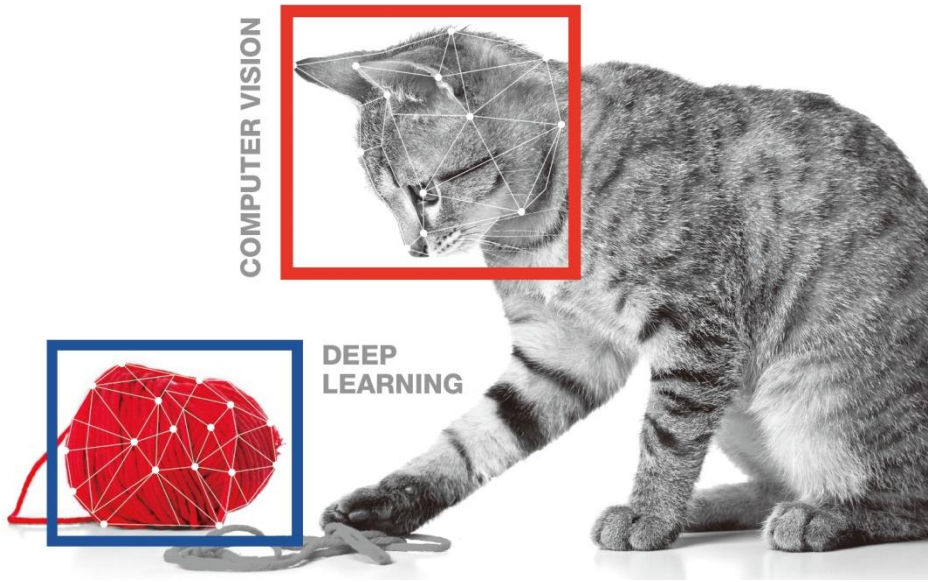


컴퓨터 비전과 딥러닝

Chapter 04 에지와 영역

Preview

■ 에지와 영역은 쌍대 문제지만 다른 접근방법 사용

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

물체 경계선

■ 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)

차례

4.1 에지 검출

■ 에지 검출 알고리즘

- 물체 내부는 명암이 서서히 변하고 경계는 급격히 변하는 특성을 활용

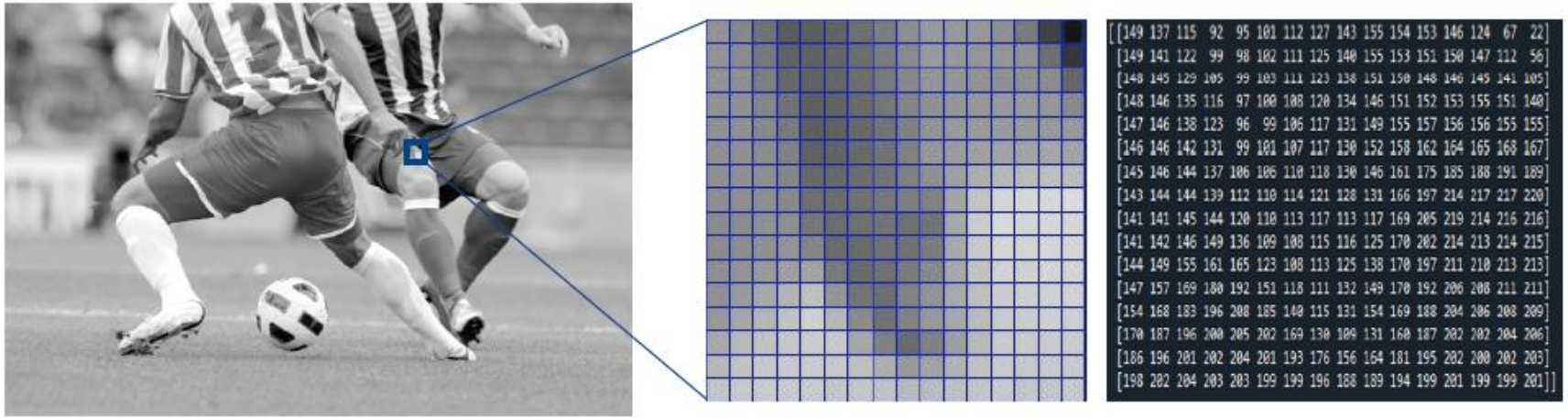


그림 4-2 명암 변화를 확인하기 위해 영상 일부를 확대

4.1 에지 검출

- 원리
  - 물체 내부나 배경은 변화가 없거나 작은 반면, 물체 경계는 변화가 큼
  - 이 원리에 따라 에지 검출 알고리즘은 명암, 컬러, 또는 텍스처 같은 특성의 변화량을 측정하고, 변화량이 큰 곳을 에지로 검출

4.1.1 디지털 영상의 미분

4.1.2 에지 모델과 연산자

- 에지 강도와 에지 방향

4.1.1 디지털 영상의 미분(변화율 측정)

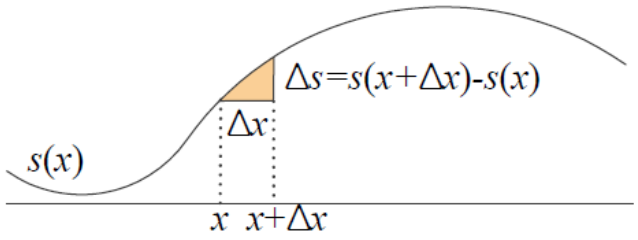
- 1차원
  - 연속 공간에서 미분

$$s'(x) = \frac{ds}{dx} = \lim_{\Delta x \to 0} \frac{s(x + \Delta x) - s(x)}{\Delta x} \qquad (3.1)$$

  - 디지털 (이산) 공간에서 미분

$$f'(x) = \frac{df}{dx} = \frac{f(x + \Delta x) - f(x)}{\Delta x} = f(x+1) - f(x) \qquad (3.2)$$

이에 해당하는 마스크 = | -1 | 1 | ← 에지 연산자: Δx=1인 경우



(a) 연속 함수의 미분

	0	1	2	3	4	5	6	7	8	9
디지털 영상 f	2	2	3	2	3	5	9	9	8	9
f의 도함수 f'	0	1	-1	1	2	4	0	-1	1	-
$\lvert f' \rvert$의 이진화	0	0	0	0	0	1	0	0	0	-

(b) 디지털 영상의 미분

에지화소

임계값=4

-1	1	로 컨볼루션

4.1.2 에지 모델과 연산자

- 계단 에지와 램프 에지
  - 자연 영상에서는 주로 램프 에지가 나타남

(a) 에지 모델

0	1	2	3	4	5	6	7	8	9	10	11	12	13
2	2	2	2	6	6	6	6	5	4	3	2	2	2

(b) 디지털 영상 f

0	0	0	4	0	0	0	-1	-1	-1	-1	0	0	-

(c) f의 1차 도함수 f'

0	0	4	-4	0	0	-1	0	0	0	1	0	-	-

(d) f의 2차 도함수 f''

-1	1

로 컨볼루션

3번 픽셀의 에지 위치를 찾기가 쉽다.
1차 미분으로 램프에지를 찾기가 어렵다

2차 미분값이 에지에서 영교차가 일어난다

-1	1

로 컨볼루션

- 2차 미분(라플라시안, Laplacian)

$$f''(x) = \frac{d^2 f}{dx^2} = f'(x) - f'(x-1)$$
$$= (f(x+1) - f(x)) - (f(x) - f(x-1))$$
$$= f(x+1) + f(x-1) - 2f(x) \qquad (3.3)$$

이에 해당하는 마스크 =

1	-2	1

4.1.2 에지 모델과 연산자

- 램프 에지에서 미분의 반응

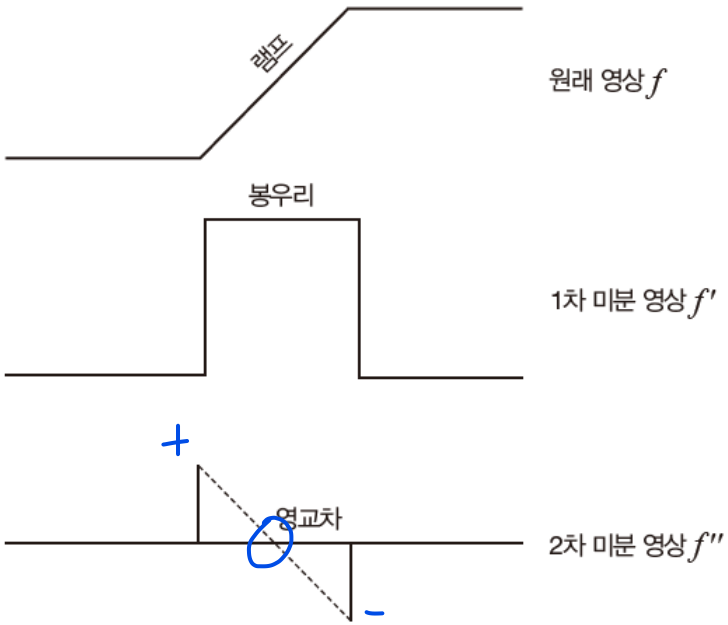


그림 4-5 램프 에지에서 발생하는 봉우리와 영교차

램프에지의 정확한 위치를 찾는 방법 필요하다.

