HW 03 – REPORT

소속 : 정보컴퓨터공학부

학번 : 201824523

이름 : 안혜준

1. 서론

Canny edge detection의 구현을 직접 해본다. 이미지 파일은 각 픽셀의 intensity를 저장하고 있는 배열로 정보를 얻기 위해선 가공이 필요하다. Canny edge detection은 edge detection의 방법 중 하나로 raw 이미지에서 edge의 정보를 추출하는 것이다.

Edge는 표면 법선, 깊이, 색상, 빛 불연속 등에 의해 발생한다. 픽셀에서 보았을 때 급격히 intensity value가 변하는 지점을 edge라고 간주할 수 있다.

일반적인 edge detection에서는 다음과 같다. Edge를 판별하기위해 미분을 하여 변곡점을 찾는다. 하지만 이미지는 discrete하기 때문에 근사를 취한다. 근사는 픽셀의 근처 neighbors에 대해 적절한 커널을 곱함으로 이루어진다. 구한 gradient를 바탕으로 적절한 threshold를 부여하여 edge를 검출한다.

Canny edge detection에서는 Sobel Kernel을 사용하여 gradient를 계산한다. 이어서 non-maximum suppression을 통해 local maxima한 edge만을 남긴다. 그리고 두 개의 threshold를 사용하여 strong edges와 weak edges, non-edges로 분류한다. 마지막으로 hysteresis 작업을 통해 strong edge와 이어지지 않은 weak edge는 삭제하여 edge detection을 수행한다.

정리하자면:

1. Noise reduction
2. Find the intensity of gradient
3. Non-Maximum Suppression
4. Double threshold
5. Edge linking by hysteresis

의 순서로 edge detection이 이루어진다.

실습에 사용될 이구아나 이미지

실습에서 사용될 이미지이다. 이 이미지에 canny edge detection을 단계별로 적용시킬 것이다.

2. 본론

1. Noise reduction

이미지에는 기본적으로 노이즈 픽셀이 존재할 가능성이 높다. 노이즈는 blurring을 통해 제거가 가능하다. 이미지가 blur가 되어도 edge의 정보는 대부분 유지되기 때문에 정보의 유실을 걱정할 필요가 없다.

PIL을 사용하여 이미지를 열고 gray scale의 NumPy 배열로 만든다. Gaussian filter로 convolution을 수행한다. 다음은 원본과 noise reduction을 수행한 이미지 사진이다.

좌 – 원본 이미지, 우 – noise reduction을 수행한 이미지

2. Finding the intensity of gradient of the image

먼저 sobel kernel을 만든다. 컴퓨터에서는 y축이 아래로 갈수록 크기 때문에 Y filter가 뒤집어져 있다. 도함수의 근사는 F[x, y + 1] – F[x, y -1]의 형태가 되어야 하고, convolution이 적용되기 때문에 y + 1 위치가 -1, y -1 위치가 +1의 형태로 구성되었다.

```
xFilter = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]], np.float32)
yFilter = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)
```

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

X filter

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

Y filter

kernel들을 사용하여 각각 convolution 연산을 취해 Ix, Iy를 구한다. NumPy의 hypot과 arctan2를 사용하여 gradient와 theta를 구한다.

```
G = np.hypot(Ix, Iy)
G = G / np.max(G) * 255
```

```
theta = np.arctan2(Iy, Ix)
```

gradient intensity 이미지

3. Non-Maximum Suppression

각 픽셀들의 gradient 방향(theta)에 따라 local maxima를 계산한다. 주변 8개의 픽셀들과의 관계를 바탕으로 계산하기 때문에 theta값에 따라 0도, 45도, 90도, 135도로 분류하여 계산한다. 여기서도 컴퓨터의 y축은 아래로 자라기 때문에 우하와 좌상이 45도가 된다.

