

CHAPTER 09

에지 검출과 응용



OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝

컴퓨터 비전 기초부터 딥러닝 활용까지!

9장 에지 검출과 응용

9.1 에지 검출

9.2 직선 검출과 원 검출

9.1 에지 검출

1. 에지 검출

➤ 미분과 그래디언트

- 영상에서 에지(edge)는 한쪽 방향으로 픽셀 값이 급격하게 바뀌는 부분을 가리킴
- 어두운 영역에서 갑자기 밝아지거나 또는 반대로 밝은 영역에서 급격하게 어두워지는 부분을 에지라고 함
- 일반적으로 객체와 배경의 경계, 또는 객체와 다른 객체의 경계에서 에지가 발생함
- 영상에서 에지를 찾아내는 작업은 객체의 윤곽을 알아낼 수 있는 유용한 방법임
- 다양한 컴퓨터 비전 시스템에서 객체 판별을 위한 전처리로 에지 검출이 사용되고 있음

1. 에지 검출

➤ 미분과 그래디언트

- 기본적으로 영상에서 에지를 찾아내려면 픽셀 값의 변화율을 측정하여 변화율이 큰 픽셀을 선택해야 함
- 수학에서 함수 또는 데이터의 변화율을 **미분**(derivative)이라고 함
- 좀 더 정확하게 기술하면 함수의 미분이란 주어진 함수의 순간 변화율을 의미함
- 1차원 연속 함수 $f(x)$ 의 미분은 다음과 같이 정의함

$$f' = \frac{df}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

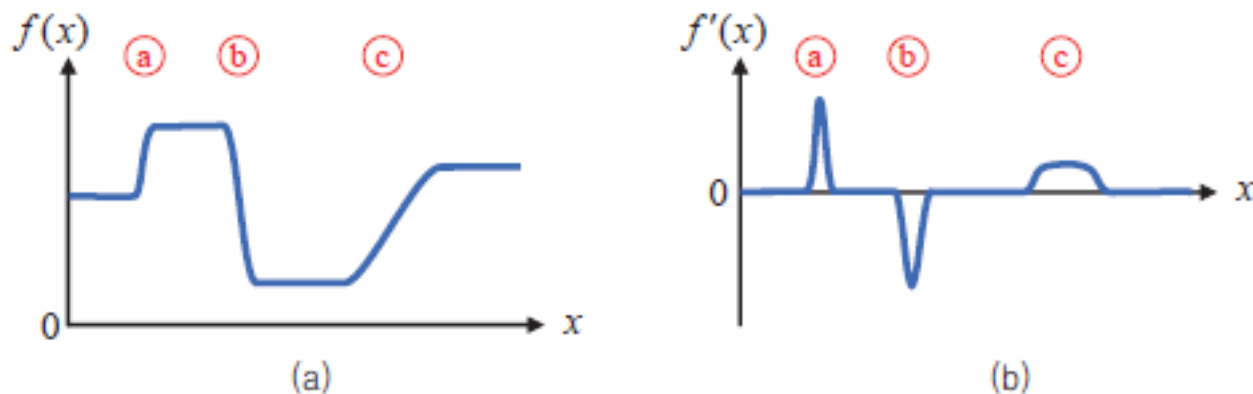
- 앞 수식에서 Δx 는 x 의 변화량을 의미함
- x 의 변화량이 무한히 0에 가까워질 때의 함수 값 변화량을 미분이라고 함
- 함수 값이 증가하는 위치에서는 함수의 미분 값이 0보다 큰 양수로 나타남
- 함수 값이 감소하는 위치에서는 함수의 미분 값이 0보다 작은 음수를 갖게 됨
- 함수 값이 일정한 구간에서는 함수의 미분이 0에 가까운 값을 가짐

1. 에지 검출

➤ 미분과 그래디언트


- 그림 9-1(a)에서 함수 그래프가 수평으로 진행되는 부분은 함수 값 변화가 없는 부분이고, 이 위치에서의 $f'(x)$ 값은 0임
- 함수 $f(x)$ 값이 급격하게 증가하는 ① 위치에서는 $f'(x)$ 값이 0보다 큰 값을 가짐
- $f(x)$ 값이 급격하게 감소하는 ② 위치에서는 $f'(x)$ 값이 0보다 훨씬 작은 음수로 나타남
- ③ 위치에서는 $f(x)$ 값이 대체로 일정한 기울기로 증가하므로 $f'(x)$ 값은 0보다 큰 양수 값이 일정하게 나타남
- 함수 $f(x)$ 값이 급격하게 바뀌는 부분을 찾기 위해서는 함수의 미분 $f'(x)$ 값이 0보다 훨씬 크거나 또는 훨씬 작은 위치를 찾아야 함

▼ 그림 9-1 1차원 연속 함수의 미분



1. 에지 검출

➤ 미분과 그래디언트

- 잘 알려진 다항함수나 삼각함수 등의 미분을 계산하는 것은 이미 수학의 미적분학에서 많이 연구 되었기 때문에 공식을 통해 쉽게 구할 수 있음
- 영상은 2차원 평면 위에 픽셀 값이 정형화되지 않은 상태로 나열되어 있는 형태이므로 미분 공식을 적용할 수 없음
- 영상으로부터 미분을 계산하려면 두 가지 특징을 고려해야 함
- 하나는 영상이 2차원 평면에서 정의된 함수라는 점임 
- 두 번째는 영상이 정수 단위 좌표에 픽셀이 나열되어 있는 이산함수라는 점임



1. 에지 검출

➤ 미분과 그래디언트

- 영상의 미분을 곧바로 설명하기에 앞서 먼저 1차원 이산함수에서 미분을 구하는 방법에 대해 알아보자
- 영상과 같이 일련의 데이터가 순서대로 나열되어 있는 경우에는 미분 근사화 방법을 이용하여 변화량을 측정할 수 있음
- 미분 근사는 다음 세 가지 방법을 주로 사용함

- 전진 차분(forward difference): $\frac{dI}{dx} \cong \frac{I(x+h) - I(x)}{h}$

- 후진 차분(backward difference): $\frac{dI}{dx} \cong \frac{I(x) - I(x-h)}{h}$

- 중앙 차분(centered difference): $\frac{dI}{dx} \cong \frac{I(x+h) - I(x-h)}{2h}$

1. 에지 검출

➤ 미분과 그래디언트

- 앞 수식에서 $I(x)$ 는 1차원 이산함수이고, h 는 이산 값의 간격을 의미함
- 미분 근사 방법을 영상에 적용할 경우, h 는 픽셀의 간격이라고 생각할 수 있으며 보통 픽셀 간격의 최소 단위인 1을 h 값으로 사용함
- 전진 차분 수식은 $I(x+1) - I(x)$ 로 정리되며, 이는 자기 자신 바로 앞에 있는 픽셀에서 자기 자신 픽셀 값을 뺀 형태임
- 후진 차분은 $I(x) - I(x-1)$ 로 정리되며, 이는 자기 자신 픽셀에서 바로 뒤에 있는 픽셀 값을 뺀 형태임
- 마지막으로 중앙 차분은 $(I(x+1) - I(x-1)) / 2$ 수식으로 정리되며, 자기 자신을 제외하고 바로 앞과 뒤에 있는 픽셀 값을 이용하는 미분 근사 방법임
- 세 가지 미분 근사 방법 중에서 중간값 차이를 이용하는 방법이 이론적으로 근사화 오류가 가장 적으며, 실제 영상에서 미분을 계산할 때에도 널리 사용되고 있음

1. 에지 검출

➤ 미분과 그래디언트

- 영상은 2차원 평면에서 정의된 함수이기 때문에 영상에서 에지를 찾기 위해서는 영상을 가로 방향과 세로 방향으로 각각 미분해야 함
- 2차원 영상 $I(x, y)$ 를 가로 방향으로 미분한다는 것은 y 좌표는 고정된 상태에서 x 축 방향으로만 미분 근사를 계산하는 것을 의미하며, 이러한 연산을 x 축 방향으로의 편미분(partial derivative)이라고 함
- x 축 방향의 편미분은 I_x 또는 $\frac{\partial I}{\partial x}$ 로 표기함
- 이와 유사하게 y 축 방향으로의 편미분은 I_y 또는 $\frac{\partial I}{\partial y}$ 라고 표기하고, x 좌표를 고정된 상태에서 y 축 방향으로 미분 근사를 수행하여 구할 수 있음
- 2차원 영상 $I(x, y)$ 에 대하여 x 축과 y 축 방향에 대한 각각의 편미분을 중앙 차분 방법으로 근사화하면 다음과 같음

$$I_x = \frac{\partial I}{\partial x} \cong \frac{I(x+1, y) - I(x-1, y)}{2}$$

$$I_y = \frac{\partial I}{\partial y} \cong \frac{I(x, y+1) - I(x, y-1)}{2}$$

1. 에지 검출

➤ 미분과 그래디언트

- 그림 9-2(a)는 영상을 x축 방향으로 편미분을 수행하는 1×3 필터 마스크임
- 그림 9-2(b)는 영상을 y축 방향으로 편미분하는 3×1 필터 마스크임
- 앞서 설명한 편미분 근사 수식을 그대로 적용하려면 필터 마스크 값에 1/2을 곱해야 하지만 보통 미분 값의 상대적 크기를 중요시하기 때문에 그림 9-2와 같이 단순화시킨 마스크를 주로 사용함
- 그림 9-2의 마스크를 이용하여 영상을 각각 필터링하면 영상을 가로 방향과 세로 방향으로 편미분한 정보를 담고 있는 행렬을 얻을 수 있음

