HW 03 – REPORT

소속 : 정보컴퓨터공학부

학번 : 202255595

이름 : 임영훈

1. 서론

이번 HW의 목적은 Canny edge detection을 해보는 것이다. Canny edge detection은 여러 단계로 구성되어 있다.

1단계는 noise를 제거해준다. 노이즈 제거에는 HW2에서 사용했던 gaussian convolution을 사용하여 제거할 수 있다.

2단계는 gradient 값의 크기와 방향을 구해주는 것이다. Gradient는 x축 방향과 y 축방향으로 구할 수 있는데 Sobel filter를 사용하면 각 방향의 gradient를 획득할 수 있다. 이렇게 구한 각 방향의 gradient 값을 합쳐, edge로 추정되는 부분과 방향을 구한다.

3단계는 non-maximum-suppression 단계이다. 2단계를 통해 구한 edge는 크기가 꽤 두껍기 때문에 local-maximum 값을 구하여 그 값을 제외한 나머지 값들은 0으로 만들어주어 edge의 두께를 얇게 해준다.

4단계는 thresholding 단계이다. Intensity 값을 이용해 strong edge, weak edge, no edge의 3 단계로 구분한다.

5단계는 linking 과정이다. Strong edge와 연결된 weak edge는 strong edge로 바꾸고 그렇지 않은 weak edge는 no edge로 바꿔준다.

이렇게 5단계를 걸치면 꽤 괜찮은 edge 이미지를 얻을 수 있다. 최종 결과는 gaussian filter의 sigma 값과 thresholding 시의 임계 값을 어떻게 정하는가에 따라 달라진다.

2. 본론

1. reduce_noise

```
def reduce_noise(img):
    """ Return the gray scale gaussian filtered image with sigma=1.6…
    grayscale_img = img.convert("L")   # grayscale로 변경
    grayscale_array = np.asarray(grayscale_img)     # array 형식으로 변경

    filtered_array = gaussconvolve2d(grayscale_array, 1.6)  # filter 적용
    res = filtered_array.astype(np.float32) # np.float32 형식으로 변경

    filtered_img = Image.fromarray(res)     # array -> 이미지로 변경

    img.show()          # 원본 이미지
    filtered_img.show() # filtered 이미지

    return res
```

샘플로 받은 이구아나 이미지에 대해 위 함수를 적용하면

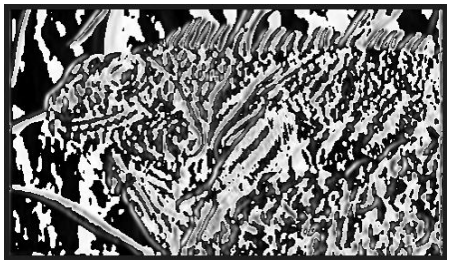
->

위와 같은 이미지를 얻을 수 있다.

2. sobel_filter

```python
def sobel_filters(img):
    """ Returns gradient magnitude and direction of input img.···
    grayscale_array = np.asarray(img)   # img -> array
    X_filter = np.array([[1, 0, -1],[2, 0, -2],[1, 0, -1]]) # Sobel X filter
    Y_filter = np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]]) # Sobel Y filter

    X_gradient_array = convolve2d(grayscale_array, X_filter)    # X 축 방향 gradient
    Y_gradient_array = convolve2d(grayscale_array, Y_filter)    # Y 축 방향 gradient

    G = np.hypot(X_gradient_array, Y_gradient_array)        # X, Y gradient 합치기, sqrt(x1**2 + y1**2), element-wise
    theta = np.arctan2(Y_gradient_array, X_gradient_array)  # theta 값 구하기

    G = G * 255 / np.max(G) # gradient 값을 0~255 사이로 만들어 주기 위해

    X_gradient_img = Image.fromarray(X_gradient_array.astype('uint8'))  # 이미지로 변환
    Y_gradient_img = Image.fromarray(Y_gradient_array.astype('uint8'))  # 이미지로 변환
    G_img          = Image.fromarray(G.astype('uint8'))                 # 이미지로 변환
    theta_img      = Image.fromarray(theta.astype('uint8'))             # 이미지로 변환

    X_gradient_img.show()
    Y_gradient_img.show()
    G_img.show()
    theta_img.show()

    return (G, theta)
```

위에서 얻은 노이즈가 제거된 흑백 이구아나 이미지에 sobel_filter 를 적용하면



X-gradient 이미지

+

Y-gradient 이미지

=>

edge-gradient 이미지

 theta 이미지

위와 같은 이미지들을 얻을 수 있다.

