HW 03 – REPORT

소속 : 정보컴퓨터공학부
학번 : 201824633
이름 : 김유진

1. 서론

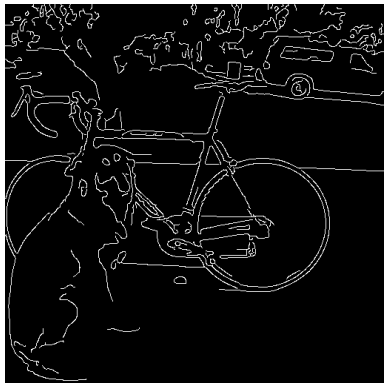
Canny Edge Detection

1. Filter Image with derivative of Gaussian
    A. 가우시안 블러를 이용해서 이미지를 흐리게 만든다.
2. Find magnitude and orientation of gradient
    A. gradient와 theta를 구한다. 이를 이용해 초기 edge를 찾는다.

3. Non-maximum suppression
    A. 2번에서 얻은 edge를 보다 sharp하게 만들어준다.

4. Thresholding and linking

   A. threshold 값보다 큰 픽셀만 출력을 해준다. 이때, threshold를 2개로 설정해 더 높은 threshold 값을 high, 낮은 threshold 값을 low라고 해준다. high보다 높을때와 high와 low 사이일 때를 분류해 linking을 해준다.



2. 본론

2-1. Noise reduction

a. load image

b. image to grayscale

c. gaussconvolve2d(array, sigma) sigma = 1.6

d. PIL -> show()

```
######### 1. Noise reduction

# Load iguana.bmp
iguana = Image.open('[HW03] Canny Edge Detection/iguana.bmp')
# Convert it to Greyscale
grey_iguana = iguana.convert('L')
# to numpy array
grey_iguana_array = np.asarray(grey_iguana, dtype=np.float32)
# blurs the image using gaussconvolved2d(array,sigma)
new_iguana_array = gaussconvolve2d(grey_iguana_array, 1.6)
# use PIL to show both the original and filtered images.
new_iguana = Image.fromarray(new_iguana_array)
iguana.show()
new_iguana.show()
```

〈좌〉 original 〈우〉 변형된 이미지

2-2. Finding the intensity gradient of the image : sobel_filter(img)

a. sobel filter 적용. => convolve2d

b. gradient 및 theta 구하기

```python
def sobel_filters(img):

    # apply Sobel filter in the x, y direction
    filter_x = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]
    filter_y = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
    # apply the convolve2d function to obtain the intensity x, y
value
    Ix = convolve2d(img, filter_x)
    Iy = convolve2d(img, filter_y)
    # Formulate Gradient and Theta
    #   G: Magnitude of gradient at each pixel in img.
    #   theta: Direction of gradient at each pixel in img.
    G = np.hypot(Ix, Iy)
    theta = np.arctan2(Iy, Ix)
    # mapping value => 0 ~ 255
    G_max = np.max(G)
    G_min = np.min(G)
    G = G / (G_max - G_min) * 255

    return (G, theta)
```

sobel filter 을 적용한 이미지

2-3. non maximum suppression

a. for thin edge

b. (0, 45, 90, 135)

c. theta : radian -> degree 변환

```python
def non_max_suppression(G, theta):
   # theta is radians. radian -> degree
    theta = np.rad2deg(theta)
    # res.shape = G.shape
    # if pixel is less than neighbor, that pixel is 0. => zeros
    res = np.zeros(G.shape)

    for i in range(1, G.shape[0]-1):
        for j in range(1, G.shape[1]-1):

            # angle standard = [0, 45, 90, 135] => 0 <= __ < 180
            angle = theta[i][j]
```

```
            if angle >= 180:
                angle -= 180
            if angle < 0:
                angle += 180

            # 중간 지점을 기준으로 angle 값이 가리키는 방향에 있는 픽셀
2개를 기준으로 더 큰 값이 있는지 없는지 비교한다.
            num1, num2 = 0, 0
            if 22.5 <= angle < 67.5: # 45
                num1 = G[i-1][j+1]
                num2 = G[i+1][j-1]
            elif 67.5 <= angle < 112.5: # 90
                num1 = G[i-1][j]
                num2 = G[i+1][j]
            elif 112.5 <= angle < 157.5: # 135
                num1 = G[i-1][j-1]
                num2 = G[i+1][j+1]
            else: # 0 : 157.5 ~ 180 / 0 ~ 22.5
                num1 = G[i][j-1]
                num2 = G[i][j+1]

            # num1 보다 크고 num2 보다 큰  경우 G[i][j]를 res[i][j]에
넣어준다.
            if (G[i][j] > num1 and G[i][j] > num2):
                res[i][j] = G[i][j]

    return res
```

non maximum suppression 적용 이미지

2-4. Double threshold

a. strong, weak and non-relevant

b. diff = max(image) − min(image)

c. T_high = min(image) + diff * 0.15

d. T_low = min(image) + diff * 0.03

```
def double_thresholding(img):

    # use the expressions to determine threshold values
    diff = np.max(img) - np.min(img)
    T_high = np.min(img) + diff * 0.15
    T_low = np.min(img) + diff * 0.03

    # default = no-relationship
    res = np.zeros(img.shape)
    # strong : 175 + 80 = 255
    res += np.where(img > T_high, 175, 0)
    # weak
    res += np.where(T_low < img, 80, 0)

    return res
```

double threshold 적용된 이미지

2-5. Edge Tracking by hysteresis

a. strong line에 연결된 weak edge를 strong으로 바꿔주고 그렇지않으면 edge에서 탈락시킨다.

b. DFS를 사용해야한다.

```
def hysteresis(img):

    strong = 255
    weak = 80

    # use dfs on all strong edges to obtain connected weak edges
    def dfs(i, j):
        # neighbor 8 pixels
        for a in range(i-1, i+2):
            for b in range(j-1, j+2):
                # weak 인 경우, strong 으로 바꿔주고 그 점을 기준으로 또 다시 dfs
                if img[a][b] == weak:
                    img[a][b] = strong
                    dfs(a, b)
                # 종료 조건 설정. strong 인 경우 탐색 중지
```

```
                elif img[a][b] == strong:
                    continue

    # weak edge -> strong [link]
    for i in range(1, img.shape[0] - 1):
        for j in range(1, img.shape[1] - 1):
            if img[i][j] == strong:
                dfs(i, j)

    # strong 인 경우만 strong value 그대로 복사, 아니면 0 으로
    res = np.where(img == strong, strong, 0)
    return res
```

최종결과 이미지

3. 결론

- 단계에 따른 이미지

Original

1

2

3

4

5