▼ 그림 9-2 중앙 차분에 의한 미분 근사 마스크

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

(a)

-1
0
1

(b)

1. 에지 검출

➤ 미분과 그래디언트

- 원래 영상의 미분은 부호가 있는 실수로 계산되지만 그림 9-3에서는 미분 결과를 시각적으로 분석하기 위하여 미분 값에 128을 더함
- 0부터 255 사이의 정수로 형변환하여 그레이스케일 영상 형태로 나타냄
- 그림 9-3(b)와 그림 9-3(c)에서 그레이스케일 값 128에 해당하는 회색 영역은 입력 영상에서 픽셀 값 변화가 작은 영역임
- 흰색 또는 검은색으로 표현된 픽셀은 입력 영상에서 픽셀 값이 급격하게 바뀌는 부분임
- 그림 9-3(b)에서 **흰색**으로 표현된 부분은 x 좌표가 **증가**함에 따라 픽셀 값이 급격하게 커지는 위치임
- **검은색**으로 표현된 부분은 픽셀 값이 급격하게 **감소**하는 위치임
- 그림 9-3(c)에서는 y 좌표가 증가함에 따른 픽셀 값이 크게 증가하거나 감소하는 부분이 흰색 또는 검은색으로 나타남

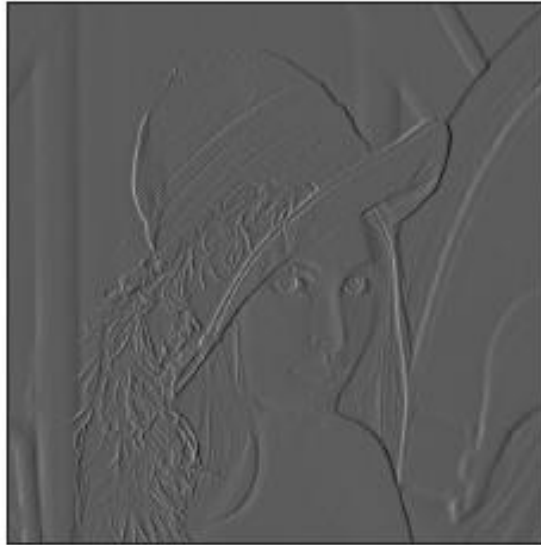
1. 에지 검출

➤ 미분과 그래디언트

▼ 그림 9-3 레나 영상의 편미분 결과



(a)



(b)



(c)

1. 에지 검출

➤ 미분과 그래디언트

- 2차원 공간에서 정의된 영상에서 에지를 찾으려면 **x축 방향과 y축 방향의 편미분을 모두 사용**해야 함
- 2차원 공간에서 정의된 함수 $f(x, y)$ 가 있을 때 이 함수의 **x축 방향 미분과 y축 방향 미분을 한꺼번에 벡터로 표현한 것을 그래디언트(gradient)**라고 하고 다음과 같이 표기함

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

- 그래디언트는 벡터이기 때문에 크기(magnitude)와 방향(phase) 성분으로 표현할 수 있음
- 그래디언트 벡터의 방향은 변화 정도가 가장 큰 방향을 나타냄
- 그래디언트 벡터의 크기는 변화율 세기를 나타내는 척도로 생각할 수 있음
- **그래디언트 크기**는 보통 $\|\nabla f\|$ 로 표기하고, 다음과 같이 구함

$$\sqrt{f_x^2 + f_y^2}$$

1. 에지 검출

➤ 미분과 그래디언트

- 그래디언트 방향 θ 는 다음 수식으로 구할 수 있음

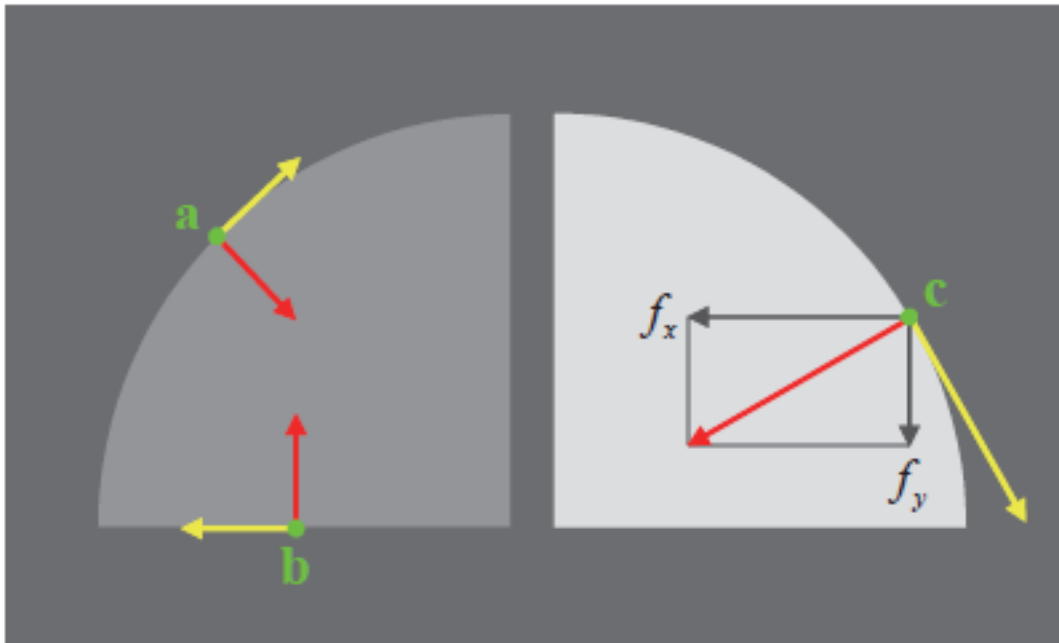
$$\theta = \tan^{-1} \left(\frac{f_y}{f_x} \right)$$

- 그림 9-4에 나타난 영상은 어두운 배경에 밝기가 다른 두 개의 객체가 있는 영상임
- 빨간색 화살표의 길이는 그래디언트 크기를 나타내고, 화살표 방향은 그래디언트 벡터의 방향을 나타냄
- 그래디언트 벡터의 크기는 밝기 차이가 클수록 크게 나타나므로 점 a, b의 화살표보다 점 c에서 화살표 길이가 더 길게 나타남
- 그래디언트 벡터의 방향은 해당 위치에서 밝기가 가장 밝아지는 방향을 가리킴
- 점 c에 대해서는 특별히 x축 방향으로의 편미분 f_x 와 y축 방향으로의 편미분 f_y 성분을 함께 표시하였으며, 이 두 성분을 이용하여 빨간색 화살표를 그릴 수 있음
- 참고로 그림 9-4에서 노란색으로 표시된 화살표는 그래디언트 벡터와 수직인 방향을 표시한 것이며, 이를 에지의 방향이라고 부름

1. 에지 검출

➤ 미분과 그래디언트

▼ 그림 9-4 그래디언트 크기와 방향



1. 에지 검출

➤ 미분과 그래디언트

- 2차원 영상에서 에지를 찾는 기본적인 방법은 그래디언트 크기가 특정 값보다 큰 위치를 찾는 것임
- 여기서 에지 여부를 판단하기 위해 기준이 되는 값을 임계값(threshold) 또는 문턱치라고 부름
- 임계값은 영상의 특성에 따라 다르게 설정해야 하며, 보통 사용자의 경험에 의해 결정 됨
- 일반적으로 임계값을 높게 설정하면 밝기 차이가 급격하게 변하는 에지 픽셀만 검출됨
- 임계값을 낮게 설정하면 약한 에지 성분도 검출됨

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- 그림 9-5(a)는 x축 방향으로의 편미분을 구하는 소벨 마스크임
- 그림 9-5(b)는 y축 방향으로의 편미분을 구하는 소벨 마스크임
- 그림 9-5(a)에 나타난 x축 방향 미분 마스크는 현재 행에 대한 중앙 차분 연산을 2회 수행함
- 이전 행과 다음 행에 대해서도 중앙 차분 연산을 1회씩 수행함
- 연산은 현재 행과 이웃 행에서의 픽셀 값 변화가 유사하다는 점을 이용하여 잡음의 영향을 줄이기 위함
- 특히 현재 행에서 두 번의 중앙 차분 연산을 수행하는 것은 현재 행의 중앙 차분 근사에 더 큰 가중치를 주기 위함
- y축 방향 미분을 계산하는 그림 9-5(b) 마스크도 같은 방식으로 설계됨

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

▼ 그림 9-5 3×3 소벨 마스크

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

(a)

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



(b)

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- OpenCV는 소벨 마스크를 이용하여 영상을 미분하는 `Sobel()` 함수를 제공함
- `Sobel()` 함수는 3×3 소벨 마스크 또는 확장된 형태의 큰 마스크를 이용하여 영상을 미분함

```
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);
```

• <code>src</code>	입력 영상
• <code>dst</code>	출력 영상. <code>src</code> 와 같은 크기, 같은 채널 수를 갖습니다.
• <code>ddepth</code>	출력 영상의 깊이
• <code>dx</code>	x 방향 미분 차수 
• <code>dy</code>	y 방향 미분 차수
• <code>ksize</code>	소벨 커널의 크기
• <code>scale</code>	필터링 연산 후 추가적으로 곱할 값
• <code>delta</code>	필터링 연산 후 추가적으로 더할 값
• <code>borderType</code>	가장자리 픽셀 확장 방식 