- 에지 검출 과정
  - 1차 미분에서 봉우리 또는 2차 미분에서 영교차를 찾음
  - 두꺼운 에지에서 위치 찾기 적용

4.1.2 에지 모델과 연산자

- 현실에서는,
  - 잡음 때문에 스무딩 필요
    - 예) 100 100 100 100 170 170 170 …→ 98 97 101 102 168 170 169 …
  - Δx=2인 연산자로 확장

$$f'(x) = \frac{df}{dx} = \frac{f(x+1) - f(x-1)}{2} \qquad (3.4)$$

이에 해당하는 마스크 = | -1 | 0 | 1 |

  - 2차원으로 확장: 그레디언트 벡터

$$\nabla f(y,x) = \left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right) = (d_y, d_x) = (f(y+1,x) - f(y-1,x), f(y,x+1) - f(y,x-1)) \qquad (3.5)$$

이에 해당하는 마스크 : m_y = | -1 | / | 0 | / | 1 |, m_x = | -1 | 0 | 1 |

4.1.2 에지 모델과 연산자

- 정방형으로 2차원으로 확장

u_x=

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

u_y=

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

(a) 프레윗(Prewitt) 연산자

u_x=

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

u_y=

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

(b) 소벨(Sobel) 연산자

그림 4-6 에지 연산자

에지 강도와 에지 방향

- 에지 검출 연산

$$\text{그레이디언트} : \nabla f = \left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right) = (d_y, d_x)$$

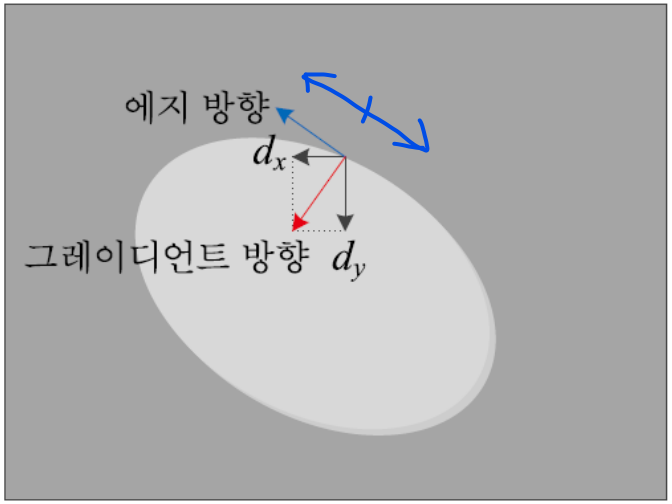
-에지강도: 경계선 검출에 사용: 특정 임계값 보다 크면 경계선으로 간주

그레이디언트 크기

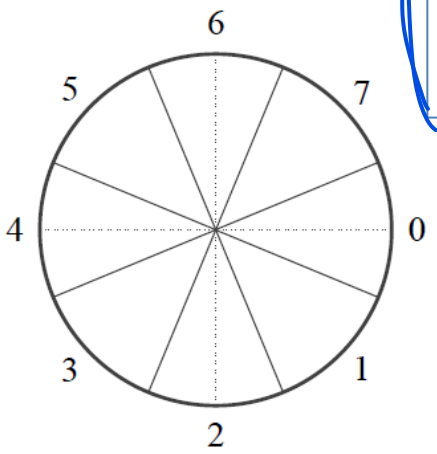
$$\text{에지 강도} : S(y,x) = \text{magnitude}(\nabla f) = \sqrt{d_y^2 + d_x^2} \qquad (3.6)$$

$$\text{그레이디언트 방향} : D(y,x) = \arctan\left(\frac{d_y}{d_x}\right)$$

-에지 방향은 그레디언트 방향에 수직.
-여기서 에지 방향은 에지 방향을 바라보고 섰을 때 왼쪽은 밝고 오른쪽은 어두운 방향으로 결정.
-반대도 가능, 일관성만 유지하면 됨.



(a) 에지 방향과 그레이디언트 방향



(b) 에지 방향의 양자화

에지 강도와 에지 방향

예제 3-1 **소벨 마스크를 이용한 에지 검출**

[그림 3-7]의 작은 예제 영상에 소벨 에지 연산자를 적용한다. (5,3)위치에 있는 화소에 대해서 앞에서 다룬 그레이디언트, 에지 강도, 에지 방향을 계산해 보자.

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	1	0
2	0	1	2	0	0	0	1	0
3	0	1	3	1	0	0	2	0
4	0	1	3	1	0	0	2	0
5	0	1	2	3	4	4	3	0
6	0	0	0	0	1	3	1	0
7	0	0	0	0	0	0	0	0

$d_y=(0\times1+0\times2+1\times1)+(3\times(-1)+1\times(-2)+0\times(-1))=-4$

$d_x=(0\times1+4\times2+1\times1)+(3\times(-1)+2\times(-2)+0\times(-1))=2$

$S(5,3) = ((-4)^2 + 2^2)^{\frac{1}{2}} = 4.47$

$D(5,3) = \arctan(-\frac{4}{2}) = -63.4°$

→ d_y와 d_x

→ 그레이디언트 방향

→ 에지 방향

에지방향은 d_x의 부호에 따라 결정됨.
$d_x>0$ 에지방향: -90~90

그림 3-7 소벨 에지 검출 예

그레이디언트는 $\nabla f = (dy, dx) = (-4, 2)$이다. 식 (3.6)을 적용하면 에지 강도는 4.47이고, 그레이디언트 방향은 −63.4°이다. 에지 방향은 그레이디언트 방향에 수직이므로 26.6°이다. 에지 방향을 [그림 3-6(b)]에 따라 양자화하면 1이 된다.

4.1.2 에지 연산자

이들 맵은 음수를 포함하며
실수이므로 32비트 실수 형
cv.CV_32F로 지정할 필요

[예시 4-1] 소벨 연산자 적용 과정

[그림 4-7]은 대각선을 기준으로 위쪽은 3, 아래쪽은 1인 가상의 영상에 소벨 에지 연산자를 적용하는 과정을 예시한다. 회색으로 표시한 (3,4) 화소에 대한 자세한 계산 과정을 설명한다.

	0	1	2	3	4	5	6	7
0	1	3	3	3	3	3	3	3
1	1	1	3	3	3	3	3	3
2	1	1	1	3	3	3	3	3
3	1	1	1	1	3	3	3	3
4	1	1	1	1	1	3	3	3
5	1	1	1	1	1	1	3	3
6	1	1	1	1	1	1	1	3
7	1	1	1	1	1	1	1	1

$$f_y'(3,4) = -6,\ f_x'(3,4) = 6$$

$$s(3,4) = \sqrt{6^2 + (-6)^2} = 8.485$$

$$d(3,4) = \arctan\left(\frac{-6}{6}\right) = -45°$$

→ f_x'와 f_y'

→ 그레이디언트 방향

⇢ 에지 방향

그림 4-7 소벨 연산자 적용 사례

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

4.1.2 에지 연산자

sobel_x

sobel_y

edge_strength

에지를 검출하는 방향성 필터 적용

- void Sobel(InputArray **src**, OutputArray **dst**, int **ddepth**, int **dx**, int **dy**, int **ksize**=3, double **scale**=1, double **delta**=0, int **borderType**=BORDER_DEFAULT)
  - **src** – input image.
  - **dst** – output image of the same size and the same number of channels as src .
  - **ddepth** – output image depth; the following combinations of src.depth() and ddepth are supported:
    - src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
    - src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
    - src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
    - src.depth() = CV_64F, ddepth = -1/CV_64F
  - when ddepth=-1, the destination image will have the same depth as the source;
  - **xorder** – order of the derivative x.
  - **yorder** – order of the derivative y.
  - **ksize** – size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
  - **scale** – optional scale factor for the computed derivative values; by default, no scaling is applied
  - **delta** – optional delta value that is added to the results prior to storing them in dst.

Python:

```
cv.Sobel( src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]] ) -> dst
```

♦ convertScaleAbs()

```
void cv::convertScaleAbs ( InputArray  src,
                           OutputArray dst,
                           double      alpha = 1 ,
                           double      beta = 0
                         )
```

Python:

```
cv.convertScaleAbs( src[, dst[, alpha[, beta]]] ) -> dst
```

```
#include <opencv2/core.hpp>
```

Scales, calculates absolute values, and converts the result to 8-bit.

On each element of the input array, the function convertScaleAbs performs three operations sequentially: scaling, taking an absolute value, conversion to an unsigned 8-bit type:

$$\texttt{dst}(I) = \texttt{saturate\_cast<uchar>}(|\texttt{src}(I) * \texttt{alpha} + \texttt{beta}|)$$

◆ addWeighted()

```
void cv::addWeighted ( InputArray  src1,
                       double      alpha,
                       InputArray  src2,
                       double      beta,
                       double      gamma,
                       OutputArray dst,
                       int         dtype = -1
                     )
```

Python:

```
cv.addWeighted( src1, alpha, src2, beta, gamma[, dst[, dtype]] ) -> dst
```

```
#include <opencv2/core.hpp>
```

Calculates the weighted sum of two arrays.