45도	90도	135도
0도		0도
135도	90도	45도

각도에 따라 분류하기에 앞서 theta를 radian에서 degree로 변환시킨다. 변환된 degree들에 같은 방향이면 같은 값을 가지도록 음수 값에 18도를 더해준다.

```
angle = theta * 180 / np.pi
angle[angle < 0] = angle[angle < 0] + 180
```

가장 바깥을 제외한 픽셀들의 degree에 대해 0도, 45도, 90도, 135도로 근사 시켜, 좌우 극한을 구해 local maxima인지 판단한다. Local maxima인 값 만을 남기고 나머지는 0으로 부여해 non-maximum suppression을 수행한다.

```
for i in range(1, H - 1):
    for j in range(1, W - 1):
        # 0도
        if (157.5 <= angle[i, j] <= 180) or (0 <= angle[i, j] < 22.5):
            lLimit = G[i, j - 1]
            rLimit = G[i, j + 1]

        # 45도
        elif 22.5 <= angle[i, j] < 67.5:
            lLimit = G[i - 1, j + 1]
            rLimit = G[i + 1, j - 1]

        # 90도
        elif 67.5 <= angle[i, j] < 112.5:
            lLimit = G[i - 1, j]
            rLimit = G[i + 1, j]

        # 135도
        elif 112.5 <= angle[i, j] < 157.5:
            lLimit = G[i - 1, j - 1]
            rLimit = G[i + 1, j + 1]

        # 좌우보다 더 큰것만 저장
        if (G[i, j] > lLimit) and (G[i, j] > rLimit):
            res[i, j] = G[i, j]
```

Non-maximum suppression이 적용된 이미지

4. Double threshold

2개의 threshold를 계산한다.

```
diff = np.max(img) - np.min(img)
thresholdH = np.min(img) + diff * 0.15
thresholdL = np.min(img) + diff * 0.03
```

Gradient intensity 값이 high threshold 이상이면 strong edge, high threshold보다 낮지만 low threshold보다 높다면 weak edge로 저장한다. Strong edge는 255, weak edge는 80으로 저장한다.

```
res = np.zeros(img.shape)
res = np.where((thresholdL <= img)&(img < thresholdH), np.ones(img.shape)*weak, res)
res = np.where(thresholdH <= img, np.ones(img.shape) * strong, res)
```

Double threshold가 적용된 이미지

5. Edge Tracking by hysteresis

Strong edge은 확실한 edge이다. Weak edge는 strong edges와 연결된 경우만 edge로 인정된다. 따라서 strong edges에 대해 dfs를 수행하여 strong edges와 연결된 weak edge들을 찾아낸다.

여기서 strong edge와 연결되었음은 그 픽셀을 둘러싸고 있는 주변 8개의 픽셀에 strong edge 픽셀이 존재하는 지로 판단한다. 또한, non-maximum suppression을 통해 테두리의 픽셀은 0으로 하였기 때문에 가장 바깥 테두리는 제외하고 찾는다.

```
def dfs(img, res, i, j, visited=[]):
    # 호출된 시점의 시작점 (i, j)은 최초 호출이 아닌 이상
    # strong 과 연결된 weak 포인트이므로 res에 strong 값을 준다
    res[i, j] = 255

    # 이미 방문했음을 표시한다
    visited.append((i, j))

    # (I, j)에 연결된 8가지 방향을 모두 검사하여 weak가 있다면 재귀적으로 호출
    for ii in range(i-1, i+2) :
        for jj in range(j-1, j+2) :
            if (img[ii, jj] == 80) and ((ii, jj) not in visited) :
                dfs(img, res, ii, jj, visited)
```

```
H, W = img.shape
res = np.zeros((H, W))
visited = []

for i in range(1, H - 1):
    for j in range(1, W - 1):                # 모든 픽셀(테두리 제외)에 대하여
        if (img[i][j] == 255):               # strong pixel이면
            dfs(img, res, i, j, visited)     # dfs 수행
```

Edge linking을 적용한 이미지

3. 결론

픽셀의 intensity의 정보를 가진 이미지에서 의미 있는 정보를 추출하기 위해 edge를 검출하였다. Canny edge detection은 1) Noise reduction, 2) gradient 크기와 방향 얻기, 3) Non-maximum suppression, 4) double Thresholding 5) linking의 과정으로 이루어진다. 이 과정을 통해 raw 이미지에서 edge의 정보를 구하였다.

결과는 다음과 같다.

원본 이미지

canny edge detection 이미지

Canny edge detection을 사용하여 이구아나의 이미지에서 edge를 검출하였다.