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- Sobel() 함수는 입력 영상 `src`를 편미분한 결과를 `dst`에 저장함
- 결과 `영상의 자료형`은 `ddepth` 인자를 통해 명시적으로 지정해야 함
- `ddepth`에 -1을 지정하면 `src`와 같은 타입을 사용하는 `dst` 영상을 생성함
- `dx`와 `dy` 인자는 각각 `x 방향`과 `y 방향`으로의 `편미분 차수`를 의미함
- Sobel() 함수에 의해 계산되는 결과 행렬 `dst`는 다음 수식과 같은 의미를 가짐

$$dst = \frac{\partial^{xorder+yorder} src}{\partial^{xorder} \partial^{yorder}}$$

- `ksize` 이후의 인자는 모두 기본값을 가지고 있으므로 실제 함수 호출 시에는 생략할 수 있음
- `ksize` 인자에 1을 지정하면 3×1 또는 1×3 커널을 사용함
- 기본값인 3을 지정하면 3×3 소벨 마스크를 사용함

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- Sobel() 함수는 x 방향과 y 방향으로의 고차 미분을 계산할 수 있지만 대부분의 경우 x 방향 또는 y 방향으로의 1차 미분을 구하는 용도로 사용됨
- 예를 들어 그레이스케일 레나 영상을 x 방향으로 편미분한 결과를 dx 행렬에, y 방향으로 편미분한 결과를 dy 행렬에 저장하려면 다음과 같이 코드를 작성함

```
Mat src = imread("lenna.bmp", IMREAD_GRAYSCALE);
```

```
Mat dx, dy;
```

```
Sobel(src, dx, CV_32FC1, 1, 0);
```

```
Sobel(src, dy, CV_32FC1, 0, 1);
```

- 앞 예제 코드에서 dx와 dy 행렬 크기는 src 행렬과 같고, 행렬 원소는 float 자료형을 사용하도록 설정함

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- OpenCV는 널리 사용되고 있는 소벨 마스크 외에도 **샤르 필터(Scharr filter)** 마스크를 이용한 미분연산도 지원함
- 샤르 필터는 3×3 소벨 마스크보다 정확한 미분 계산을 수행하는 것으로 알려져 있음

▼ 그림 9-6 3×3 샤르 필터 마스크

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

(a)

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

(b)

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- 샤프 필터 마스크를 이용하여 영상을 미분하려면 Scharr() 함수를 사용함 

```
void Scharr(InputArray src, OutputArray dst, int ddepth,  
            int dx, int dy, double scale = 1, double delta = 0,  
            int borderType = BORDER_DEFAULT);
```

- | | |
|--------------|-----------------------------------|
| • src | 입력 영상 |
| • dst | 출력 영상. src와 같은 크기, 같은 채널 수를 갖습니다. |
| • ddepth | 출력 영상의 깊이 |
| • dx | x 방향 미분 차수 |
| • dy | y 방향 미분 차수 |
| • scale | 필터링 연산 후 추가적으로 곱할 값 |
| • delta | 필터링 연산 후 추가적으로 더할 값 |
| • borderType | 가장자리 픽셀 확장 방식 |

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- 샤프 필터를 이용한 영상의 미분은 앞서 설명한 Sobel() 함수를 이용하여 구할 수도 있음
- Sobel() 함수의 ksize 인자에 FILTER_SCHARR 또는 -1을 지정하면 3×3 샤프 마스크를 사용하여 영상을 미분함
- Sobel() 또는 Scharr() 함수를 이용하여 x 방향으로 미분과 y 방향으로 미분을 각각 계산하여 행렬에 저장한 후, 두 미분 행렬을 이용하여 그래디언트 크기를 계산할 수 있음
- OpenCV는 2차원 벡터의 x 방향 좌표와 y 방향 좌표를 이용하여 벡터의 크기를 계산하는 magnitude() 함수를 제공함

```
void magnitude(InputArray x, InputArray y, OutputArray magnitude);
```

- x 벡터의 x 좌표를 나타내는 실수 행렬 또는 벡터
- y 벡터의 y 좌표를 나타내는 실수 행렬 또는 벡터. x와 크기와 타입이 같아야 합니다.
- magnitude 벡터의 크기를 나타내는 실수 행렬 또는 벡터. x와 같은 크기, 같은 타입을 갖습니다.

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- `magnitude()` 함수의 입력으로 사용되는 `x`와 `y`는 `CV_32F` 또는 `CV_64F` 깊이를 사용하는 행렬 또는 벡터이어야 함
- `magnitude()` 함수의 출력 `magnitude`를 구성하는 원소 값은 다음 수식에 의해 계산됨

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

- 만약 `x` 방향으로 미분과 `y` 방향으로 미분이 저장된 두 개의 행렬이 있을 때, **그라디언트의 방향**을 계산하고 싶다면 `phase()` 함수를 사용할 수 있음

```
void phase(InputArray x, InputArray y, OutputArray angle, bool angleInDegrees = false);
```

- `x` 벡터의 `x` 좌표를 나타내는 실수 행렬 또는 벡터
- `y` 벡터의 `y` 좌표를 나타내는 실수 행렬 또는 벡터. `x`와 크기와 타입이 같아야 합니다.
- `angle` 벡터의 방향을 나타내는 실수 행렬 또는 벡터. `x`와 같은 크기, 같은 타입을 갖습니다.
- `angleInDegrees` 이 값이 `true`이면 각도(degree) 단위를 사용하고, `false`이면 라디안(radian) 단위를 사용합니다.

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- `phase()` 함수에서 `x`와 `y`는 입력이고, `angle`은 출력임
- `angle`의 각 원소는 다음 수식에 의해 계산됨

$$\text{angle}(I) = \text{atan2}\left(\frac{y(I)}{x(I)}\right)$$

- 코드 9-1에 나타난 `sobel_edge()` 함수는 레나 영상에 대해 `x`축 방향과 `y`축 방향의 1차 미분을 구하고, 그래디언트 크기가 특정 임계 값보다 큰 픽셀을 에지로 검출함

코드 9-1 소벨 마스크 기반 에지 검출 예제 [ch09/edges]

```
01 void sobel_edge()
02 {
03     Mat src = imread("lenna.bmp", IMREAD_GRAYSCALE);
04
05     if (src.empty()) {
06         cerr << "Image load failed!" << endl;
07         return;
08     }
09 }
```

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

코드 9-1 소벨 마스크 기반 에지 검출 예제 [ch09/edges]

```
10     Mat dx, dy;
11     Sobel(src, dx, CV_32FC1, 1, 0);
12     Sobel(src, dy, CV_32FC1, 0, 1);
13
14     Mat fmag, mag;
15     magnitude(dx, dy, fmag);
16     fmag.convertTo(mag, CV_8UC1);
17
18     Mat edge = mag > 150;
19
20     imshow("src", src);
21     imshow("mag", mag);
22     imshow("edge", edge);
23
24     waitKey();
25     destroyAllWindows();
26 }
```

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

- 10~12행 x축 방향으로 1차 편미분, y축 방향으로 1차 편미분을 각각 구하여 dx와 dy 행렬에 저장합니다. dx와 dy 행렬은 float 자료형을 사용하도록 설정하였습니다.
- 15행 dx와 dy 행렬로부터 그래디언트 크기를 계산하여 fmag에 저장합니다. dx와 dy가 모두 float 자료형을 사용하므로 fmag도 float 자료형을 사용하는 행렬로 생성됩니다.
- 16행 실수형 행렬 fmag를 그레이스케일 형식으로 변환하여 mag에 저장합니다.
- 18행 에지 판별을 위한 그래디언트 크기 임계값을 150으로 설정하여 에지를 판별합니다. 행렬 edge의 원소 값은 mag 행렬 원소 값이 150보다 크면 255로, 작으면 0으로 설정됩니다.

2. 마스크 기반 에지 검출

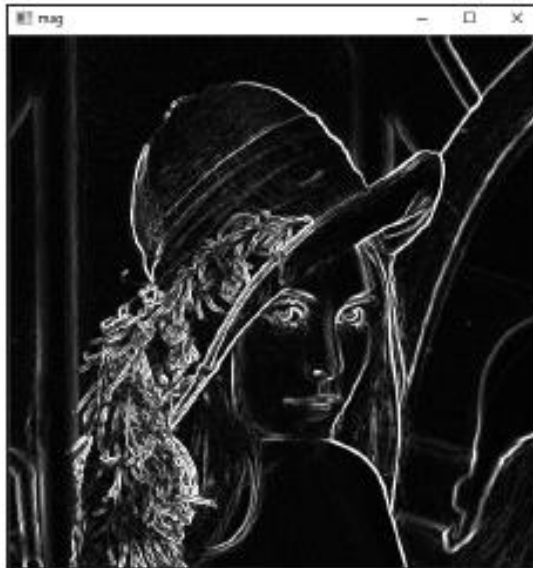
➤ 마스크 기반 에지 검출

- 그림 9-7에서 src는 입력으로 사용한 lenna.bmp 영상이고, mag 영상은 그래디언트 크기를 그레이스케일 영상 형식으로 나타냄
- 만약 각 픽셀에서 계산된 그래디언트 크기가 255보다 큰 경우에는 포화 연산에 의해 흰색으로 표현됨
- 그림 9-7에서 edge 영상은 그래디언트 크기가 150보다 큰 픽셀은 흰색으로, 그렇지 않은 픽셀은 검은색으로 표현된 이진 영상임
- 만약 코드 9-1의 18행에서 사용된 임계값을 150보다 낮게 설정하면 더 많은 에지 픽셀이 edge 영상에 나타나게 됨
- 다만 임계값을 너무 낮추면 잡음의 영향도 에지로 검출될 수 있으므로 주의해야 함

2. 마스크 기반 에지 검출

➤ 마스크 기반 에지 검출

▼ 그림 9-7 소벨 마스크 기반 에지 검출 실행 결과



3. 캐니 에지 검출기

➤ 캐니 에지 검출기

- 1986년 캐니(J. Canny)는 에지 검출을 최적화 문제 관점으로 접근함으로써 소벨 에지 검출 방법의 단점을 해결할 수 있는 방법을 제시함[Canny86]
- 캐니는 자신의 논문에서 다음 세가지 항목을 좋은 에지 검출기의 조건으로 제시함
 1. **정확한 검출**(good detection): 에지를 검출하지 못하거나 또는 에지가 아닌데 에지로 검출하는 확률을 최소화해야 합니다.
 2. **정확한 위치**(good localization): 실제 에지의 중심을 찾아야 합니다.
 3. **단일 에지**(single edge): 하나의 에지는 하나의 점으로 표현되어야 합니다.

