9주차: 에지 검출

1. 에지: 한쪽 방향으로 **픽셀값이 급격하게 바뀌는 부분**

1. 변화율이 큰 픽셀을 선택하기 위해 미분과 그래디언트 계산함

2. 미분과 그래디언트

1. 단, 영상의 미분 계산시 두가지 특성 고려
   1. 영상이 2차원 평면상 정의된 함수
   2. 영상이 정수 단위 좌표에 픽셀이 나열된 이산함수

2. 미분근사(구하려는 게 x일 때)
   1. 전진차분: $(x+h)$
   2. 후진차분: $(x-h)$
   3. 중앙차분: $\frac{(x+h-x-h)}{2h}$

3. 2차원 평면에서 미분 (주로 단순 마스크 사용 why? 미분값의 상대적 크기를 중시하므로)

   - x축 방향으로 편미분하는 **1x3마스크**: (오른쪽으로) |-1 | 0 | 1 |
   - y축 방향으로 편미분하는 **3x1마스크**: (아래로) |-1 | 0 | 1 |

4. 그래디언트: x축 방향 y축 방향 미분을 한꺼번에 벡터로 표현한 것

   - 크기와 방향 성분으로 표현 가능
     - 크기: **변화율의 세기**를 나타내는 척도
     - 방향: **변화 정도가 가장 큰** 방향

5. 2차원영상에서 에지 찾는 방법
   1. 그래디언트 크기의 특정값보다 큰 위치 찾기
   2. 임계값은 영상의 특성에 따라 다르게 설정

   - 보통은 사용자의 경험에 의해 결정됨
   - 용어) 임계값: 엣지 여부를 판단하는 기준
   - 예시) 임계값 높게 설정: 밝기차가 급격한 엣지픽셀만 검출 / 낮게 설정: 너도나도 다 엣지라고 판단됨

3. 마스크 기반 엣지 검출 (소벨 필터)

```
# x축방향 마스크
mx = np.array([[-1,0,1],
               [-2,0,2],
               [-1,0,1]])
# y축방향 마스크
my = np.array([[-1,-2,-1],
               [0,0,0],
               [-1,0,1]])

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

3. 소벨API를 생성한 엣지 검출: `Sobel()`

```
sobelx = cv.Sobel(src, -1, 1, 0, ksize=3)
sobely = cv.Sobel(src, -1, 0, 1, ksize=3)
```

3. 마스크 기반 엣지 검출 (샤르 필터)

```
# x축방향 마스크
mx = np.array([[-3,0,3],
               [-10,0,10],
               [-3,0,3]])
# y축방향 마스크
my = np.array([[-3,-10,-3],
              [0,0,0],
              [3,10,3]])
dx = cv.filter2D(img, -1, mx) # ksize인자에 -1을 넣어 샤르필터
dy = cv.filter2D(img, -1, my)
```

3. 샤르 API를 생성한 엣지 검출: `Scharr()`

```
scharrx = cv.Scharr(src, -1, 1, 0)
scharry = cv.Scharr(src, -1, 0, 1)
```

4. 결과출력(참고)

```
merged1 = np.hstack((src, edge_gx, edge_gy))
merged2 = np.hstack((src, scharrx, scharry))
merged = np.vstack((merged1, merged2))
cv2.imshow('Scharr', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5. 그래디언트 계산법

1. 2D 벡터의 크기 계산 함수: `magnitude()`

```
magnitude(x, y, magnitude=None) -> magnitude
```

2. 2D 벡터의 방향 계산 함수: `phase()`
    - x와 y는 입력이고, angle은 출력 -> 방향을 구할 수 있다.

```
cv2.phase(x, y, angle=None, angleInDegrees=None) -> angle #
angleInDegrees: True이면 각도 단위, False이면 래디언 단위
```

6. sobel_edge()예제

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

  #x,y에 대해 1차 편미분
  dx = cv.Sobel(src, cv.CV_32F, 1,0)
  dy = cv.Sobel(src, cv.CV_32F, 0,1)

  #값 클래핑
  fmag = cv.magnitude(dx,dy)
  mag = np.uint8(np.clip(fmag,0,255)) #실수에서 정수로