3. non-max-supression

```python
def non_max_suppression(G, theta):
    """ Performs non-maximum suppression.…
    height = G.shape[0]
    width = G.shape[1]

    res = np.zeros((height, width))   # NMS 이미지 array

    # edge 영역은 제외하고 진행
    for y in range(1, height-1):
        for x in range(1, width-1):
            degree = theta[y][x] * 180/ np.pi    # radin -> degree 단위 변환

            case = math.floor((degree + 180 + 22.5) / 45) % 4    # case 구하기
            comp_num1 = 0
            comp_num2 = 0

            # 왼쪽 위부터 (0,0)이고 밑으로 갈 수록 y 값이 증가하고 오른쪽으로 갈수록 x 값이 증가함에 유의!!!
            # case 1, case 3 헷갈리지 않게 조심하기
            if(case == 0):  # 좌, 우와 비교
                comp_num1 = G[y][x-1]
                comp_num2 = G[y][x+1]
            elif(case == 1):    # 오른쪽 위 대각선 방향과 비교
                comp_num1 = G[y-1][x+1]
                comp_num2 = G[y+1][x-1]
            elif(case == 2):    # 위, 아래와 비교
                comp_num1 = G[y-1][x]
                comp_num2 = G[y+1][x]
            elif(case == 3):    # 왼쪽 위 대각선 방향과 비교
                comp_num1 = G[y-1][x-1]
                comp_num2 = G[y+1][x+1]

            if((comp_num1 < G[y][x]) and (comp_num2 < G[y][x])):    # local-max 값이면
                res[y][x] = G[y][x]
            else:                                                   # local-max 아닌 경우
                res[y][x] = 0

    return res
```

Non-max-suppression 시, 기존의 (x, y) 좌표계가 아니라 아래로 갈수록 y축 값이 커지는 좌표계를 생각해야 한다. 이를 유의하여 코드를 작성하고 gradient 이미지에 NMS를 적용하면

이와 같이 edge의 크기가 줄어든 이미지를 얻을 수 있다.

4. double_thresholding

```python
def double_thresholding(img):
    """ ...
    min = np.min(img)   # min 값
    max = np.max(img)   # max 값
    diff = max - min
    T_high = min + diff * 0.15
    T_low = min + diff * 0.03

    res = img.copy()
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            intensity = img[y][x]
            if(intensity < T_low):      # no edge, 0
                res[y][x] = 0
            elif(intensity < T_high):   # weak edge, 80
                res[y][x] = 80
            else:                       # strong edge, 255
                res[y][x] = 255

    return res
```

1. no edge

2. Weak edge

3. Strong edge

이렇게 3가지로 분류를 하면

왼쪽 과 같은 이미지를 얻을 수 있다

5. hysteresis

```
def hysteresis(img):
    """ Find weak edges connected to strong edges and link them. ···
    width = img.shape[1]
    height = img.shape[0]

    res = np.zeros((height, width))
    visited = []    # dfs시 방문한 pixel의 위치 정보 저장

    for y in range(1, height-1):
        for x in range(1, width-1):
            visited.append((y, x))
            if(img[y, x] == 255):
                dfs(img, res, y, x, visited)

    return res
```

마지막으로 strong edge와 weak edge를 연결해주면

Edge 이미지를 얻을 수 있다.

3. 결론

이미지를 smooth 하게 만들고 gradient 값을 계산하고 추가적인 기법을 통하여 edge 이미지를 만드는 것에 성공하였다. 하지만 노이즈가 없는 이미지에 blur 처리를 하는 과정이 꼭 필요하지 않은 것 같아 gaussian filter를 빼고 edge 이미지를 만들어 보았다

하지만 detail한 edge 부분이 너무 많이 살아남아 edge 이미지라고 할 수는 없을 것 같다 따라서 이미지에 gaussian filter를 적용하는 과정이 필요하다는 것을 알았다..

또한 sigma 값을 3으로 변경하면 결과적으로

이와 같은 이미지를 얻을 수 있다.

sigma=1.6 과 sigma=3 인 이미지를 비교하면 sigma=3 인 이미지는 detail 한 edge가 제거된 것을 알 수 있다. 또한 thresholding 방법도 다르게 하면 이미지의 모습이 달라질 것이다. 이와 같은 factor들을 잘 선택하여 canny edge detection을 하면 괜찮은 edge 이미지를 얻을 수 있을 것 같다.