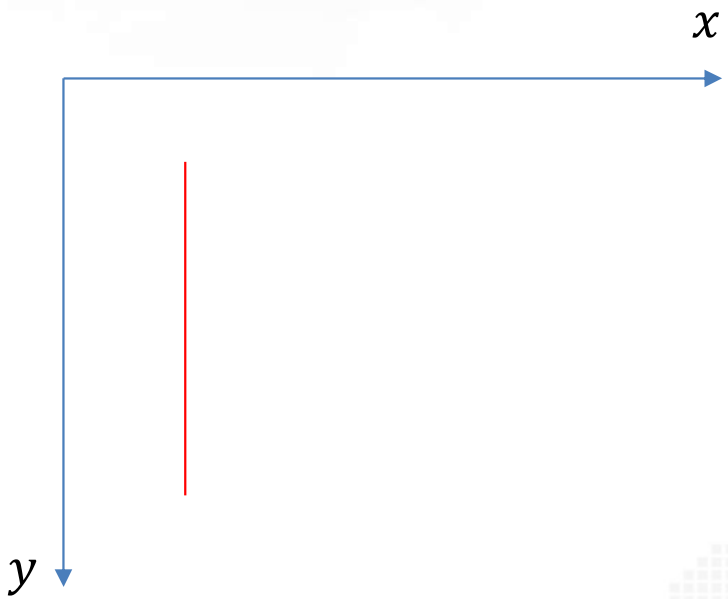


0	0	0	1	1	0	0	0
0	0	1	1	1	1	1	1
1	1	3	1	0	0	0	0
0	2	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