3. 캐니 에지 검출기

➤ 캐니 에지 검출기

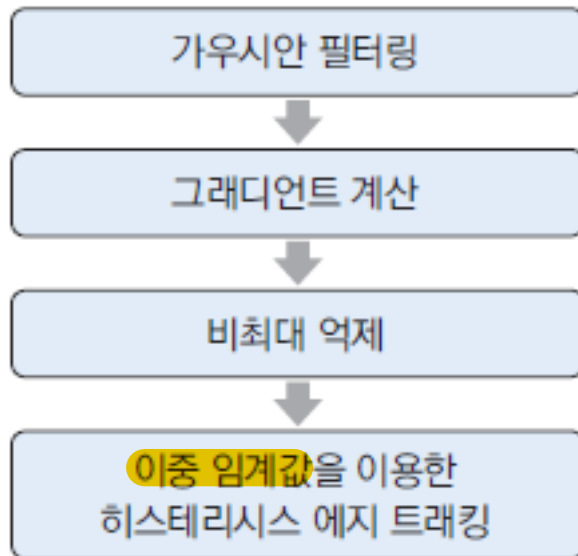
- 캐니는 이러한 조건을 만족하는 새로운 형태의 에지 검출 방법을 제시하였으며, 이를 캐니 에지 검출기(canny edge detector)라고 함
- 캐니 에지 검출기는 그래디언트의 크기와 방향을 모두 고려하여 좀 더 정확한 에지 위치를 찾을 수 있음
- 에지는 서로 연결되어 있는 가능성이 높다는 점을 고려하여 그래디언트 크기가 다소 약하게 나타나는 에지도 놓치지 않고 찾을 수 있음

3. 캐니 에지 검출기

➤ 캐니 에지 검출기

- 캐니 에지 검출기는 내부적으로 크게 네 개의 연산 과정을 포함

▼ 그림 9-8 캐니 에지 검출기 수행 과정



3. 캐니 에지 검출기

➤ 가우시안 필터링

- 캐니 에지 검출기의 첫 번째 과정은 가우시안 필터링임
- 캐니 에지 검출기의 첫 번째 단계에서 가우시안 필터를 적용하는 이유는 영상에 포함된 잡음을 제거하기 위함
- 다만 가우시안 필터링에 의해 영상이 부드러워지면서 에지의 세기도 함께 감소할 수 있기 때문에 적절한 표준 편차를 선택하여 가우시안 필터링을 수행해야 함
- 영상에 포함된 잡음이 심하지 않다면 가우시안 필터링은 생략할 수 있음

➤ 그래디언트 계산

- 캐니 에지 검출기의 두 번째 과정은 영상의 그래디언트를 구하는 작업임
- 캐니 에지 검출기에서 그래디언트 계산은 보통 3×3 소벨 마스크를 사용함
- 캐니 에지 검출기는 좀 더 정확한 에지를 찾기 위해 그래디언트 방향도 함께 고려함
- 가로 방향과 세로 방향으로 각각 소벨 마스크 필터링을 수행한 후, 그래디언트 크기와 방향을 모두 계산해야 함

3. 캐니 에지 검출기

➤ 그래디언트 계산

- 2차원 공간에서 정의된 함수 $f(x, y)$ 의 그래디언트를 $\nabla f = f_x \mathbf{i} + f_y \mathbf{j}$ 라고 할 경우, 그래디언트 크기는 $\|\nabla f\| = \sqrt{f_x^2 + f_y^2}$ 정의되고 이를 벡터 ∇f 의 L2 노름(L2 norm)이라고 함
- 그래디언트 크기를 실제로 계산할 때에는 연산 속도 향상을 위해 그래디언트 크기를 $\|\nabla f\| \approx |f_x| + |f_y|$ 형태로 계산하기도 하며, 이를 벡터 ∇f 의 L1 노름(L1 norm)이라고 함
- 실제로 OpenCV에 구현되어 있는 캐니 에지 검출기에서도 그래디언트 크기 계산 시 기본적으로 L1노름을 사용함

3. 캐니 에지 검출기

➤ 비최대 억제

- 에지 검출을 위해 단순히 그래디언트 크기가 특정 임계값보다 큰 픽셀을 선택할 경우, 에지 근방의 여러 픽셀이 한꺼번에 에지로 선택될 수 있음
- 에지가 두껍게 표현되는 현상을 방지하기 위해 캐니 에지 검출기에서는 비최대 억제(non-maximum suppression) 과정을 사용함
- 비최대 억제는 그래디언트 크기가 국지적 최대(local maximum)인 픽셀만을 에지 픽셀로 설정하는 기법임
- 상대적으로 국지적 최대가 아닌 픽셀은 에지 픽셀에서 제외하기 때문에 비최대 억제라는 용어를 사용함
- 일반적인 2차원 영상에서 국지적 최대를 찾으려면 특정 픽셀을 둘러싸고 있는 모든 픽셀 값을 검사하여 국지적 최대인지를 판별해야 함
- 캐니 에지 검출기의 비최대 억제 과정에서는 그래디언트 벡터의 방향과 같은 방향에 있는 인접 픽셀끼리만 국지적 최대 검사를 수행함
- 결과적으로 비최대 억제를 수행함으로써 가장 변화율이 큰 위치의 픽셀만 에지로 검색됨

3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

- 하나의 임계값을 사용할 경우 이분법으로 결과가 판단되기 때문에 환경 변화에 민감해질 수 있음
- 이러한 문제를 보완하기 위해 캐니 에지 검출기에서는 두 개의 임계값을 사용함
- 캐니 에지 검출기에서 사용하는 두 개의 임계값 중에서 높은 임계값을 T_{High} , 낮은 임계값을 T_{Low} 라고 표기함
- 만약 그래디언트 크기가 T_{High} 보다 크면 이 픽셀은 최종적으로 에지로 판단함
- 그래디언트 크기가 T_{Low} 보다 작으면 에지 픽셀이 아니라고 판단함
- 그래디언트 크기가 T_{Low} 와 T_{High} 사이인 픽셀은 에지일 수도 있고 에지가 아닐 수도 있다고 판단함
- 이런 픽셀에 대해서는 추가적인 검사를 수행함

3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

- 설명의 편의를 위해 그래디언트 크기가 T_{High} 보다 큰 픽셀을 강한 에지(strong edge)라고 표현함
- 그래디언트 크기가 T_{Low} 와 T_{High} 사이인 픽셀은 약한 에지(weak edge)라고 표현함
- 캐니에지 검출기의 마지막 단계에서는 히스테리시스 에지 트래킹(hysteresis edge tracking) 방법을 사용하여 약한 에지 중에서 최종적으로 에지로 판별할 픽셀을 선택함
- 히스테리시스 에지 트래킹 방법은 에지 픽셀이 대체로 상호 연결되어 있다는 점을 이용함
- 만약 약한 에지 픽셀이 강한 에지 픽셀과 서로 연결되어 있다면 이 픽셀은 최종적으로 에지로 판단함
- 반면에 강한 에지와 연결되어 있지 않은 약한 에지 픽셀은 최종적으로 에지가 아니라고 판단함