  _, edge = cv.threshold(mag, 150, 255,cv.THRESH_BINARY) #결과중 159 이상은
155로
  cv.imshow()
```

7. 캐니 에지 검출기 => 소벨 에지 검출 방법의 단점 해결 방법 제시

1. 좋은 에지 검출 조건 3가지
    - **정확한 검출**
    - **정확한 위치**
    - **단일 에지**
2. 캐니 과정
    - 가우시안 필터링 -> 그래디언트 계산 -> 비최대억제 -> 이중 임계값을 이용한 히스테리시스 에지 트래킹
        - **가우시안 필터링**: 적절한 표준편차 선택필요(잡음 안심하면 생략도 가능)
        - **그래디언트 계산**: 각각 소벨 마스크 필터링 후 그래디언트 크기와 방향 모두 계산함
        - **비최대억제**: 가장 변화율이 큰 위치의 픽셀이 에지로 검색됨
            - 인접 픽셀끼리만 국지적 최대 검사 수행(근처 픽셀 다보는거 x, 해당 벡터 방향만 보고 검사).
        - **이중임계값 이용한 히스테리시스 에지**: 2개의 임계값을 사용함(T_high, T_low)
            - 만일 그래디언트 크기가 T_high보다 크면 => 강한에지
            - T_low보다 작으면 => 에지 픽셀 아님
            - 사이인 경우는 추가검사 수행함 => 약한 에지
            - 즉, 에지 픽셀은 상호연결된 점을 이용해 약한 에지 픽셀이 강한 엣지 픽셀과 연결돼 있으면 최종적으로 엣지로 판단, 반대는 엣지 아님
3. 코드

```
dst1 = cv.Canny(src, 50, 100)
dst2 = cv.Canny(src, 50, 150) #임계값이 낮아지면 잡음 검출됨
```

8. 직선 검출과 원 검출

1. 허프 변환 직선 검출
    - 우선 엣지를 찾아내고 엣지 픽셀들이 일직선상에 배열 되어 있는지 확인함
    - 2차원 x,y좌표에서 직선의 방정식을 파라미터 공간으로 변환해 직선을 찾는 알고리즘

```
src = cv.imread()
edge = cv.Canny(src, 50, 150)
lines = cv.HoughLines(edge, 1, math.py / 180, 250)
dst = cv.cvtColor(edge, cv.COLOR_GRAY2BGR)

if lines is not None:
  for i in range(lines.shape[0]):
    rho = lines[i][0][0] # 축적 배열에서 픽셀 단위 해상도
    theta = lines[i][0][1] # 픽셀 단위에서 라디안 단위 해상도

    cos_t = math.cos(theta) #코사인값 구함
    sin_t = math.sin(theta) #사인값 구함

    x0, xy = rho * cos_t, rho * sin_t # 각각 곱함
    alpha = 1000 # 이 값을 크게 설정해야 자연스러운 직선 그림
    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 # 빨갛게 선 그리기
```

2. 확률적 허프 변환
    - 직선의 시작점과 끝점 좌표를 반환해 선분을 찾는 방법임 -> 코드 간결
    - `HoughLinesP()`함수

```
src = cv.imread()
edge = cv.Canny(src, 50,150)
lines = cv.HoughLines(edge, 1, math.py/180, 160, minLineLegth=50,
maxLineGruop = 5)
dst = cv.cvtColor(edge, cv.COLOR_GRAY2BGR)

if lines is not None:
  for i in range(lines.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()
```

3. 허프 변환 원 검출
    - 3차원 파라미터 공간에서 축적 배열 정의-> 가장 많은 누적 위치를 찾아야함 (but. 자원 소모가 심해 허프 그래디언트 방법으로 원 검출)

```
src = cv.imreaD()
blurred = cv.blur(src, (3,3)) #잡음 제거 위한 블러링
circles = cv.HoughCircles(bluured, 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]
    cv.circles(dst, (cx,cy), radius, (0, 0, 255, 2, cv.LINE_AA)
```

10주차: 컬러영상처리

1. 컬러영상 다루기

```
- 특징
  - openCV는 BGR / 0: 검정(색상성분x), 255: 흰색(색상성분 가득)
  - `src[0,0]하여 픽셀값 가져옴
```