The function addWeighted calculates the weighted sum of two arrays as follows:

$$\texttt{dst}(I) = \texttt{saturate}(\texttt{src1}(I) * \texttt{alpha} + \texttt{src2}(I) * \texttt{beta} + \texttt{gamma})$$

영상의 라플라시안 계산

- void Laplacian(InputArray **src**, OutputArray **dst**, int **ddepth**, int **ksize**=1, double **scale**=1, double **delta**=0, int **borderType**=BORDER_DEFAULT)
  - **src** – Source image.
  - **dst** – Destination image of the same size and the same number of channels as src .
  - **ddepth** – Desired depth of the destination image.
  - **ksize** – Aperture size used to compute the second-derivative filters. The size must be positive and odd.

$$\texttt{dst} = \Delta \texttt{src} = \frac{\partial^2 \texttt{src}}{\partial x^2} + \frac{\partial^2 \texttt{src}}{\partial y^2}$$

  - This is done when ksize > 1 . When ksize == 1 , the Laplacian is computed by filtering the image with the following aperture: 3x3 Laplacian filter
  - **scale** – Optional scale factor for the computed Laplacian values. By default, no scaling is applied.
  - **delta** – Optional delta value that is added to the results prior to storing them in dst .

Python:

cv.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]]) -> dst

OpenCV BorderType

- 영상 경계 확장 방법
  - int **borderType**=BORDER_DEFAULT

BorderTypes 열거형 상수	설명
BORDER_CONSTANT	
BORDER_REPLICATE	
BORDER_REFLECT	
BORDER_REFLECT_101	
BORDER_REFLECT101	BORDER_REFLECT_101과 같음
BORDER_DEFAULT	BORDER_REFLECT_101과 같음

OpenCV 기본자료형

- primitive data type: unchar, bool, char, unsigned short, signed short, int, float, double

- A tuple of values of one of these primitive data types(all values in the tuple have the same type)
  - CV_<bit_depth>{S|U|F}C (C는 채널 수)
    - CV_8UC3 (8비트 양수 정수를 갖는 3개 채널 행렬) : 화소 하나당 24비트(B 8비트, G 8비트, R 8 비트) 입력,
    - CV_32SC1(32비트 부호있는 정수, 1개 채널),
    - CV_64FC1 (64비트 부동 소수점, 1개 채널)
    - CV_32FC1=CV_32F (최대 채널은 512)

- void **cartToPolar**(InputArray **x**, InputArray **y**, OutputArray **magnitude**, OutputArray **angle**, bool **angleInDegrees**=false)

•**x** – array of x-coordinates; this must be a single-precision or double-precision floating-point array.
•**y** – array of y-coordinates, that must have the same size and same type as x.
•**magnitude** – output array of magnitudes of the same size and type as x.
•**angle** – output array of angles that has the same size and type as x; the angles are measured in radians (from 0 to 2*Pi) or in degrees (0 to 360 degrees).
•**angleInDegrees** – a flag, indicating whether the angles are measured in radians (which is by default), or in degrees.

$$\texttt{magnitude}(I) = \sqrt{\texttt{x}(I)^2 + \texttt{y}(I)^2},$$
$$\texttt{angle}(I) = \texttt{atan2}(\texttt{y}(I), \texttt{x}(I))[\cdot 180/\pi]$$

```
cv::Mat norm, dir;
cv::Sobel(image, sobelX, CV_32F, 1, 0);
cv::Sobel(image, sobelY, CV_32F, 0, 1);
cv::cartToPolar(sobelX, sobelY, mag, ang)
```

- (30,20)위치에 있는 픽셀의 sobel적용 후 dx, dy, 그리고 magnitude, angle(degree, 360)를 구해보시오

4.2 캐니 에지

- 1986년에 Canny 에지 발표 [Canny86] 현재 가장 널리 사용
  - 에지 검출을 최적화 문제로 해결
  - 세 가지 기준

1. 최소 오류율 : 거짓 긍정과 거짓 부정이 최소여야 한다. 즉, 없는 에지가 생성되거나 있는 에지를 못 찾는 경우를 최소로 유지해야 한다.
2. 위치 정확도 : 검출된 에지는 실제 에지의 위치와 가급적 가까워야 한다.
3. 에지 두께 : 실제 에지에 해당하는 곳에는 한 두께의 에지만 생성해야 한다.

A computational approach to edge detection
J Canny - Pattern Analysis and Machine Intelligence, IEEE ..., 1986 - ieeexplore.ieee.org
Abstract-This paper describes a computational approach to edge detection. The success of the approach depends on the definition of a comprehensive set of goals for the computation of edge points. These goals must be precise enough to delimit the desired behavior of the ...
19422회 인용 관련 학술자료 전체 17개의 버전 Web of Science: 6073 인용 저장

← Google scholar

4.2 캐니 에지

캐니 에지 검출 알고리즘

알고리즘 3-3 캐니 에지 검출(스케치 버전)

입력 : 영상 $f(j,i), 0 \le j \le M-1, 0 \le i \le N-1$, 가우시안의 표준편차 σ 블러, 스무딩

출력 : 에지 영상 $e(j,i), 0 \le j \le M-1, 0 \le i \le N-1$ // 에지는 1, 비에지는 0인 이진 영상

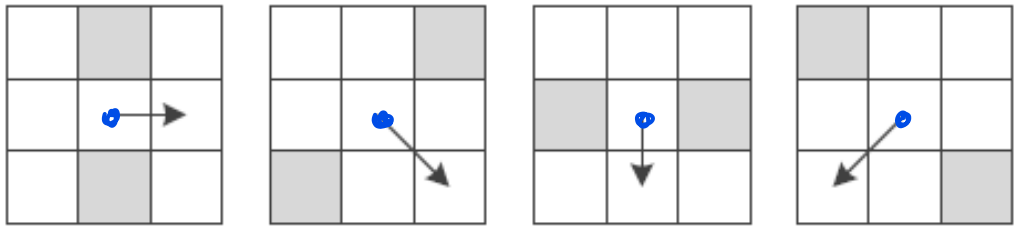
1	입력 영상 f에 σ 크기의 가우시안 스무딩을 적용한다.
2	결과 영상에 소벨 연산자를 적용하여 에지 강도와 에지 방향 맵을 구한다.
3	비최대 억제를 적용하여 얇은 두께 에지 맵을 만든다.
4	이력 임계값을 적용하여 거짓 긍정을 제거한다.

step1: 노이즈 제거
step2: 그레디언트 계산
에지 강도 = 그래디언트 크기=$magnitude(\nabla f)$
그레디언트 방향 = $\arctan\left(\frac{\Delta y}{\Delta x}\right)$, 에지 방향= 그레디언트 방향과 수직
step3: 에지 방향으로 주변 픽셀과 비교해서 최대값이 아닌 곳은 0으로
step4: 임계값을 2개(hysteresis) 사용하여 확실한 에지 판정

4.2 캐니 에지

비최대 억제(에지에서 제거)

- 이웃 두 화소보다 에지 강도가 크지 않으면 억제됨



(a) 에지 방향=0 (b) 에지 방향=1 (c) 에지 방향=2 (d) 에지 방향=3

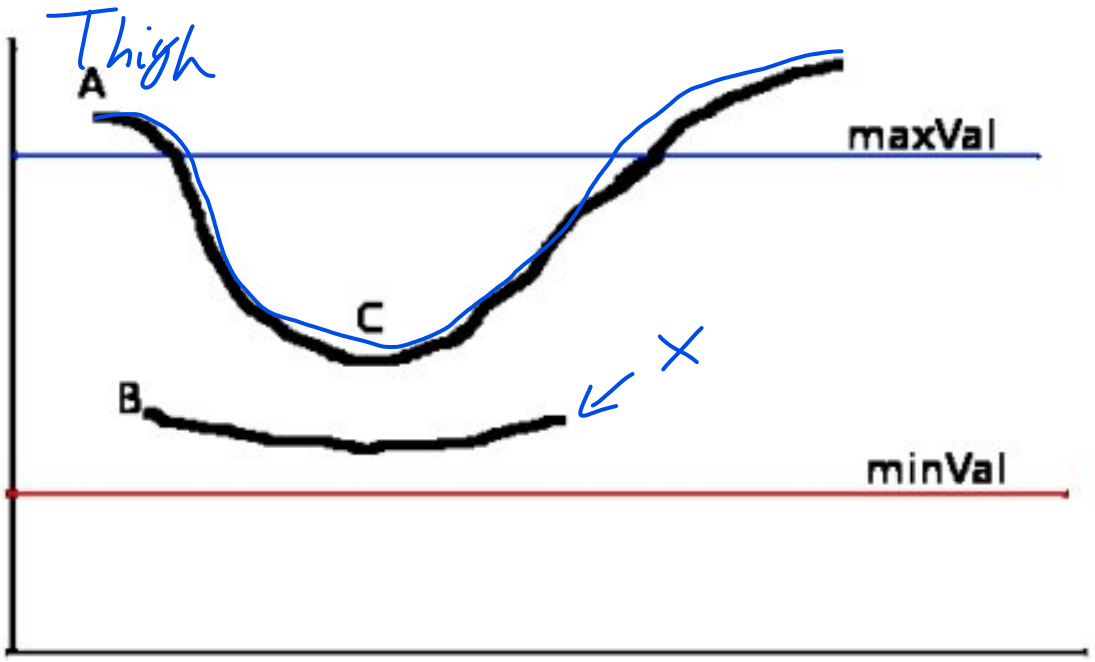
주변 2개보다 크면 에지이다

그림 3-17 비최대 억제를 위한 두 이웃 화소(방향 4는 0, 5는 1, 6은 2, 7은 3과 같음)