3. 캐니 에지 검출기

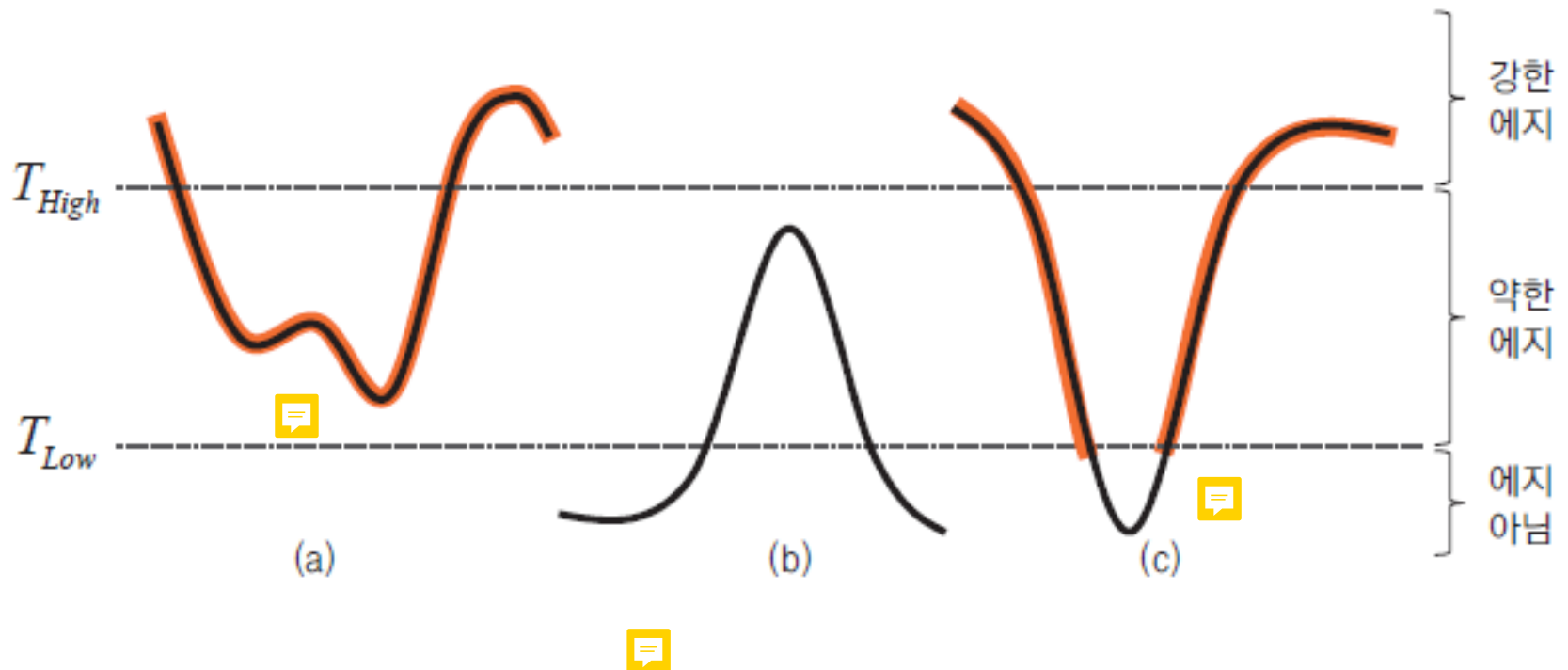
➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

- 그림 9-9는 서로 연결되어 있는 에지 픽셀의 그래디언트 크기를 표현한 그림임
- 세 가지 형태의 에지 후보에 대하여 최종적으로 에지로 선택되는 픽셀들은 두꺼운 주황색으로 표시함
- 그림 9-9(a)는 약한 에지 픽셀이 모두 강한 에지와 연결되어 있으므로 모든 픽셀이 최종적으로 에지로 결정됨
- 그림 9-9(b)에서는 약한 에지 픽셀이 강한 에지와 연결되어 있지 않으므로 최종 에지로 선택되지 않음
- 그림 9-9(c)에 나타난 약한 에지는 강한 에지와 연결되어 있으므로 최종적으로 에지로 판별하지만 그래디언트 크기가 T_{Low} 보다 작은 픽셀은 에지로 판별하지 않음

3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

▼ 그림 9-9 이중 임계값을 이용한 히스테리시스 에지 트래킹



3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

- OpenCV에서 캐니 에지 검출 알고리즘은 Canny() 함수에 구현되어 있음

```
void Canny(InputArray image, OutputArray edges,
           double threshold1, double threshold2,
           int apertureSize = 3, bool L2gradient = false);
void Canny(InputArray dx, InputArray dy, OutputArray edges,
           double threshold1, double threshold2,
           bool L2gradient = false);
```

- **image** 8비트 입력 영상
- **dx** 입력 영상의 x 방향 미분 영상. CV_16SC1 또는 CV_16SC3
- **dy** 입력 영상의 y 방향 미분 영상. CV_16SC1 또는 CV_16SC3
- **edges** 출력 에지 영상. 입력 영상과 크기가 같은 8비트 단일 채널 영상입니다.
- **threshold1** 히스테리시스 에지 검출을 위한 첫 번째 임계값
- **threshold2** 히스테리시스 에지 검출을 위한 두 번째 임계값
- **apertureSize** 그래디언트 계산을 위한 소벨 마스크 크기
- **L2gradient** 그래디언트 크기 계산 시 L2 노름을 사용하려면 true를 지정합니다. 이 값이 false이면 L1 노름을 사용합니다.

3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

- 코드 9-2에 나타난 `canny_edge()` 함수는 두 가지 임계값 쌍을 이용하여 캐니 에지 검출을 수행하고 그 결과를 각각 화면에 출력함

코드 9-2 캐니 에지 검출 예제 [ch09/edges]

```

01  void canny_edge()
02  {
03      Mat src = imread("lenna.bmp", IMREAD_GRAYSCALE);
04
05      if (src.empty()) {
06          cerr << "Image load failed!" << endl;
07          return;
08      }
09
10      Mat dst1, dst2;
11      Canny(src, dst1, 50, 100);
12      Canny(src, dst2, 50, 150);
13

```

3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

코드 9-2 캐니 에지 검출 예제 [ch09/edges]

```
14     imshow("src", src);
15     imshow("dst1", dst1);
16     imshow("dst2", dst2);
17
18     waitKey();
19     destroyAllWindows();
20 }
```

- 11행 낮은 임계값을 50, 높은 임계값을 100으로 설정하여 캐니 에지 검출을 수행하고 그 결과를 dst1에 저장합니다.
- 12행 낮은 임계값을 50, 높은 임계값을 150으로 설정하여 캐니 에지 검출을 수행하고 그 결과를 dst2에 저장합니다.

3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

- 그림 9-10에서 src 창은 입력 영상인 lenna.bmp 파일이고, dst1과 dst2 창은 서로 다른 임계값을 사용하여 구한 캐니 에지 검출 결과 영상임
- dst1과 dst2의 에지 영상을 구할 때 낮은 임계값은 50으로 동일하게 설정하였고 높은 임계값은 각각 100과 150으로 설정함
- 임계값을 낮출수록 에지로 판별되는 픽셀이 더 많아지므로 dst1 영상에 더 많은 에지 픽셀이 검출된 것을 확인할 수 있음
- 다만 임계값을 낮출수록 잡음에 해당하는 픽셀도 에지로 검출할 가능성이 높아질 수 있으므로 주의해야 함

3. 캐니 에지 검출기

➤ 이중 임계값을 이용한 히스테리시스 에지 트래킹

▼ 그림 9-10 캐니 에지 검출 예제 실행 결과



9.2 직선 검출과 원 검출

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 영상에서 직선 성분을 찾기 위해서는 우선 에지를 찾아내고, 에지 픽셀들이 일직선상에 배열되어 있는지를 확인해야 함
- 영상에서 직선을 찾기 위한 용도로 허프 변환(hough transform) 기법이 널리 사용됨
- 허프 변환은 2차원 xy 좌표에서 직선의 방정식을 파라미터(parameter) 공간으로 변환하여 직선을 찾는 알고리즘임
- 일반적인 2차원 평면에서 직선의 방정식은 다음과 같이 나타낼 수 있음

$$y = ax + b$$

- 이 수식에서 a는 기울기(slope)이고, b는 y 절편(y intersection)임
- 이 직선의 방정식은 가로축이 x, 세로축이 y인 2차원 xy 좌표 공간에서 정의되어 있음
- a와 b는 직선의 형태를 결정하는 파라미터임

$$b = -xa + y$$

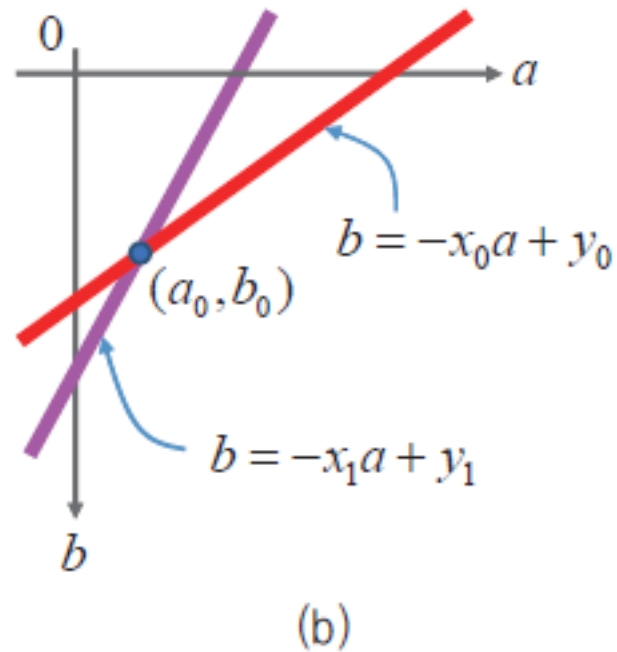
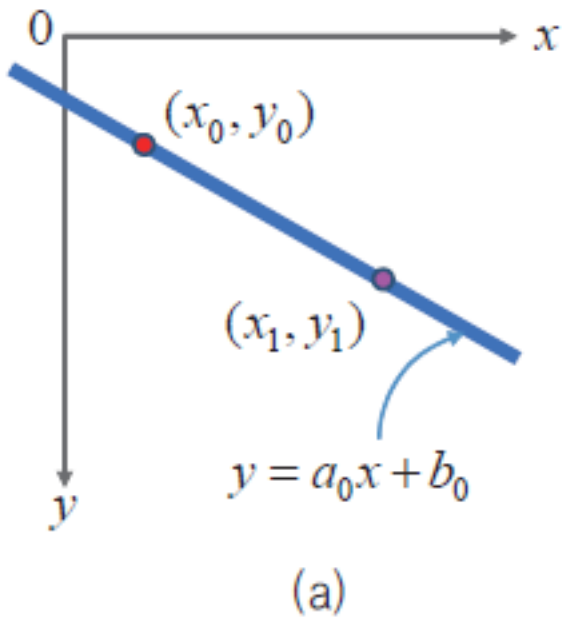


