Chap 9 에지 검출과 응용

에지(Edge) 검출

에지(Edge)

- 한쪽 방향으로 픽셀 값이 급격하게 바뀌는 부분 (어두운 -> 밝은 or 밝은 -> 어두운)
- 일반적으로 객체와 배경의 경계 혹은 객체와 다른 객체의 경계에서 에지가 발생.
- 영상에서 에지를 찾아내는 작업은 객체의 윤곽을 알아낼 수 있는 유용한 방법
- 객체 판별을 위한 전처리로 에지 검출을 사용.

미분과 그래디언트

- 기본적으로 영상에서 에지를 찾기 위해서는 픽셀값의 변화율을 측정하여 변화율이 큰 픽셀을 선택
- 수학에서 함수 또는 데이터의 변화율을 미분이라 함. 즉 순간 변화율
- 함수 f(x)의 값이 급격하게 바뀌는 부분을 찾기 위해서는 함수의 미분값이 0보다 훨씬 크거나 훨씬 작은 위치를 찾아야 한다.

- 그러나 영상은 2차원 평면 위에 픽셀값이 정형화되지 않은 상태로 나열되어 있는 형태이므로 미분 공식을 적용할 수 없다.

영상으로부터 미분을 계산하려면 두가지 특성을 고려

1. 영상이 2차원 평면에서 정의된 함수
2. 영상이 정수 단위 좌표에 픽셀이 나열되어 있는 이산함수

미분 근사

1. 전진차분 - 구하고자 하는게 x일때 한칸 앞 (x + h)로 부터의 변화율
2. 후진차분 - 구하고자 하는게 x일때 한칸 뒤 (x - h)로 부터의 변화율
3. 중앙차분 - (x + h) - (x - h)를 분자로하고 분모로 2h로 나누어 준다.

- 세가지의 방법중 중간값 차이를 이용하는 방법이 이론적으로 가장 근사화 오류가 적고 많이 사용하는 방법

2차원 평면에서의 미분

- 영상은 2차원 평면에서 정의된 함수. 가로 방향, 세로 방향 각각 미분해야함
- 즉, 한쪽 방향 좌표는 고정하고, 나머지 방향으로만 미분근사를 계산. 이후 각각의 값을 합성

1. x축 방향으로 편미분 하는 1*3필터 마스크

-1	0	1

2. y축 방향으로 편미분 하는 3*1필터 마스크

-1
0

-1
1

- 편미분 근사 수식을 그대로 적용해야 하지만 보통은 미분값의 상대적 크기를 중요시 하기 때문에 단순한 마스크를 주로 사용

그래디언트

- 2차원 공간에서 정의된 영상에서 에지를 찾으려면 x축 방향, y축 방향 편미분을 모두 사용해야 함.
- x축 방향, y축 방향 미분을 한꺼번에 백터로 표현한 것을 그래디언트 라고 함.

1. 그래디언트는 백터이기에 *크기* 와 *방향* 성분으로 표현 가능
2. 그래디언트의 백터의 방향은 변화정도가 가장 큰 방향을 나타냄
3. 그래디언트 백터의 크기는 변화율 세기를 나타내는 척도

2차원 영상에서 에지를 찾는 기본적인 방법

- 그래디언트 크기가 특정 값보다 큰 위치를 찾는 것
- 여기서 에지 여부를 판단하기 위해 기준이 되는 값을 *임계값* 또는 문턱치라 한다.
- 임계값은 영상의 특성에 따라 다르게 설정해야 하고, 보통은 사용자의 경험에 의해 결정된다.

> 일반적으로 임계값을 높게 설정하면 밝기차이가 급격한 에지픽셀만 검출. 낮게 설정하면 약한 에지 성분도 검출

마스크 기반 에지 검출

1. x축 방향으로의 편미분을 구하는 소벨 마스크

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

2. y축 방향으로의 편미분을 구하는 소벨 마스크

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

- 연산은 현재 행과 이웃 행에서의 픽셀 값 변화가 유사하다는 점을 이용하여 잡음의 영향을 줄임
- 현재 행에서 두 번의 중앙 차분 연산을 수행하는 이유 -> 현재 행의 중앙차분근사에 더 큰 가중치를 주기 위해