이력 임계값

- 두 개의 임계값 T_{high}와 T_{low} 사용하여 거짓 긍정 줄임
  - 거짓 긍정:에지가 아닌 경우 에지로 판정된 경우
- 에지 추적은 T_{high}를 넘는 화소에서 시작, 추적 도중에는 T_{low} 적용
- T_{low}를 사용하면 필요한 에지 보다 더 많이 검출
- T_{high}를 사용하면 중요한 외곽선에 속하는 에지를 정의
- T_{low}를 사용한 에지가 T_{high}에 연결되어 있으면 에지, 그렇지 않으면 제거
- Canny recommended a *upper:lower* ratio between 2:1 and 3:1.

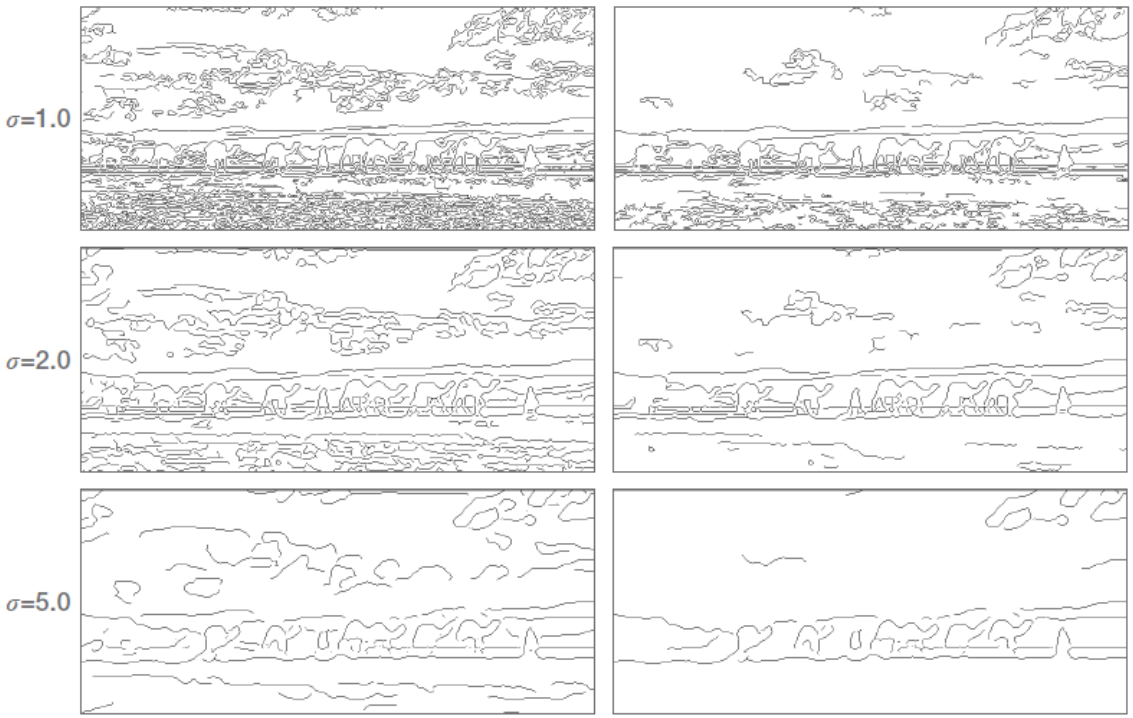
4.2 캐니 에지



A는 maxVal 보다 위에 있으므로 확실한 에지입니다. B와 C는 minVal, maxVal 사이에 있는 픽셀입니다. 여기서 B는 확실한 에지와 연결되어 있지 않지만, C는 확실한 에지와 연결이 되어 있네요~ 따라서 B는 에지가 아니므로 제거하고 C는 에지이므로 남겨둡니다.

4.2 캐니 에지

(a) 원래 영상(342×800)



(b) 낮은 임계값 (c) 높은 임계값

그림 3–18 캐니 에지

- σ=2.0 일때
- 임계값: (high, low) 2.5배
- 왼쪽 임계값=(0.125, 0.05)
- 오른쪽 임계값=(0.25, 0.1)
- σ 가 커질 수록 디테일이 사라진다.

4.2 캐니 에지

프로그램 4-2 캐니 에지 실험하기

```
01  import cv2 as cv
02
03  img=cv.imread('soccer.jpg')          # 영상 읽기
04
05  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
06
07  canny1=cv.Canny(gray,50,150)         # Tlow=50, Thigh=150으로 설정
08  canny2=cv.Canny(gray,100,200)        # Tlow=100, Thigh=200으로 설정
09
10  cv.imshow('Original',gray)
11  cv.imshow('Canny1',canny1)
12  cv.imshow('Canny2',canny2)
13
14  cv.waitKey()
15  cv.destroyAllWindows()
```

4.2 캐니 에지

물체 경계와 그림자 에지를 구별하지 못하는 한계
사람은 물체의 3차원 모델과 겉모습 모델appearance model을
사용하여 의미적으로 검출

- void Canny(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false)
  - **image** – single-channel 8-bit input image.
  - **edges** – output edge map; it has the same size and type as image .
  - **threshold1** – first threshold for the hysteresis procedure.(에지 강도맵상에서의 임계값)
  - **threshold2** – second threshold for the hysteresis procedure.
  - **apertureSize** – aperture size for the Sobel() operator.
  - **L2gradient** – a flag, indicating whether a more accurate L2-norm should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default L1-norm is enough (L2gradient=false).

Python:

cv.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]]) -> edges

cv.Canny(dx, dy, threshold1, threshold2[, edges[, L2gradient]]) -> edges

dx	16-bit x derivative of input image (CV_16SC1 or CV_16SC3).
dy	16-bit y derivative of input image (same type as dx).

과제

- food.jpg 영상을 사용하여 다음을 구하여 보시오
  - (y,x)=(30,40)에서 dy, dx,에지 강도, 에지 방향을 구하여라.
  - sobel 에지(에지 강도 맵)를 구하여라.
- sigmaX=sigmaY=1(3x3 필터)로 가우시안 스므딩 한 후 Canny 에지를 구하라.

4.3 직선 검출

- 에지 맵(1: 에지 화소, 0: 에지가 아닌 화소) → 이웃한 에지 화소를 연결하여 경계선으로 변환 → 경계선을 직선으로 근사하여 변환
- 에지를 명시적으로 연결하여 경계선을 찾고 직선으로 변환
  - 이후 처리 단계인 물체 표현이나 인식에 유리