1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- xy 좌표 공간과 ab 좌표 공간과의 관계를 제대로 이해하기 위해 그림 9-11을 살펴보자

▼ 그림 9-11 xy 공간에서 ab 파라미터 공간으로 허프 변환



1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

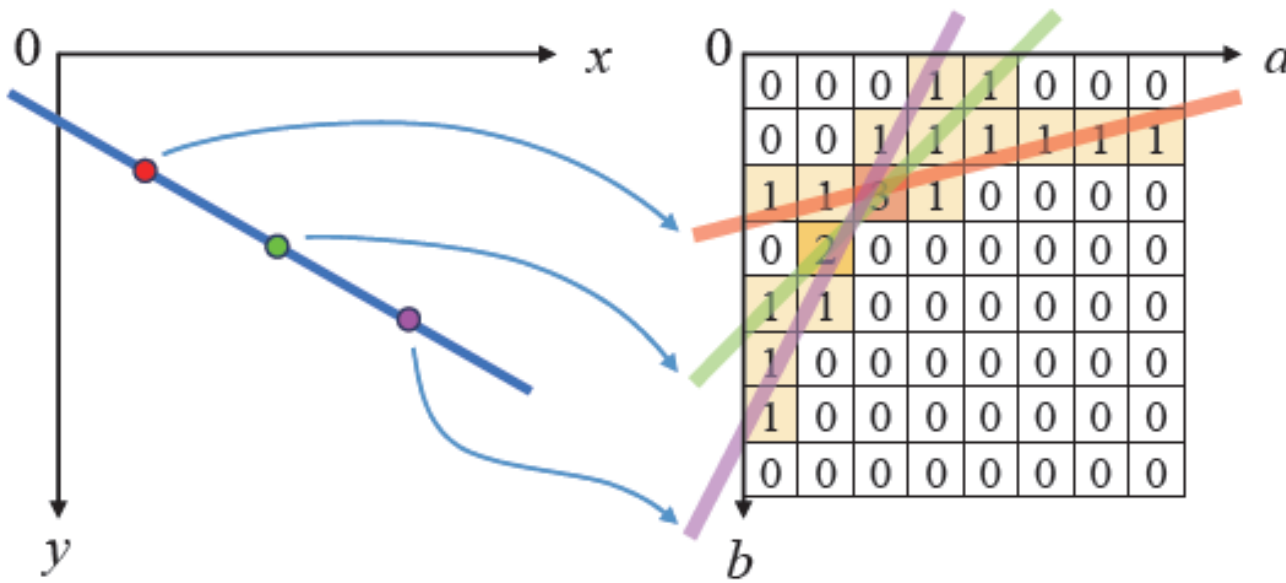
- 허프 변환을 이용하여 직선의 방정식을 찾으려면 xy 공간에서 에지로 판별된 모든 점을 이용하여 ab 파라미터 공간에 직선을 표현함
- 직선이 많이 교차되는 좌표를 모두 찾아야 함
- 이때 직선이 많이 교차하는 점을 찾기 위해서 보통 축적 배열(accumulation array)을 사용함
- 축적 배열은 0으로 초기화된 2차원 배열에서 직선이 지나가는 위치의 배열 원소 값을 1씩 증가시켜 생성함

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 그림 9-12에서는 하나의 직선에 대해 허프 변환 예를 설명하였으며, 여러 개의 직선이 존재하는 영상이라면 축적 배열에서 여러 개의 국지적 최댓값 위치를 찾아서 직선의 방정식 파라미터를 결정할 수 있음


▼ 그림 9-12 2차원 영상 공간에서 파라미터 공간으로 변환



1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- $y = ax + b$ 형태의 직선의 방정식을 사용할 경우 모든 형태의 직선을 표현하기 어렵다는 단점이 있음
- 대표적으로 $y = ax + b$ 수식은 y 축과 평행한 수직선을 표현할 수 없음
- 수직선을 표현하려면 기울기 a 값이 무한대가 되어야 하기 때문임
- 실제 허프 변환을 구현할 때에는 다음과 같이 **극좌표계** 형식의 직선의 방정식을 사용함

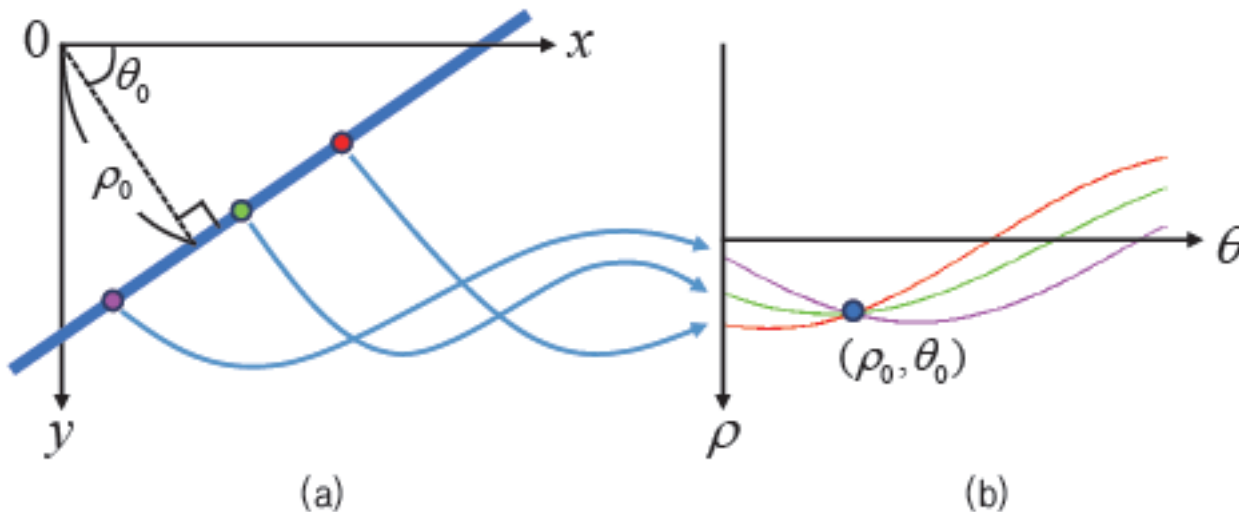
$$x \cos \theta + y \sin \theta = \rho$$

- 앞 수식에서 ρ 는 원점에서 직선까지의 수직 거리를 나타냄
- θ 는 원점에서 직선에 수직선을 내렸을 때 x 축과 이루는 각도를 의미함
- 이 경우 xy 공간에서 한 점은 공간에서는 삼각함수 그래프 형태의 곡선으로 표현됨
- 공간에서 한 점은 xy 공간에서 직선으로 나타나게 됨
- 극좌표계 형식의 직선의 방정식을 사용하여 허프 변환을 수행할 경우에도 축적 배열을 사용함
- 축적 배열에서 국지적 최대값이 발생하는 위치에서의 ρ 와 θ 값을 찾아 직선의 방정식을 구할 수 있음

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 그림 9-13(a)는 입력 영상이 사용하는 2차원 xy 좌표계이며, 파란색 직선은 $x \cos \theta_0 + y \sin \theta_0 = \rho_0$ 임
- 이 직선 위의 세 점을 선택하고, 각 점에 대응하는 $\rho\theta$ 공간에서의 곡선을 그림 9-13(b)에 나타냄
- $\rho\theta$ 공간에서 세 곡선은 하나의 점에서 모두 교차하며, 이 점의 좌표 (ρ_0, θ_0) 가 그림 9-13(a)의 파란색 직선을 나타내는 파라미터임

▼ 그림 9-13 $\rho\theta$ 파라미터를 이용한 직선의 표현



1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- ρ 와 θ 는 실수 값을 가지기 때문에 C/C++ 코드로 축적 배열을 구현하려면 ρ 와 θ 가 가질 수 있는 값의 범위를 적당한 크기로 나눠서 저장하는 양자화(quantization) 과정을 거쳐야 함
- 예를 들어 θ 는 0부터 π 사이의 실수를 가질 수 있는데, 이 구간을 180단계로 나눌 수도 있고 360단계로 나눌 수도 있음
- 구간을 촘촘하게 나눌 경우 입력 영상에서 정밀한 직선 검출이 가능하지만 연산 시간이 늘어날 수 있음
- 반면에 구간을 너무 듬성하게 나눌 경우 연산이 빨라지지만 직선 검출의 정확도가 낮아질 수 있음

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- OpenCV에서는 HoughLines() 함수를 사용하여 허프 변환 직선 검출을 수행할 수 있음

```
void HoughLines(InputArray image, OutputArray lines,
                double rho, double theta, int threshold,
                double srn = 0, double stn = 0,
                double min_theta = 0, double max_theta = CV_PI);
```

- **image** 8비트 단일 채널 입력 영상. 주로 에지 영상을 지정합니다.
- **lines** 직선 정보(ρ , θ)를 저장할 출력 벡터
- **rho** 축적 배열에서 ρ 값의 해상도(픽셀 단위)
- **theta** 축적 배열에서 θ 값의 해상도(라디안 단위)
- **threshold** 축적 배열에서 직선으로 판단할 임계값
- **srn** 멀티스케일 허프 변환에서 ρ 해상도를 나누는 값. srn 에 양의 실수를 지정하면 ρ 해상도와 ρ/srn 해상도를 각각 이용하여 멀티스케일 허프 변환을 수행합니다. srn 과 stn 이 모두 0이면 일반 허프 변환을 수행합니다.
- **stn** 멀티스케일 허프 변환에서 θ 해상도를 나누는 값
- **min_theta** 검출할 직선의 최소 θ 값
- **max_theta** 검출할 직선의 최대 θ 값