마스크 기반 에지 검출 예제

```
# code는 간단하게 sudo로 표현

def sobel_derivative():
    src = cv.imread()

    if src is None:
        return

    mx = np.array([[-1,0,1],
                   [-2,0,2],
                   [-1,0,1],])

    my = np.array([[-1,-2,-1],
                   [0,0,0],
                   [1,2,1],])

    dx = cv.filter2D(src, -1, mx, delta=128)
    dy = cv.filter2D(src, -1, my, delta=128)
```

소벨(Sobel)필터

- 소벨 마스크를 이용해 영상을 미분하는 sobel()
- sobel()은 3*3 소벨 마스크 또는 확장된 형태의 큰 마스크를 이용해 영상을 미분함
- ddepth인자에 명시정으로 결과 영상의 자료형을 지정해야 함. (-1을 지정하면 원본과 같은 타입)
- dx, dy인자는 각각 방향의 편미분 차수를 의미
- sobel()은 x,y방향으로의 고차미분을 계산할 수 있지만 대부분 1차 미분을 구하는 용도로 사용

샤르(Sharr)필터

- 3*3 소벨마스크보다 정확한 미분계산을 수행

1. x축

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

2. y축

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

-3	-10	-3
3	10	3

- 사용법: Scharr()함수를 호출해서 사용한다.

- Sobel()에 ksize인자에 FILTER_SCHARR 혹은 -1을 넣어도 샤르필터가 적용됨.
- sobel() or Sharr()의 계산 결과를 저장하고 이를 통해 *그래디언트* 크기를 계산 할 수 있음

그래디언트 계산법

- opencv는 2차원 백터의 x방향 좌표와 y방향 좌표를 이용하여 ㅑ magnitude() 함수를
- magnitude() 함수의 입력으로 사용되는 x와 y는 CV_32F or CV_64F의 깊이를 사용하는 행렬 or 백터
- 만약 x방향 y방향 두개의 행렬이 있을때 그래디언트 방향을 계산하고 싶다면 phase()함수를 사용할 수 있음
- phase()함수에서 x와 y는 입력이고, angle은 출력 -> 방향을 구할 수 있다.

sobel_edge()예제

- sobel_edge()함수는 x축 방향과 y축 방향 1차 미분을 구하고 그래디언트 크기가 특정 임계값보다 큰 픽셀을 에지로 검출
- x축 방향으로 1차 편미분, y축 방향으로 1차 편미분을 각각 도출하여 dx,dy 행렬에 저장
- dx,dy행렬로부터 그래디언트 크기를 계산하여 fmag에 저장

```
# sudo코드로 로직만 작성

def sobel_edge():
  src = cv.imread()

  dx = cv.Sobel(src, cv.CV_32F,1,0)
  # x에 대한 1차 편미분
  dy = cv.Sobel(src, cv.CV_32F,0,1)
  # x에 대한 1차 편미분

  fmag = cv.magnitude(dx,dy)
  mag = np.uint8(np.clip(fmag,0,255))
  # 값을 cliping하는 과정

  _,edge = cv.threshold(mag,150,255,cv.THRESH_BINARY)
  # 결과중 150이상을 255로 채움

  cv.imshow()
```

- code를 실행하면 edge영상에서 좀더 깔끔하게 이진화 된 모습을 확인 가능.

캐니 에지 검출기

- 에지 검출을 최적화 문제 관점으로 접근. 소벨 에지 검출 방법의 단점을 해결할 수 있는 방법을 제시.
- 케니는 3가지 항목을 좋은 에지 검출기의 조건으로 제시.

1. 정확한 검출
2. 정확한 위치
3. 단일에지

- 케니 에지 검출기는 그래디언트의 크기와 방향을 모두 고려하여 좀 더 정확한 에지 검출 가능
- 에지는 서로 연결되어 있는 가능성이 높다는 점을 고려하여 그래디언트 크기가 다소 약하게 나타나는 에지도 놓치지 않고 찾음

과정

가우시안 필터링 -> 그래디언트 계산 -> 비최대 억제 -> 이중임계값을 이용한 히스테리시스 에지 트래킹

1. 가우시안 필터링

- 가우시안 필터를 적용함으로 잡음을 제거. 그러나 영상이 부드러워 지면서 에지의 세기도 함께 감소할 수 있기 때문에 적절한 표준편차를 선택해야함.
- 잡음이 심하지 않다면 가우시안 필터링 생략가능