이미지

축적배열

기울기 표현의 한계

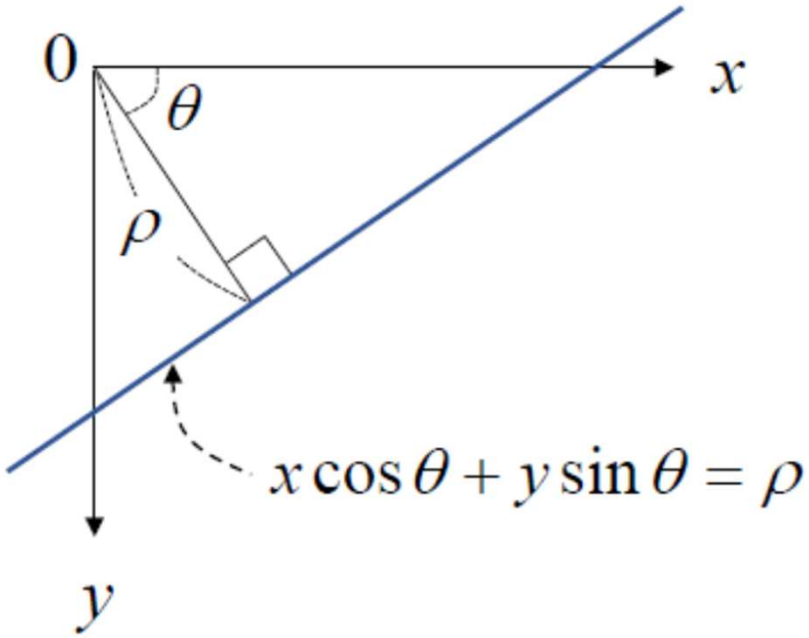


$$m = \frac{\Delta y}{\Delta x}$$

- $x = c$ 인 경우, 기울기로 표현이 불가능함
- 수직선의 경우 Δx가 0이므로 나눗셈 정의가 불가능함

극좌표의 표현

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



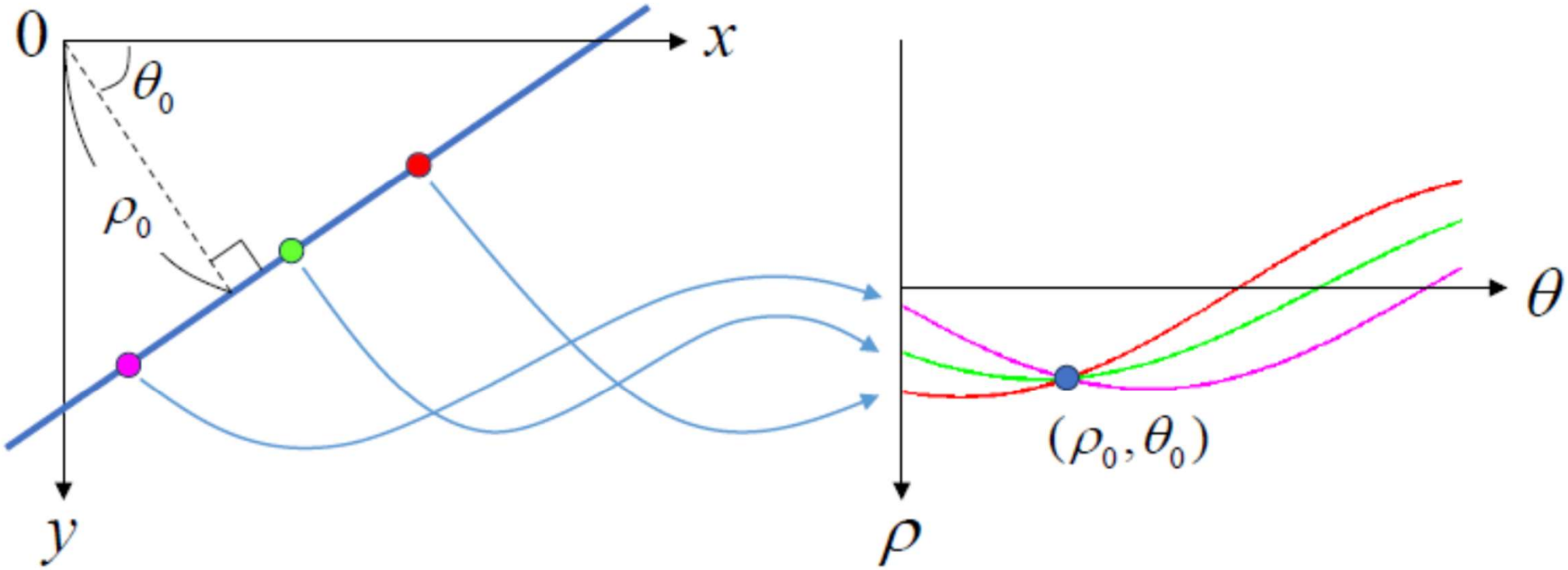
$$\begin{cases} \text{기울기} = -\dfrac{\cos\theta}{\sin\theta} \\ \text{y절편} = \dfrac{\rho}{\sin\theta} \end{cases}$$