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 코드 9-3의 `hough_lines()` 함수에서는 `Canny()` 함수로 에지 영상을 구하고, 이 영상을 `HoughLines()` 함수 입력으로 사용하여 직선을 검출함
- `HoughLines()` 함수가 반환하는 직선 파라미터 정보를 이용하여 영상 위에 빨간색 직선을 그려서 화면에 나타냄

코드 9-3 허프 변환 직선 검출 예제 [ch09/hough]

```
01 void hough_lines()
02 {
03     Mat src = imread("building.jpg", IMREAD_GRAYSCALE);
04
05     if (src.empty()) {
06         cerr << "Image load failed!" << endl;
07         return;
08     }
09
```


1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

코드 9-3 허프 변환 직선 검출 예제 [ch09/haugh]

```

10     Mat edge;
11     Canny(src, edge, 50, 150);
12
13     vector<Vec2f> lines;
14     HoughLines(edge, lines, 1, CV_PI / 180, 250);
15
16     Mat dst;
17     cvtColor(edge, dst, COLOR_GRAY2BGR);
18
19     for (size_t i = 0; i < lines.size(); i++) {
20         float r = lines[i][0], t = lines[i][1];
21         double cos_t = cos(t), sin_t = sin(t);
22         double x0 = r * cos_t, y0 = r * sin_t;
23         double alpha = 1000;
24

```

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

코드 9-3 허프 변환 직선 검출 예제 [ch09/hough]

```

25         Point pt1(cvRound(x0 + alpha * (-sin_t)), cvRound(y0 + alpha * cos_t));
26         Point pt2(cvRound(x0 - alpha * (-sin_t)), cvRound(y0 - alpha * cos_t));
27         line(dst, pt1, pt2, Scalar(0, 0, 255), 2, LINE_AA);
28     }
29
30     imshow("src", src);
31     imshow("dst", dst);
32
33     waitKey(0);
34     destroyAllWindows();
35 }
```



1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 3행 building.jpg 영상을 그레이스케일 형식으로 불러와 src에 저장합니다.
- 10~11행 캐니 에지 검출기를 이용하여 구한 에지 영상을 edge에 저장합니다.
- 13~14행 HoughLines() 함수를 이용하여 직선의 방정식 파라미터 ρ 와 θ 정보를 lines에 저장합니다. 축적 배열에서 ρ 간격은 1픽셀 단위로, θ 는 1° 단위로 처리합니다.
- 17행 그레이스케일 에지 영상 edge를 BGR 3채널 컬러 영상으로 변환하여 dst에 저장합니다.
- 19행 HoughLines() 함수에 의해 구해진 직선의 개수만큼 for 반복문을 수행합니다.
- 20~26행 직선의 방정식 파라미터 중에서 ρ 를 변수 r에, θ 를 변수 t에 저장합니다. x0와 y0는 원점에서 직선에 수선을 내렸을 때 만나는 점의 좌표입니다. pt1과 pt2에는 (x0, y0)에서 충분히 멀리 떨어져 있는 직선 상의 두 점 좌표가 저장됩니다.
- 27행 검출된 직선을 두께가 2인 빨간색 실선으로 그립니다.

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

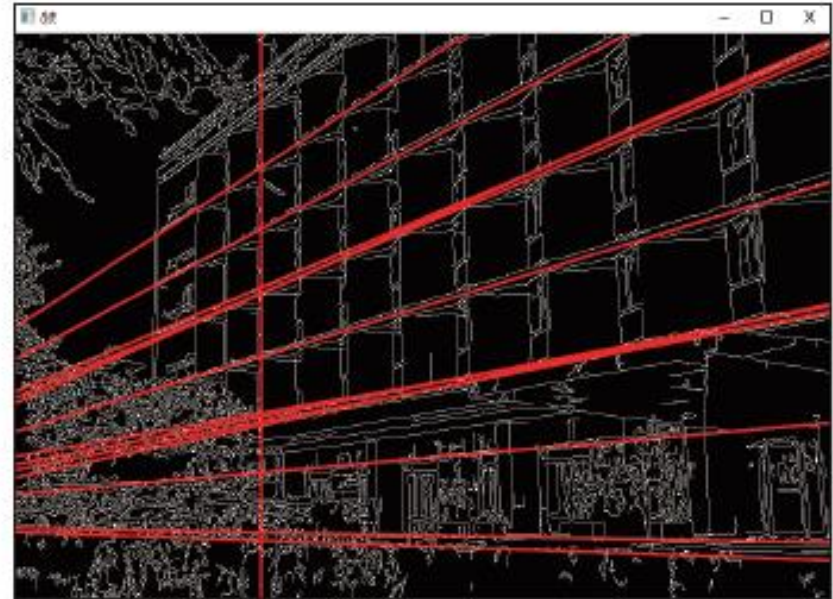
- 코드 9-3에서 사실 직선을 검출하는 작업은 앞부분에서 Canny() 함수와 HoughLines() 함수를 호출하는 것으로 완료됨
- 그 뒤에 길게 나타나는 소스 코드는 HoughLines() 함수에 의해 구한 $\rho\theta$ 파라미터 값을 이용하여 에지 영상에 빨간색 직선을 그리기 위한 코드임
- 영상 위에 직선을 그리려면 직선의 양 끝점 좌표를 알아야 하기 때문에 ρ 와 θ 값을 이용하여 직선상의 두 점 좌표 pt1과 pt2를 구함
- 이때 변수 alpha 값을 충분히 크게 설정해야 pt1과 pt2가 영상 바깥쪽에 위치하며, 자연스러운 직선을 그릴 수 있음
- 만약 alpha 값을 작게 설정하면 직선의 일부만 그려질 수 있으니 주의해야 함

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 그림 9-14에서 src는 입력 영상으로 사용한 building.jpg 파일이고, dst는 캐니 에지 검출 영상 위에 허프 변환에 의해 구해진 직선을 함께 표시한 영상임
- 만약 코드 9-3에서 HoughLines() 함수의 다섯 번째 인자로 지정한 직선 검출 임계값을 250보다 작게 설정하면 더 많은 직선을 확인할 수 있음

▼ 그림 9-14 허프 변환 직선 검출 예제 실행 결과



1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- OpenCV는 기본적인 허프 변환 직선 검출 방법 외에 확률적 허프 변환(probabilistic Houghtransform)에 의한 직선 검출 방법도 제공함
- 확률적 허프 변환 방법은 직선의 방정식 파라미터 ρ 와 θ 를 반환하는 것이 아니라 직선의 시작점과 끝점 좌표를 반환함
- 확률적 허프 변환 방법은 선분을 찾는 방법임
- OpenCV에서 확률적 허프 변환 방법은 HoughLinesP() 함수에 구현되어 있음

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

```
void HoughLinesP(InputArray image, OutputArray lines,
                 double rho, double theta, int threshold,
                 double minLineLength = 0, double maxLineGap = 0);
```

- `image` 8비트 단일 채널 입력 영상. 주로 에지 영상을 지정합니다.
- `lines` 선분의 시작점과 끝점의 정보(`x1, y1, x2, y2`)를 저장할 출력 벡터
- `rho` 축적 배열에서 ρ 값의 해상도(픽셀 단위)
- `theta` 축적 배열에서 θ 값의 해상도(라디안 단위)
- `threshold` 축적 배열에서 직선으로 판단할 임계값
- `minLineLength` 검출할 선분의 최소 길이
- `maxLineGap` 직선으로 간주할 최대 에지 점 간격

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 코드 9-4에 나타난 `hough_line_segments()` 함수의 구성은 앞서 소개한 코드 9-3의 `hough_lines()` 함수와 거의 유사함
- 다만 `HoughLinesP()` 함수는 검출한 직선의 시작점과 끝점의 좌표를 반환하기 때문에 검출된 직선을 그리는 코드가 훨씬 간단함

코드 9-4 확률적 허프 변환 선분 검출 예제 [ch09/hough]

```

01 void hough_line_segments()
02 {
03     Mat src = imread("building.jpg", IMREAD_GRAYSCALE);
04
05     if (src.empty()) {
06         cerr << "Image load failed!" << endl;
07         return;
08     }
09
10     Mat edge;
11     Canny(src, edge, 50, 150);
12

```


1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

코드 9-4 확률적 허프 변환 선분 검출 예제 [ch09/hough]

```

13     vector<Vec4i> lines;
14     HoughLinesP(edge, lines, 1, CV_PI / 180, 160, 50, 5);
15
16     Mat dst;
17     cvtColor(edge, dst, COLOR_GRAY2BGR);
18
19     for (Vec4i l : lines) {
20         line(dst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255),
21             2, LINE_AA);
22     }
23
24     imshow("src", src);
25     imshow("dst", dst);
26
27     waitKey(0);
28     destroyAllWindows();
29 }
```