2. 그래디언트 계산

- 가로, 세로 방향으로 각각 소벨 마스크 필터링을 수행한 후, 그래디언트의 크기와 방향을 모두 계산
- L2노름 (그래디언트의 크기) , L1노름(L2대신 간단한 수식으로 계산. 연산속도 향상을 위해 사용)

3. 비최대 억제

- 에지 검출을 위해 단순히 그래디언트 크기가 특정 임계값보다 큰 픽셀을 선택할 경우, 에지 근방의 여러 픽셀이 한꺼번에 에지로 선택 될 수 있음 -> 즉, 에지가 두껍게 표현되는 현상이 발생
- 이를 방지하기 위해 비최대 억제 과정을 사용. -> 그래디언트 크기가 국지적 최대인 픽셀만을 에지 픽셀로 설정 -> 즉 가장 큰 값만 선택. 그래서 비최대억제
- 그래디언트 백터의 방향과 같은 방향에 있는 인접 픽셀끼리만 국지적 최대 검사를 수행. (근처 픽셀을 다보지 않고 해당 백터 방향만 보고 검사)
- 결과적으로 가장 변화율이 큰 위치의 픽셀이 에지로 검색.

4. 이중임계값을 이용한 히스테리시스 에지 트래킹

- 하나의 임계값을 사용할 경우 이분법으로 결과가 판단되기 때문에 환경 변화에 민감해 질 수 있다.
- 그래서 캐니 에지 검출기는 2개의 임계값을 사용.
- 높은 임계값을 T_high, 낮은 임계값을 T_low로 표기

- 만약 그래디언트 크기가 T_high보다 크면 이 픽셀은 최종적으로 에지. -> 강한에지
- 그래디언트 크기가 T_low보다 작으면 에지 픽셀이 아님.
- 사이에 있다면 추가 검사를 수행 -> 약한에지

--> 캐니에지 검출기의 마지막 단계에서는 히스테리시스 에지 트래킹 방법을 사용하여 약한 에지중 최종적으로 에지로 판별할 픽셀을 선택

히스테리시스 에지 트래킹

- 에지 픽셀이 대체로 상호 연결되어 있다는 점을 이용.
- 만약 약한 에지픽셀이 강한 에지 픽셀과 서로 연결되어 있다면 이 픽셀은 최종적으로 에지로 판단, 반대의 경우는 에지가 아니라고 판단.

케니 에지 검출기 예제

```
def canny_edge():
  src = cv.imread()

  #인자는 두가지 임계값 쌍
  dst1 = cv.Canny(src,50,100)
  dst2 = cv.Canny(src,50,150)

  cv.imshow('dst1',dst1)
  cv.imshow('dst2',dst2)
```

- 임계값을 낮출수록 잡음이 검출됨으로 주의

직선 검출과 원 검출

허프 변환 직선 검출

- 영상에서 직선 성분을 찾기 위해서는 우선 에지를 찾아내고 에지 픽셀들이 일직선상에 배열되어 있는지 확인.
- 해당 용도로 허프 변환 기법이 널리 사용됨.
- 허프변환은 2차원 x,y좌표에서 직선의 방정식을 파라미터 공간으로 변환하여 직선을 찾는 알고리즘

- 일반적인 직선의 방정식은 y = ax + b 형태. a는 기울기 b는 y절편.
- 이 직선의 방정식은 가로축이 x, 세로축이 y인 2차원 x,y좌표 공간에 정의
- a와 b는 직선의 형태를 결정하는 파라미터
- b = -xa + y

허프변환으로 직선의 방정식을 찾는 방법

- x,y공간에서 에지로 판별된 모든 점을 이용하여 a,b파라미터 공간에 직선을 표현

- 직선이 많이 교차되는 좌표를 모두 찾음 -> 이때 축적배열을 사용 (축적배열은 0으로 초기화 된 2차원 배열에서 직선이 지나가는 위치의 배열 원소값을 1씩 증가시켜 생성)
- 기타 설명은 요약이 안됨. -> 매우 어려움 해당 페이지 상세 참조. 그림을 봐야 이해가 됨.