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

➔ $x\cos\theta + y\sin\theta = \rho$

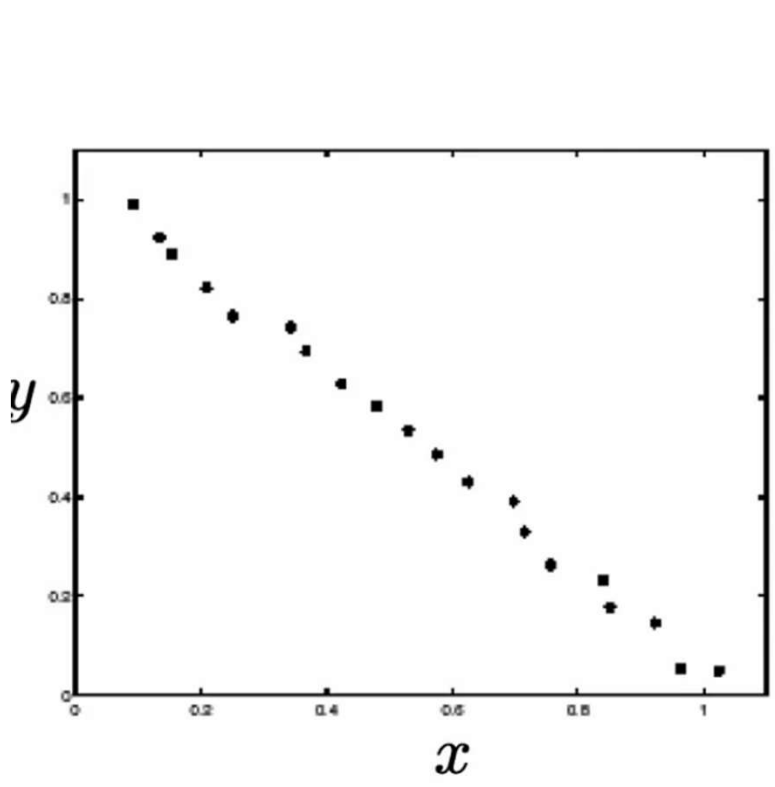
Image space에서 parameter space로 변환

극좌표의 표현



- Image space에서 하나의 x, y 조합은 parameter space에서 하나의 파형을 가짐
- 직선에 놓인 점이라면 parameter space어느 한 지점에서 교점이 생김