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

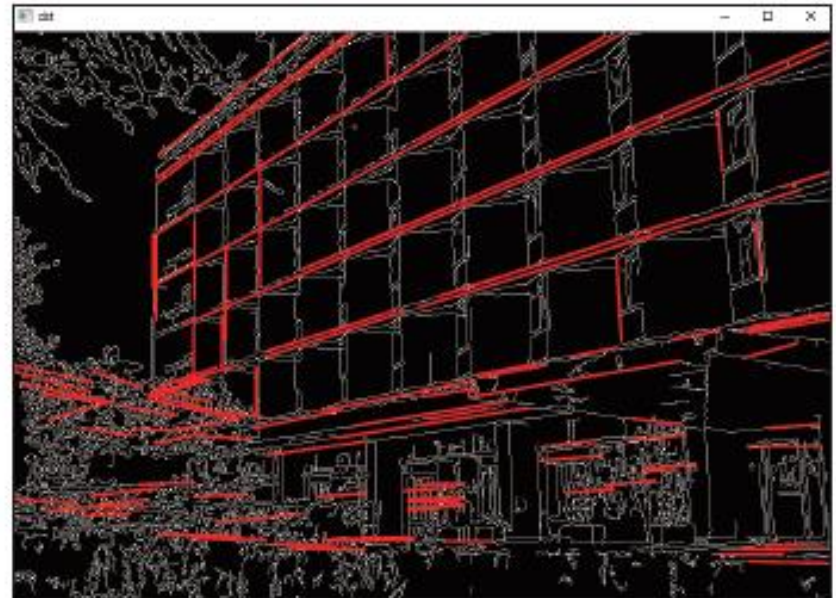
- 3행 `building.jpg` 영상을 그레이스케일 형식으로 불러와 `src`에 저장합니다.
- 11행 캐니 에지 검출기를 이용하여 구한 에지 영상을 `edge`에 저장합니다.
- 13~14행 `HoughLinesP()` 함수를 이용하여 모든 직선 성분의 시작점과 끝점 좌표를 구합니다.
- 17행 그레이스케일 에지 영상 `edge`를 BGR 3채널 컬러 영상으로 변환하여 `dst`에 저장합니다.
- 19행 `HoughLinesP()` 함수에 의해 구해진 모든 직선 성분을 `dst` 영상 위에 빨간색 직선으로 그립니다.

1. 허프 변환 직선 검출

➤ 허프 변환 직선 검출

- 그림 9-15에서 src는 입력 영상으로 사용한 building.jpg 파일임
- dst는 캐니 에지 검출 영상 위에 확률적 허프 변환에 의해 구해진 직선 성분을 함께 표시한 영상임
- 확률적 허프 변환에 의해 실제로 직선이 있는 위치에서만 선분이 그려진 것을 확인할 수 있음

▼ 그림 9-15 확률적 허프 변환 선분 검출 예제 실행 결과



2. 허프 변환 원 검출

➤ 허프 변환 원 검출

- 중심 좌표가 (a,b)이고 반지름이 r인 원의 방정식은 다음과 같이 표현함
$$(x-a)^2 + (y-b)^2 = r^2$$
- 이러한 원의 방정식은 세 개의 파라미터를 가지고 있음
- 허프 변환을 그대로 적용하려면 3차원 파라미터 공간에서 축적 배열을 정의함
- 가장 누적이 많은 위치를 찾아야 함
- 3차원 파라미터 공간에서 축적 배열을 정의하고 사용하려면 너무 많은 메모리와 연산 시간을 필요로 하게 됨
- OpenCV에서는 일반적인 허프 변환 대신 허프 그래디언트 방법(Hough gradient method)을 사용하여 원을 검출함

2. 허프 변환 원 검출

➤ 허프 변환 원 검출

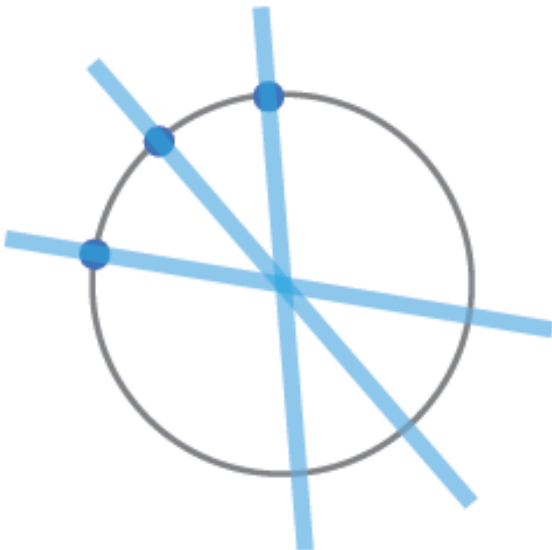
- 허프 그래디언트 방법은 두 가지 단계로 구성됨
- 첫 번째 단계에서는 영상에 존재하는 모든 원의 중심 좌표를 찾고, 두 번째 단계에서는 검출된 원의 중심으로부터 원에 적합한 반지름을 구함
- 원의 중심 좌표를 찾는 과정에서 축적 배열이 사용됨
- 다만 허프 그래디언트 방법에서 사용하는 축적 배열은 파라미터 공간에서 만드는 것이 아니라 입력 영상과 동일한 xy 좌표 공간에서 2차원 배열로 만듦
- 원의 중심을 찾기 위해 허프 그래디언트 방법은 입력 영상의 모든 에지 픽셀에서 그래디언트를 구함
- 그래디언트 방향을 따르는 직선상의 축적 배열 값을 1씩 증가시킴

2. 허프 변환 원 검출

➤ 허프 변환 원 검출

- 그림 9-16처럼 원주상의 모든 점에 대해 그래디언트 방향의 직선을 그림
- 직선상의 축적 배열 값을 증가시키면 결과적으로 원의 중심 위치에서 축적 배열 값이 크게 나타나게 됨
- 일단 원의 중심을 찾은 후에는 다양한 반지름의 원에 대해 원주상에 충분히 많은 에지 픽셀이 존재하는지 확인하여 적절한 반지름을 선택함

▼ 그림 9-16 허프 그래디언트 방법을 이용한 원 중심 검출



2. 허프 변환 원 검출

➤ 허프 변환 원 검출

- OpenCV에서는 HoughCircles() 함수를 사용하여 원을 검출할 수 있음

```
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** 입력 영상. 에지 영상이 아닌 **원본 그레이스케일 영상**을 지정합니다.
- **circles** 검출된 원 정보를 저장할 출력 벡터
- **method** **HOUGH_GRADIENT**만 지정 가능합니다.
- **dp** 입력 영상과 축적 배열의 크기 비율
- **minDist** 인접한 원 중심의 최소 거리
- **param1** Canny 에지 검출기의 **높은 임계값**
- **param2** 축적 배열에서 **원 검출을 위한 임계값**
- **minRadius** 검출할 원의 최소 반지름
- **maxRadius** 검출할 원의 최대 반지름

2. 허프 변환 원 검출

➤ 허프 변환 원 검출

- 코드 9-5에 나타난 `hough_circles()` 함수는 은색 동전이 놓여 있는 영상에서 원을 검출하고, 검출된 원을 빨간색으로 표시함

코드 9-5 허프 원 검출 예제 [ch09/hough]

```

01  void hough_circles()
02  {
03      Mat src = imread("coins.png", IMREAD_GRAYSCALE);
04
05      if (src.empty()) {
06          cerr << "Image load failed!" << endl;
07          return;
08      }
09
10      Mat blurred;
11      blur(src, blurred, Size(3, 3));
12

```



2. 허프 변환 원 검출

➤ 허프 변환 원 검출

코드 9-5 허프 원 검출 예제 [ch09/hough]

```

13     vector<Vec3f> circles;
14     HoughCircles(blurred, circles, HOUGH_GRADIENT, 1, 50, 150, 30);
15
16     Mat dst;
17     cvtColor(src, dst, COLOR_GRAY2BGR);
18
19     for (Vec3f c : circles) {
20         Point center(cvRound(c[0]), cvRound(c[1]));
21         int radius = cvRound(c[2]);
22         circle(dst, center, radius, Scalar(0, 0, 255), 2, LINE_AA);
23     }
24
25     imshow("src", src);
26     imshow("dst", dst);
27

```

2. 허프 변환 원 검출

➤ 허프 변환 원 검출

코드 9-5 허프 원 검출 예제 [ch09/hough]

```
28     waitKey(0);
29     destroyAllWindows();
30 }
```

- 3행 coins.png 동전 영상을 불러와 src에 저장합니다.
- 11행 입력 영상 src의 잡음을 제거하는 용도로 blur() 함수를 적용합니다.
- 14행 HoughCircles() 함수를 이용하여 원을 검출합니다. 축적 배열 크기는 입력 영상과 같은 크기로 사용하고, 두 원의 중심점 거리가 50픽셀보다 작으면 검출하지 않습니다. 캐니 에지 검출기의 높은 임계값은 150으로 지정하고, 축적 배열 원소 값이 30보다 크면 원의 중심점으로 선택합니다. 검출된 원의 중심 좌표와 반지름 정보는 circles 변수에 저장됩니다.
- 16~17행 입력 영상 src를 3채널 컬러 영상으로 변환하여 dst에 저장합니다.
- 19~23행 dst 영상 위에 검출된 원을 빨간색으로 그립니다.

2. 허프 변환 원 검출

➤ 허프 변환 원 검출

- 그림 9-17에서 src는 입력 영상 coins.png 파일이고, dst는 허프 그래디언트 방법으로 검출된 원을 함께 표시한 영상임
- 입력 영상에 존재하는 모든 동전에 빨간색 원이 적절하게 그려진 것을 확인할 수 있음

▼ 그림 9-17 허프 원 검출 예제 실행 결과