```
src = imread() #원본 영상
dst = np.zeros(src.shape,0 src.dtype) #새 영상
for j in range(src.shape[0]): #dst에 원본 복사함
  for i in range(src.shape[1]):
    p1 = src[j,i]
    p2 = dst[j,i]

    p2[0] = 255 - p[0] #B 255에서 색상성분을 각각 빼주는 것
    p2[1] = 255 - p[1] #G
    p2[2] = 255 - p[2] #R
```

2. 색 공간 변환: BGR 세가지 색 성분 조합으로 색을 표현하는 방식

1. 종류
    - HSV, HSL : 컬러영상 처리 용이함
    - YCrCb, YUV: 휘도구분됨(주로 프린트할 때)
2. `cvtColor()`함수
    - `BGR2GRAY GRAY2BGR BGR2HSV BGR2YCrCb`

3. 색상채널 나누기

1. `split()`함수: 나누기
    - `cv.split(src)`

2. `merge()`함수: 병합

4. 컬러영상 처리기법

1. 색감 유지하고 명암비만 높이는 방법(로직)
    - `cv.cvtColor(src, cv.COLOR_BGR2YCrCb)` **YCrCb로 컬러영상 변환**
    - `cv.split(src_ycrcb)`로 Y, Cr, Cb **성분 추출**
    - `cv.equalizeHist(ycrcb_planes[0])`로 **컬러 히스토그램 평활화**
    - `cv.merge(ycrcb_planes)`로 합병
    - `cv.cvtColor(src, cv.COLOR_YCrCb2BGR)`로 다시 **BGR로 컬러영상 변환**

5. 색상 범위 지정에 의한 영상 분할

1. `inRange()`함수
    - `mask = cv.inRange(src_hsv, lowerb, upperb)`

6. 히스토그램 역투영

1. 빨간색, 노란색, 녹색, 파란색 등의 원색 찾을 때 효과적
2. `calcBackProject()`함수, 상세코드는.. 10번보기

11주차: 이진화와 모폴로지

1. 영상의 이진화

1. `threshold()`함수
    - 열거형 상수를 통해 함수의 동작이 결정됨. -- `cv.THRESH_BINARY` - 기본동작
      -- `cv.THRESH_BINARY_INV` - 반전을 시킴
      -- `cv.THRESH_TRUNC` - 크면 넣고, 아니면 원본값
      -- `cv.THRESH_TOZERO` - 크면 원본, 아니면 0
      -- `cv.THRESH_TOZERO_INV` - 크면 0, 아니면 원본값
      -- `cv.THRESH_OTSU` - 오츠 알고리즘으로 자동 임계값 설정 - 앞선 5개와 | 연산자로 조합해서 사용
      -- `cv.THRESH_TRIANGLE` - 삼각 알고리즘으로 자동 임계값 설정 - 앞선 5개와 | 연산자로 조합해서 사용

2. 적응형 이진화

1. 전역이진화: 영상 모든 픽셀을 같은 임계값을 적용해 이진화 수행하는 방식 => **단, 영상 특성상 적용시키기 어려운 경우도 있음**
    - 예) 불균일한 조명성분 -> 하나의 임계값으로 객체와 배경 구분이 어려움 => **각 픽셀마다 서로 다른 임계값을 사용하면 좋음**
    - 적응형 이진화는 영상의 모든 픽셀이 정해진 크기와 사각형 블록영역 설정, 블록 내부 픽셀값 분포로 고유 임계값 결정해 이진화함

2. `adaptiveThreshold()`함수

```
bsize = pos
if bsize % 2 == 0:
    bsize = bsize -1
```

```
if bsize <3:
    bsize = 3
dst = cv.dadaptiveThreshold(src, 255, cv/ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, bsize, 5)
```

3. 모폴로지 연산: 영상의 객체 형태 및 구조에 대해 분석,처리하는 기법

1. 구조요소: **(대부분)3x3 정방향 구조요소 사용 and (대부분) 고정점을 중심으로 사용**
    - `getStructuringElement()`함수로 구조요소 간단히 생성 가능
2. 침식과 팽창(모폴로지의 대표적 연산)
    - 침식: `erode()`함수, 외곽선 깎이는 연산으로 **객체영역 축소, 배경 확대**
    - 팽창: `dilate()`함수, 왜곽선 확대하는 연산으로 **객체영역 확대, 배경 축소**

```
_, src_bin = cv.threshold(src, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)
dst1 = cv.erode(src_bin, None)
dst2 = cv.dilate(src_bgin, None)
```

4. 열기와 닫기

1. 열기: 침식 후 팽창 (작은 크기 객체 효과적 제거 가능)
2. 닫기: 팽창 후 침식 (객체 내부의 작은 구멍 매꾸기 가능)

```
dst1 = cv.morphologyEX(src_bin, cv.MORPH_OPEN, None)
dst2 = cv.mophologyEX(src_bin, cv.MORPH_CLOSE, None)
```