In real world



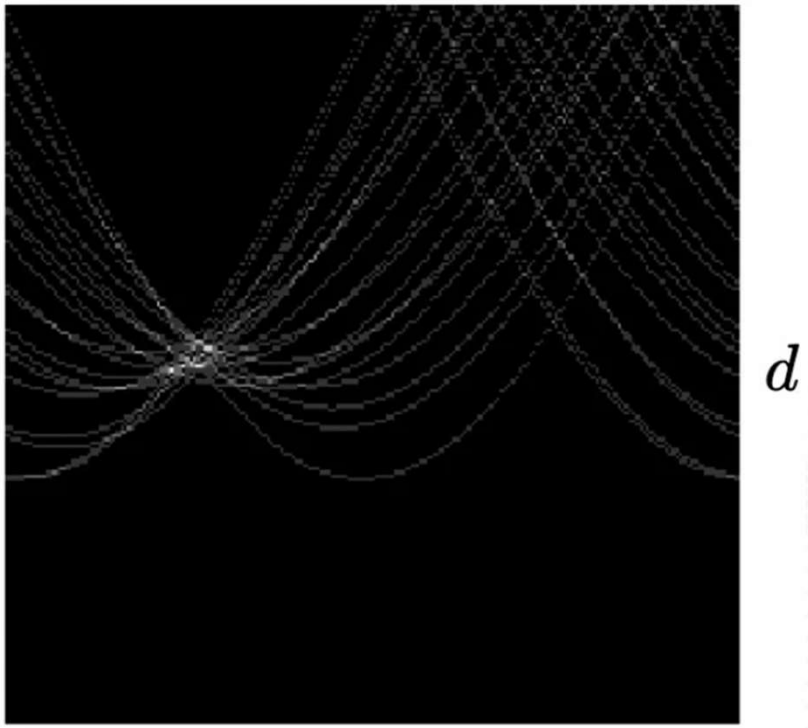


Image space
edge coordinates

Votes

In Hough Space

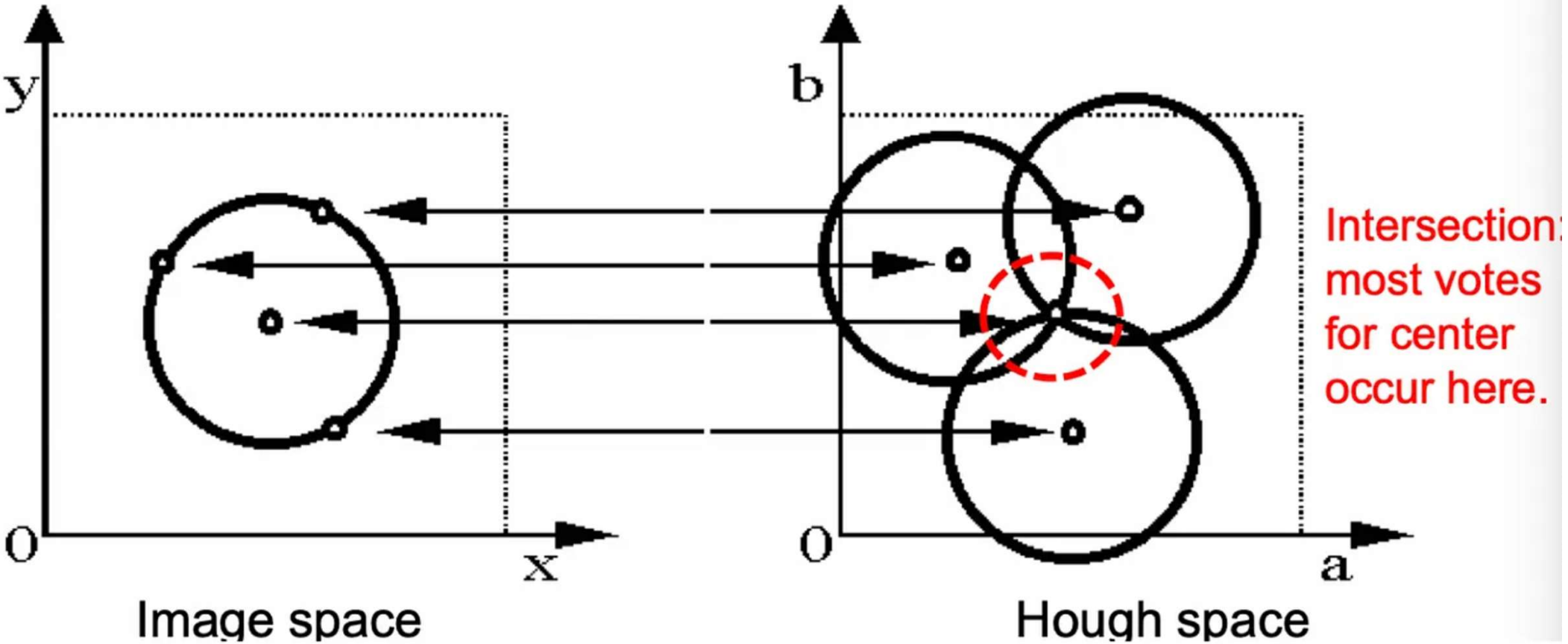
AI based Computer Vision

Hough Circle Transform

Division of Computer Engineering
Byunghwan Jeon, PhD

Circle Equation

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



3D Parameter space (a, b, r)