4.3.1 경계선 찾기

4.3.2 허프 변환

4.3.3 RANSAC

4.3.1 경계선 찾기

■ 8-연결된 에지 화소를 연결해 경계선$_{contour}$ 구성

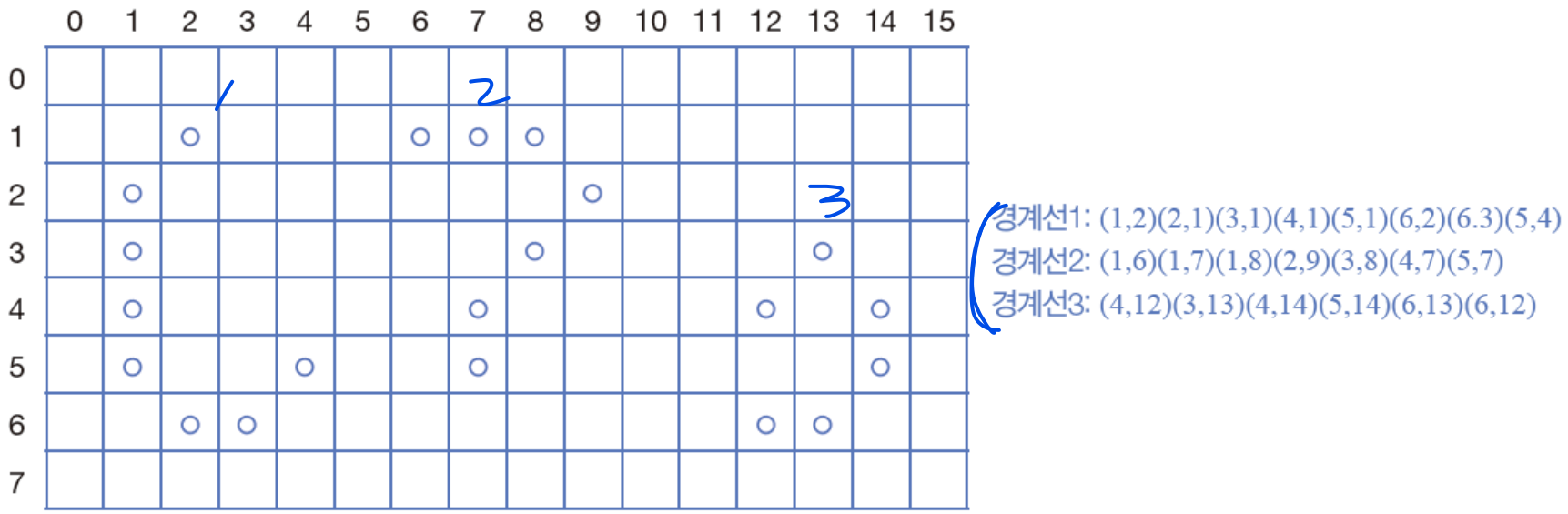


그림 4-9 에지 맵에서 경계선 찾기

■ findContours 함수 이용

4.3.1 경계선 찾기

프로그램 4-3 에지 맵에서 경계선 찾기

```
01  import cv2 as cv
02  import numpy as np
03
04  img=cv.imread('soccer.jpg')              # 영상 읽기
05  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
06  canny=cv.Canny(gray,100,200)
07
08  contour,hierarchy=cv.findContours(canny,cv.RETR_LIST,cv.CHAIN_APPROX_NONE)
09
10  lcontour=[]
11  for i in range(len(contour)):
12      if contour[i].shape[0]>100:          # 길이가 100보다 크면
13          lcontour.append(contour[i])
14
15  cv.drawContours(img,lcontour,-1,(0,255,0),3)
16
17  cv.imshow('Original with contours',img)
18  cv.imshow('Canny',canny)
19
20  cv.waitKey()
21  cv.destroyAllWindows()
```

이 매개변수를 통해
여러 가지 근사 방법 제공

시작점으로 돌아올 때까지 추적하므로
실제로는 50보다 크면

성분의 외곽선 추출

- 영상 분석 목표: 객체를 식별하고 추출
  - 첫 번째 단계가 이진 맵 생성
  - 두번 째 이진 영상에서 연결 성분 추출(connected component)

- void findContours(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())
  - **image** – Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . The function modifies the image while extracting the contours. If mode equals to CV_RETR_CCOMP or CV_RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1).
  - **contours** – Detected contours. Each contour is stored as a vector of points.
  - **hierarchy** – Optional output vector, containing information about the image topology.
    - vector<Vec4i> 자료형의 윤곽선 계층구조에 관한 출력벡터, hierarchy[i][j], j=0,1,2,3, ith contour. For each i-th contour contours[i], the elements hierarchy[i][0] , hierarchy[i][1] , hierarchy[i][2] , and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative.
  - **mode** – Contour retrieval mode
    - **RETR_EXTERNAL** retrieves only the extreme outer contours. It sets hierarchy[i][2]=hierarchy[i][3]=-1 for all the contours.
    - **RETR_LIST** retrieves all of the contours without establishing any hierarchical relationships.
    - **RETR_CCOMP** retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
    - **RETR_TREE** retrieves all of the contours and reconstructs a full hierarchy of nested contours.

Python:

```
cv.findContours( image, mode, method[, contours[, hierarchy[, offset]]] ) -> contours, hierarchy
```

성분의 외곽선 추출

- **method** – Contour approximation method
  - **CHAIN_APPROX_NONE** stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, max(abs(x1-x2),abs(y2-y1))==1.
  - **CHAIN_APPROX_SIMPLE** compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
  - **CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS** applies one of the flavors of the Teh-Chin chain approximation algorithm. See [TehChin89] for details.
- **offset** – Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

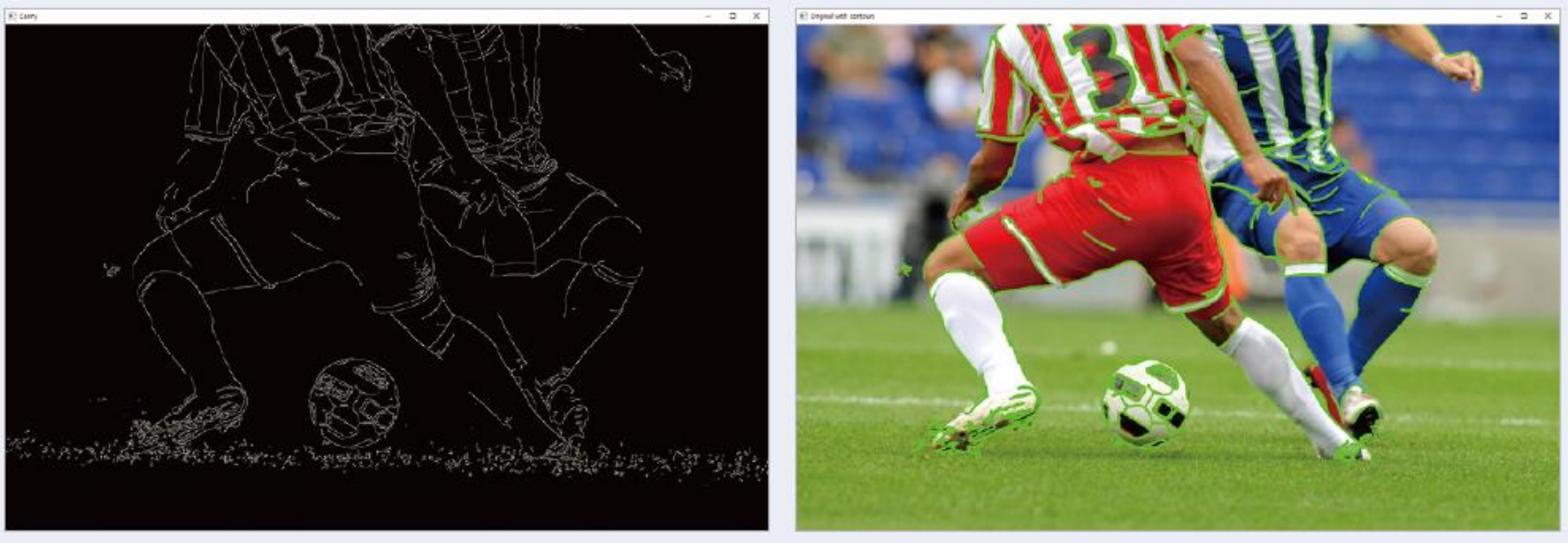
성분의 외곽선 그리기

- void drawContours(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point())
  - **image** – Destination image.
  - **contours** – All the input contours. Each contour is stored as a point vector.
  - **contourIdx** – Parameter indicating a contour to draw. If it is negative, all the contours are drawn.
  - **color** – Color of the contours.
  - **thickness** – Thickness of lines the contours are drawn with.
  - **lineType** – Line connectivity. **8** (or omitted) - 8-connected line. **4** - 4-connected line. **CV_AA** - antialiased line
  - **hierarchy** – Optional information about hierarchy. It is only needed if you want to draw only some of the contours (see maxLevel).
  - **maxLevel** – Maximal level for drawn contours. If it is 0, only the specified contour is drawn. If it is 1, the function draws the contour(s) and all the nested contours. If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested contours, and so on. This parameter is only taken into account when there is hierarchy available.

Python:

```
cv.drawContours( image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]] ) -> image
```

4.3.1 경계선 찾기



4.3.2 허프 변환

- 허프 변환 **y = a x + b => b = - x a + y**
  - 에지 연결 과정 없이 선분 검출 (전역 연산을 이용한 지각 군집화)
  - 영상 공간 *y-x*를 기울기 절편 공간 *b-a*로 매핑