허프 변환 직선 검출 예제

- hough_lines()함수로 검출 가능. Canny()함수로 에지 영상을 구하고 해당 영상을 HoughLines()함수 입력으로 사용하여 직선
- HoughLines()함수가 반환하는 직선 파라미터 정보를 이용하여 영상 위에 빨간 직선을 그리는 예제

```
# sudo코드로 로직 표현
def hough_lines():
  src = cv.imread()

  edge = cv.Canny(src,50,150)
  lines = cv.HoughLines(edge, 1, math.pi / 180, 250)

  dst = cv.cvtColor(edge, cv.COLOR_GRAY2BGR)

  if lines is not None:
    for i in range(linse.shape[0]):
      rho = linse[i][0][0]
      # 축적배열에서 픽셀단위 해상도
      theta = linse[i][0][1]
      # 픽셀단위에서 라디안단위 해상도
      cos_t = math.cos(theta)
      sin_t = math.sin(theta)
      x0, y0 = rho * cos_t, rho * sin_t
      alpha = 1000
      pt1 = (int(x0 - alpha * sin_t), int(y0 + alpha * cos_t))
      pt2 = (int(x0 + alpha * sin_t), int(y0 - alpha * cos_t))
      cv.line(dst, pt1, pt2, (0,0,255), 2, cv.LINE_AA)
```

- 중요한점은 alpha값을 충분히 크게 설정해야 자연스러운 직선을 그릴 수 있음.

확률적 허프 변환

- openCV는 기본적인 허프 변환 직선 검출 방법 외에 확률적 허프 변환에 의한 직선 검출방법도 제공
- 확률적 허프 변환 방법은 직선의 방정식 파라미터를 반환하지 않고, 직선의 시작점과 끝점 좌표를 반환. -> 즉 선분을 찾는 방법
- HoughLinesP()함수에 구현

```
def hough_line_segments():
  src = cv.imread()
  edge = cv.Canny(src, 50, 150)
```

```
  lines = cv.HoughLines(edge, 1, math.pi / 180, 160, minLineLength = 50,
maxLineGroup=5)

  dst = cv.cvtColor(edge, cv.COLOR_GRAY2BGR)

  if lines is not None:
    for i in range(linse.shape[0]):
      pt1 = (lines[i][0][0], lines[i][0][1])
      pt2 = (lines[i][0][2], lines[i][0][3])

      cv.line(dst, pt1, pt2, (0,0,255), 2, cv.LINE_AA)

  cv.imshow()
```

허프변환 원 검출

- 원의 방정식은 3개의 파라미터를 갖고 있음.
- 허프변환을 적용하려면 3차원 파라미터 공간에서 축적 배열을 정의. -> 가장 누적이 많은 위치를 찾아야 함

- 그러나 3차원 파라미터 공간에서 축적배열을 사용하면 자원소모가 심해 허프 그래디언트 방법으로 원을 검출

허프그래디언트 방법

- 먼저 영상에 존재하는 모든 원의 중심 좌표를 찾고, 검출된 원의 중심으로 부터 원에 적합한 반지름을 구함
- 이때 축적배열은 파라미터 공간에서 만들지 않고, 입력 영상과 동일한 x,y좌표 공간에서 2차원 배열로 만듦
- 원의 중심을 찾기 위해서 일단 모든 에지 픽셀에서의 그래디언트를 구함. 그리고 그래디언트 방향을 따르는 직선상의 축적 배열값을 1씩 증가
- 모든 점에 대해 직선을 그리면 원의 중심은 축적 배열값이 크게 증가되어짐. 원의 중심을 이렇게 찾고, 이후 다양한 반지름의 원에 대해 충분히 많은 에지 픽셀이 존재하는지 확인하고, 적절한 반지름을 선택
- HoughCircles()를 통해 원 검출이 가능.

```
def hough_circle():
  src = cv.imread()

  blurred = cv.blur(src, (3,3))
  # 잡음 제거를 위해 블러를 사용

  circles = cv.HoughCircles(blurred, cv.HOUGH_GRADIENT, 1, 50, param1 =
150, param2 = 30)

  dst = cv.cvtColor(src, cv.COLOR_GRAY2BGR)

  if circles is not None:
```

```
    for i in range(circles.shape[1]):
      cx,cy,radius = circles[0][i]

      # 실제 원을 그리는 코드 좌표와, 반지름을 넣어준다.
      cv.circle(dst, (cx,cy), radius, (0,0,255), 2, cv.LINE_AA)
```