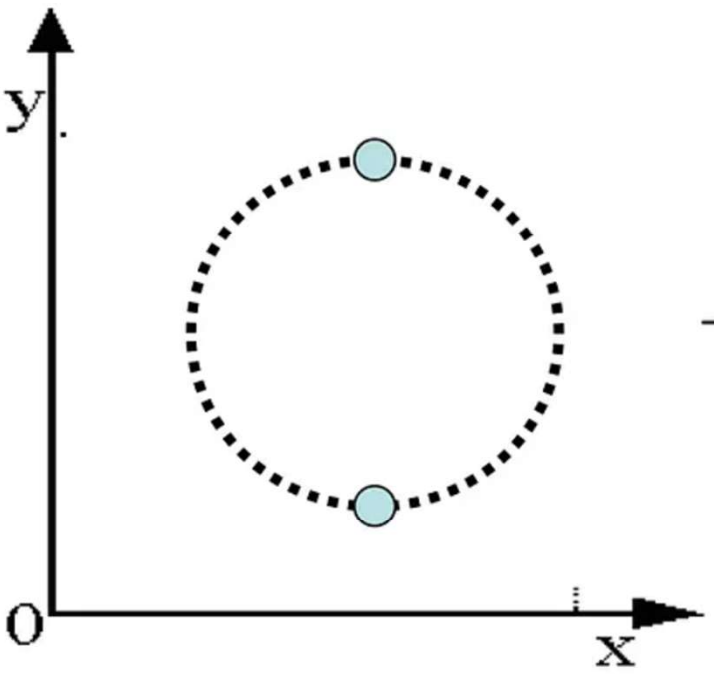
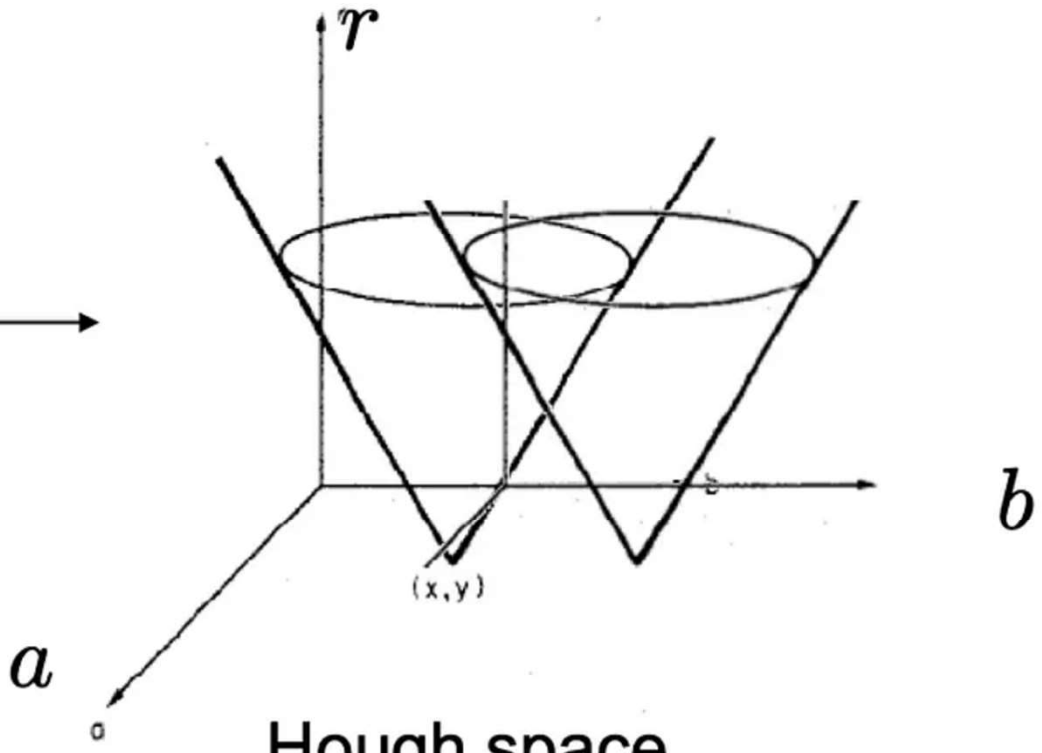


Image space



Hough space

Code


```python
def houghCircleTransform(self, edgeMap, min_rad, max_rad):
    nY, nX = edgeMap.shape
    accumulator = np.zeros((nY, nX, max_rad - min_rad + 1), dtype=np.uint8) #누적 배열

    for y in range(nY):
        print(print(y))
        for x in range(nX):

            if edgeMap[y, x] > 0:
                for r in range(min_rad, max_rad + 1):
                    for theta in range(360):

                        radian = theta*np.pi/180.0
                        x_ = int(x - r*np.cos(radian))
                        y_ = int(y - r*np.sin(radian))

                        if (0 <= x_ < nX) and (0 <= y_ < nY):
                            accumulator[y_, x_, r-min_rad] += 1

    centers = np.argwhere(accumulator > np.max(accumulator) * 0.8)
    # 최대 voting값의 80% 이상인 경우 원으로 판단
    #조건을 만족하는 요소의 좌표 (y, x, r)를 리턴함

    circles = []  # 검출된 원을 저장할 리스트
    cnt = 0
    for c in centers:
        y, x, r = c
        print(cnt, ", ", y, x, r)
        circles.append((x, y, r + min_rad))  # (x, y, 반지름) 형태로 저장
        cnt += 1

    return centers, accumulator
```

Circle Detection in our Framework