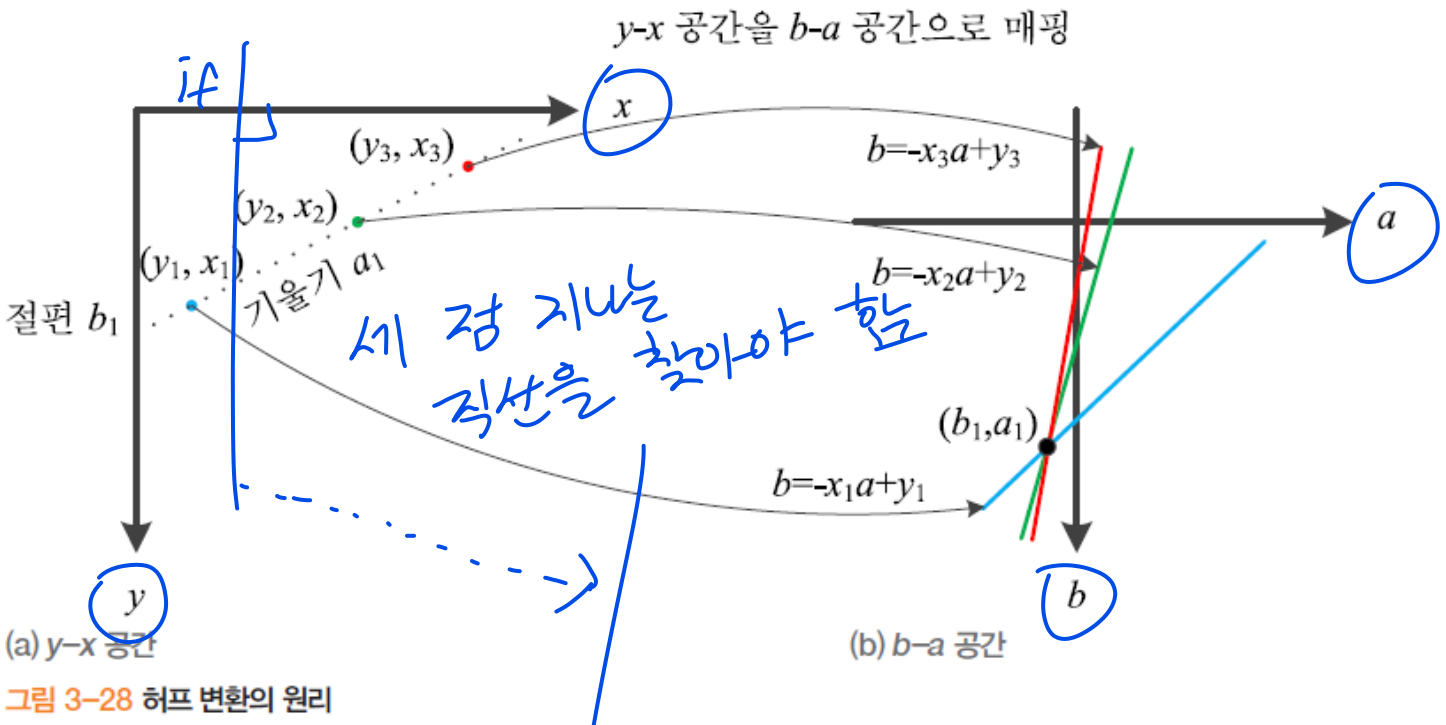


그림 3–28 허프 변환의 원리

4.3.2 허프 변환

- 수직선의 기울기가 ∞인 문제
  - 극좌표계 사용하여 해결

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

$$y = -\frac{s}{c}x + \frac{\rho}{c}$$

$$y = ax + b \quad (3.16)$$

y-x 공간을 ρ-θ 공간으로 매핑

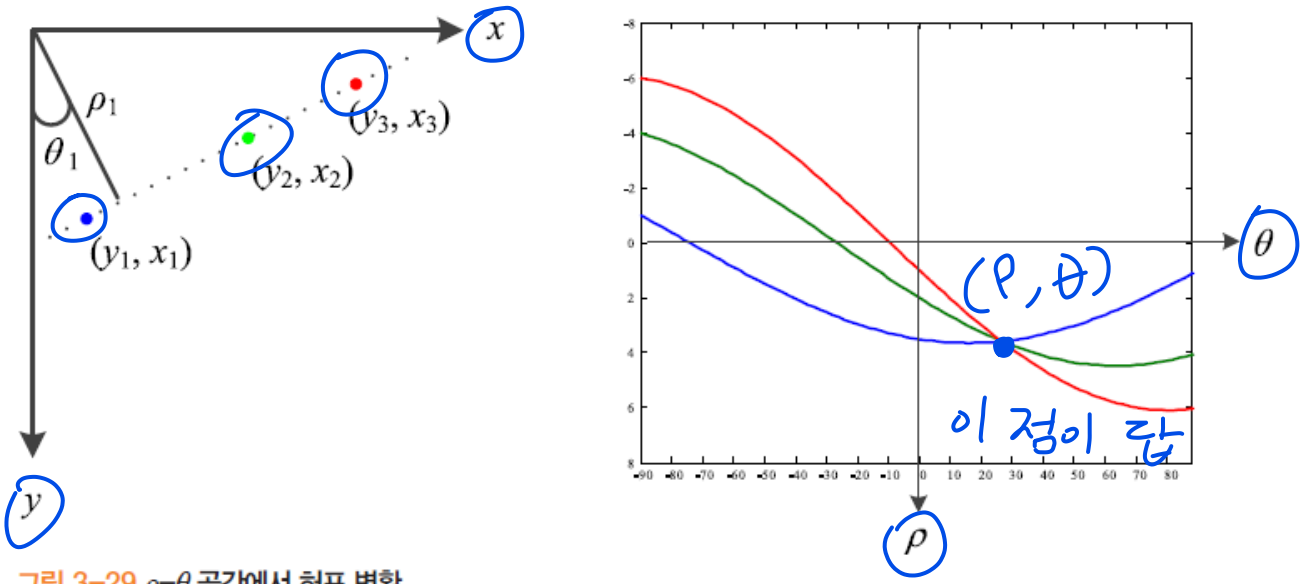


그림 3-29 ρ-θ 공간에서 허프 변환

4.3.2 허프 변환

밀집된 곳 찾기

- 양자화된 누적 배열 이용하여 해결

알고리즘 3-7 직선 검출을 위한 허프 변환

입력 : 에지 영상 $e(j,i)$, $0 \le j \le M-1$, $0 \le i \le N-1$, 임계값 T // 에지는 1, 비에지는 0인 이진 영상

출력 : (ρ_k, θ_k), $1 \le k \le n$(n개의 직선)

1. 2차원 누적 배열 A를 0으로 초기화한다.
2. for(에지 영상 e에 있는 에지 화소 (y_i, x_i) 각각에 대해)
3. $\quad y_i\cos\theta + x_i\sin\theta = \rho$가 지나는 A의 모든 칸을 1만큼 증가시킨다.
4. A에서 T를 넘는 지역 최대점 (ρ_k, θ_k)를 모두 찾아 직선으로 취한다.

원 검출

- 3차원 누적 배열 사용

$$(x-a)^2+(y-b)^2 = r^2 \qquad (4.7)$$

4.3.2 허프 변환

예제 3-3 **허프 변환**

[그림 3-30]은 [그림 3-29]를 이산 공간에 다시 그린 것이다. 왼쪽 그림에서 세 점은 $(y_1, x_1)=(4,1)$, $(y_2, x_2)=(2,4)$, $(y_3, x_3)=(1,6)$이다. $(y_1, x_1)=(3.5,1)$이면 세 점이 정확히 일직선 상에 있지만, 디지털 영상의 특성상 약간의 위치 오차가 발생했다고 간주하자.

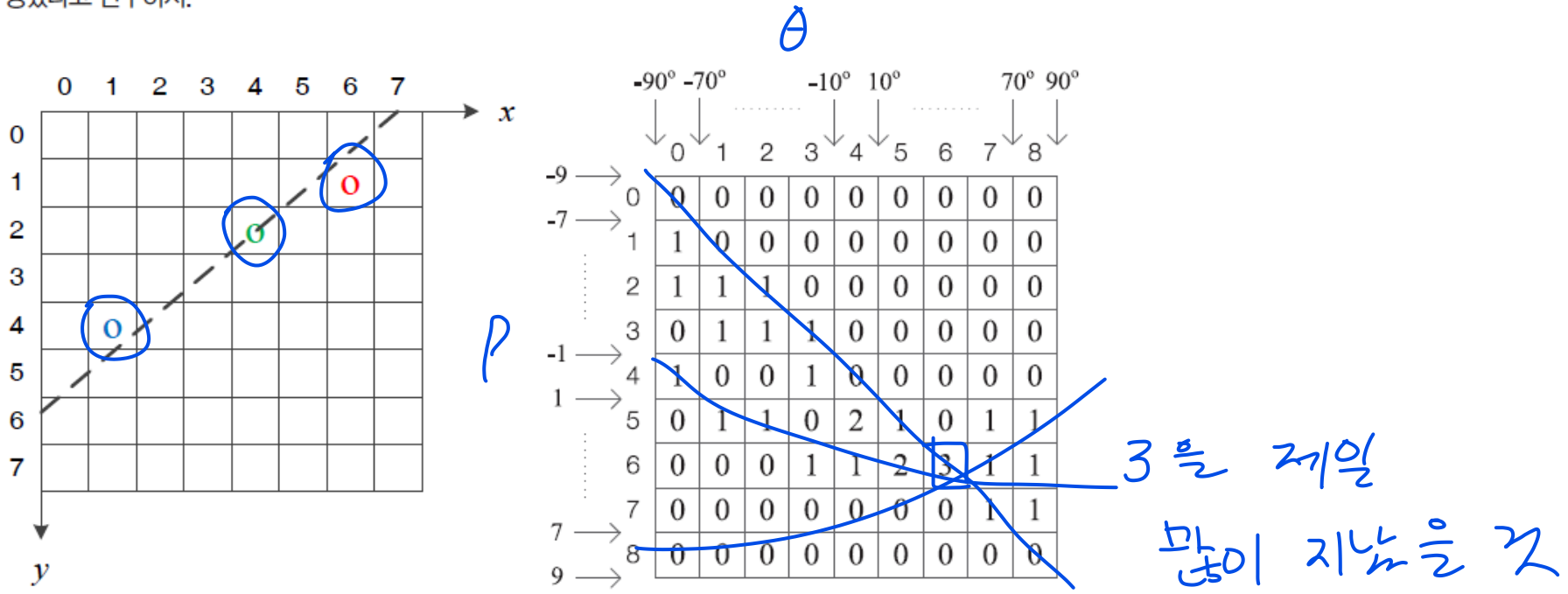


그림 3-30 이산 공간에서 허프 변환

θ축은 20° 간격으로 양자화하여 총 아홉 개의 구간을 가지도록 하였다. ρ축은 범위 $[-9, 9]$를 2 크기의 구간으로 나누어 총 아홉 개의 구간을 가지도록 양자화하였다. 따라서 누적 배열 A는 9×9이다. [알고리즘 3-7]에 따라 A를 0으로 초기화한 후, 2~3행을 수행하여 세 점의 자취를 누적시키면 오른쪽 그림과 같은 배열이 된다. 이 배열에서 지역 최대점은 3을 갖는 (6,6)으로, $(\rho, \theta)=(4, 40°)$에 해당한다. $y \cos 40° + x \sin 40° = 4$라는 직선을 검출한 셈이다. 왼쪽 그림에 있는 점선이 검출한 직선이다.

4.3.2 허프 변환

프로그램 4-4 허프 변환을 이용해 사과 검출하기

```
01  import cv2 as cv
02
03  img=cv.imread('apples.jpg')
04  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
05
06  apples=cv.HoughCircles(gray,cv.HOUGH_GRADIENT,1,200,param1=150,param2=20,
                           minRadius=50,maxRadius=120)
07
08  for i in apples[0]:
09      cv.circle(img,(int(i[0]),int(i[1])),int(i[2]),(255,0,0),2)
10
11  cv.imshow('Apple detection',img)
12
13  cv.waitKey()
14  cv.destroyAllWindows()
```

영상에서 원 검출

- void HoughCircles(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**=100, double **param2**=100, int **minRadius**=0, int **maxRadius**=0)
  - **image** – 8-bit, single-channel, grayscale input image.
  - **circles** – Output vector of found circles. Each vector is encoded as a 3-element floating-point vector .
  - **method** – Detection method to use. Currently, the only implemented method is CV_HOUGH_GRADIENT , which is basically *21HT* , described in [Yuen90].
  - **dp** – Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height.
  - **minDist** – Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
  - **param1** – First method-specific parameter. In case of HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny() edge detector (the lower one is twice smaller).
  - **param2** – Second method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
  - **minRadius** – Minimum circle radius.
  - **maxRadius** – Maximum circle radius.
- 잘못된 원 검출을 유발할 수 있는 영상 잡음을 줄이기 위해 영상을 부드럽게
- 원 검출 결과는 cv::Vec3f 인스턴스 벡타에 저장: (x,y,r) 중심벡터와 반지름
- $(x-a)^2+(y-b)^2=r^2$ => 축척배열이 (a, b, r) 3차원

```
cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]]) → circles
```

허프변환 함수

- void HoughLines(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **srn**=0, double **stn**=0)
  - **image** – 8-bit, single-channel binary source image. The image may be modified by the function.: 선 형태로 정렬된 점 집합(캐니 연산자로 얻을 영상도 가능)
  - **lines** – Output vector of lines. Each line is represented by a two-element vector (ρ, θ) . ρ is the distance from the coordinate origin (top-left corner of the image). θ is the line rotation angle in radians ( ). 즉 (ρ, θ)를 원소로 갖는 벡터
  - **rho** – Distance resolution of the accumulator in pixels.
  - **theta** – Angle resolution of the accumulator in radians.
  - **threshold** – Accumulator threshold parameter. Only those lines are returned that get enough votes ( ).
  - **srn** – For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The coarse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive.
  - **stn** – For the multi-scale Hough transform, it is a divisor for the distance resolution theta.
  - 각 선의 끝 점이 주어지지 않기 때문에 선을 검출하지만 선분은 검출하지 않음에 유의

Python:

```
cv.HoughLines( image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]] ) -> lines
```

확률적 허프 변환

- 선분 검출을 허용(끝점이 있는 직선 검출)
- void HoughLinesP(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **minLineLength**=0, double **maxLineGap**=0)
  - **image** – 8-bit, single-channel binary source image. The image may be modified by the function.
  - **lines** – Output vector of lines. Each line is represented by a 4-element vector , where and are the ending points of each detected line segment.
  - **rho** – Distance resolution of the accumulator in pixels.
  - **theta** – Angle resolution of the accumulator in radians.
  - **threshold** – Accumulator threshold parameter. Only those lines are returned that get enough votes ( ).
  - **minLineLength** – Minimum line length. Line segments shorter than that are rejected.
  - **maxLineGap** – Maximum allowed gap between points on the same line to link them.

Python:

```
cv.HoughLinesP( image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]] ) -> lines
```

```
import cv2 as cv
import numpy as np
import sys
import math
img=cv.imread('road.jpg')
h, w = img.shape[:2]
if img is None:
    sys.exit(' No file')
img_original=img.copy()
gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
edges=cv.Canny(gray, 125, 350)
lines=cv.HoughLines(edges,1, math.pi/180, 90)
for i in range(len(lines)):
    for rho, theta in lines[i]:
        tx, ty = np.cos(theta), np.sin(theta)
        x0, y0 = tx*rho, ty*rho
        x1 = int(x0 + w*(-ty))
        y1 = int(y0 + w*tx)
        x2 = int(x0 - w*(-ty))
        y2 = int(y0 - w*tx)
        cv.line(img, (x1,y1), (x2,y2), (0,0,255), 2)
 res=np.vstack((img_original, img)); cv.imshow('IMG', res)
cv.waitKey(); cv.destroyAllWindows()
```

```
#%% HoughLinesP

img=cv.imread('road.jpg')

if img is None:
    sys.exit(' No file')

img_original=img.copy()
gray=cv.cvtColor(img, cv.COLOR_BGR2GRAY)
edges=cv.Canny(gray, 125, 350)
lines=cv.HoughLinesP(edges,1, math.pi/360, 90, 100, 10)
print(len(lines))
for i in range(len(lines)):
    for x1, y1, x2, y2 in lines[i]:
        cv.line(img, (x1, y1), (x2,y2), (0,0,255), 2)

cv.imshow('HoughLinesP', img)
cv.waitKey()
cv.destroyAllWindows()
```

실습: 다음 영상에서 직선을 찾아보시오

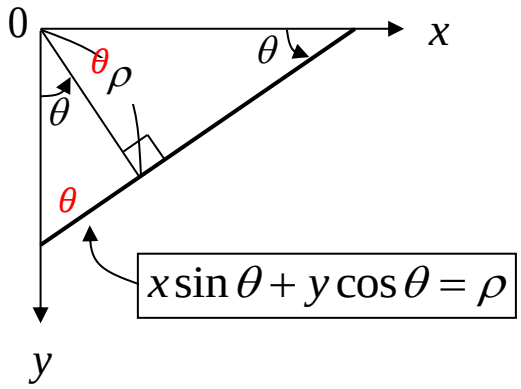
```
img=skimage.data.checkerboard()
print(img.shape)

img_original=img.copy()
img=cv.cvtColor(img, cv.COLOR_GRAY2BGR)
```

영상에서 허프 변환으로 선 검출

- 직선의 방정식 $y = ax + b$ 를 사용할 때의 문제점
  - y 축과 평행한 수직선을 표현하지 못함(a: 무한대)
  - 극좌표계 형태의 직선의 방정식을 사용

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



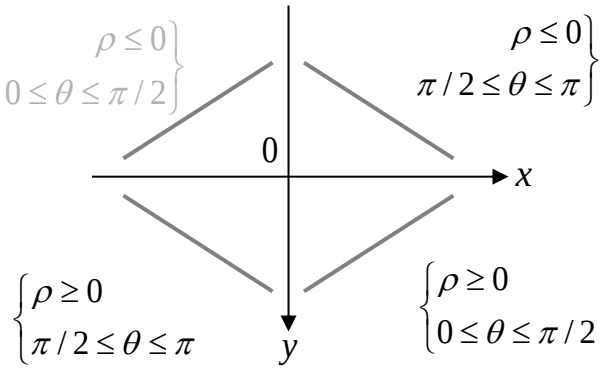
$$-\sqrt{M^2 + N^2} \le \rho \le \sqrt{M^2 + N^2}$$

$$0 \le \theta \le \pi$$

M과 N은 영상의 가로 및 세로의 픽셀크기

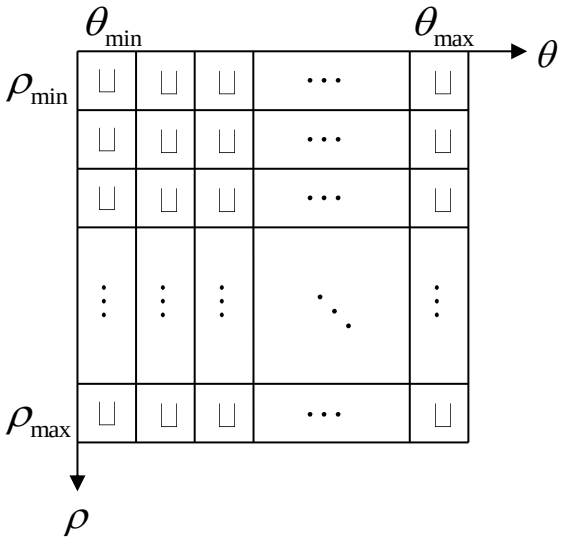
(ρ, θ) 공간에서의 축척배열(accumulator)

- 직선의 방향에 따른 $\rho\theta$ 파라미터 값의 범위

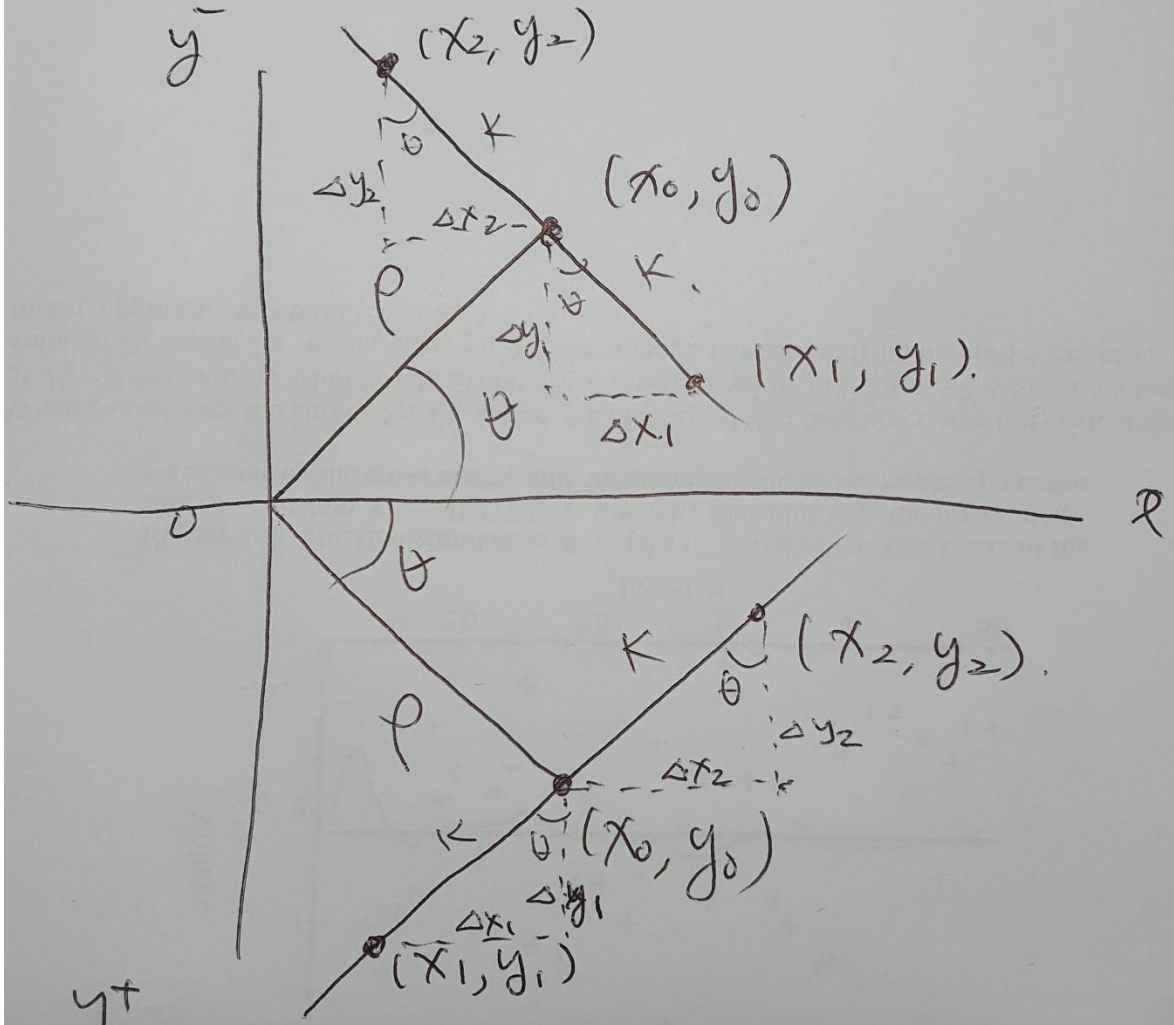


$$x\cos\theta + y\sin\theta = \rho$$

- 파라미터 공간에서 축적 배열 구성(양자화)



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



$$(x_0, y_0) = (\rho\cos\theta,\ \rho\sin\theta)$$

$(x_0, y_0) = (\rho \cos\theta, \rho \sin\theta)$

$\theta > 0$ $\quad x_1 = x_0 - \Delta x_1 = x_0 - K * \sin\theta$

$= x_0 + K * \sin(-\theta)$

$y_1 = x_0 + \Delta y_1 = y_0 + K \cos\theta$

$x_2 = x_0 + \Delta x_2 = x_0 + K \sin\theta$

$= x_0 - K \sin(-\theta)$

$y_2 = y_0 - \Delta y_2 = y_2 - K * \cos\theta$

$\theta < 0$ $\quad x_1 = x_0 + \Delta x_1 = x_0 + K \sin(-\theta)$

$y_1 = y_0 - \Delta y_1 = y_0 + K \cdot \cos(-\theta)$

$= y_0 + K \cos\theta$

$x_2 = x_0 - \Delta x_2 = x_0 - K * \sin(-\theta)$

$y_2 = y_0 + \Delta y_2 = y_0 - K \cos(-\theta)$

$= y_0 - K \cos(\theta)$

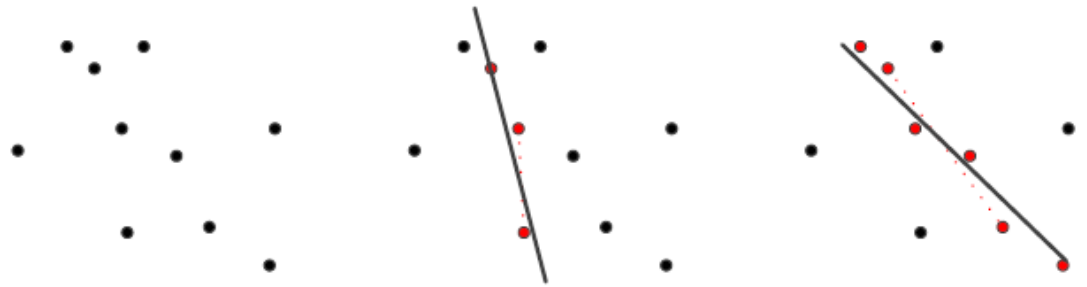
4.3.3 RANSAC

- RANSAC(Random Sample Consensus)
  - 1981년 Fischler&Bolles이 제안 [Fischler81]
  - 인라이어를 찾아 어떤 모델을 적합시키는 기법
  - 난수 생성하여 인라이어 군집을 찾기 때문에 임의성 지님

임의의 점지나는 직선 군처의 점

- 선분 검출에 적용
  - 모델은 직선의 방정식 $y=ax+b$

인라이어가 많고, 인라이어를 지나는 직선들의 2차가 적으면 good



(a) 입력 샘플 집합 (b) 1차 시도 (c) 2차 시도

그림 3-31 RANSAC의 원리

4.3.3 RANSAC

알고리즘 3-8 직선 검출을 위한 RANSAC

입력 : 에지 영상 $e(j,i)$, $0 \le j \le M-1$, $0 \le i \le N-1$ // 에지는 1, 비에지는 0인 이진 영상
반복 횟수 n, 인라이어 판단 t, 인라이어 집합의 크기 d, 직선 적합 오차 e

출력 : 하나의 직선(기울기 a와 절편 b)

```
1   line=Ø;
2   for(loop=1 to n) {
3     에지 화소 두 개를 임의로 선택한다.
4     이 두 점으로 직선의 방정식 l을 계산한다.
5     이 두 점으로 집합 inlier를 초기화한다.
6     for(이 두 점을 제외한 모든 에지 화소 p에 대해)
7       if(p가 직선 l에 허용 오차 t 이내로 적합) p를 inlier에 넣는다.
8     if(|inlier|≥d){   // 집합 inlier가 d개 이상의 샘플을 가지면
9       inlier에 있는 모든 샘플을 가지고 직선의 방정식 l을 새로 계산한다.
10      if(l의 적합 오차 < e) l을 집합 line에 넣는다.
11    }
12  }
13  line에 있는 직선 중 가장 좋은 것을 취한다.
```