HW 03 - REPORT

소속 : 정보컴퓨터공학부
학번 : 202155592
이름 : 이지수

서론

실습 목표

1. Canny Edge Detection의 각 단계를 이해하고 이를 코드로 작성할 수 있다.
    a. blur 처리된 흑백 이미지를 생성할 수 있다.
    b. sobel filter가 적용된 이미지를 생성할 수 있다.
    c. Non-maximum supression이 적용된 이미지를 생성할 수 있다.
    d. Double threshold가 적용된 이미지를 생성할 수 있다.
    e. hysteresis로 edge tracking을 할 수 있다.

이론적 배경

1. Edge detection

- high-pass filter(=edge처럼 high frequency 성분만 남기는 것)를 적용한 것이라고 볼 수 있다.
- 필요성: Resilience to lighting and color

  ⇒ useful for recognition, matching patches across images
- edge는 image intensity function에서 급격한 변화가 일어나는 peak이다.

2. Image gradient

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

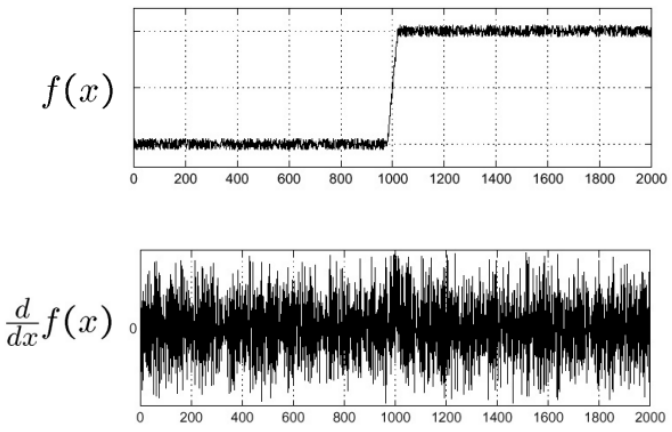
- gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

- gradient direction

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

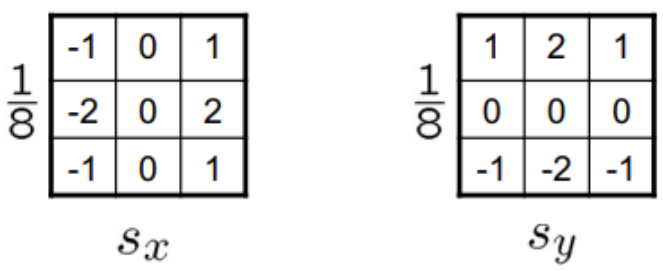
3. Effects of noise



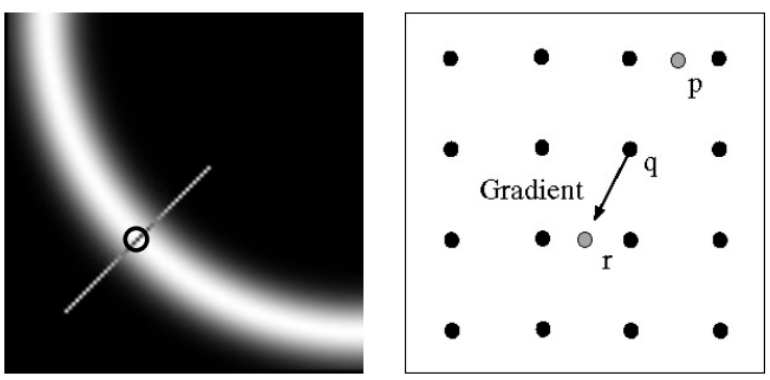
위 사진처럼 peak 지점이 너무 많을 경우 어떻게 edge 검출을 해야 될까? low-pass filter를 적용하여 smoothing 하면 noise를 줄일 수 있다.

4. The Sobel operator

- 중심 픽셀의 차분(difference) 비중을 두 배로 준 필터
- Common approximation of derivative of Gaussian



5. Non-maximum supression



- Check if pixel is **local maximum** along gradient direction
- 주변 픽셀과 비교했을 때 픽셀이 local maximum이라면 edge이고 local maximum이 아니라면 edge가 아니다.
- NMS를 적용하면 더 정교하고 얇은 edge를 검출할 수 있다.

6. Double thresholding

- R > T: strong edge
- R < T but R > t: weak edge
- R < t: no edge

7. Hysteresis thresholding

- Strong edges are edges
- Weak edges are edges iff they connect to strong
- Look in some neighborhood (usually 8 closest)

8. Canny edge detector
    a. Filter image with derivative of Gaussian
    b. Find magnitude and orientation of gradient
    c. Non-maximum suppression
    d. Thresholding and linking (hysteresis)

본론

1. Noise reduction

```
def reduce_noise(img):
    """ Return the gray scale gaussian filtered image with sigma=1.6
    Args:
        img: RGB image. Numpy array of shape (H, W, 3).
    Returns:
        res: gray scale gaussian filtered image (H, W).
    """
    # greyscale 이미지로 변환
    im_grey = img.convert('L')
    im_array = np.asarray(im_grey, dtype=np.float32)

    # 'gaussconvolve2d' 함수로 filter 생성 후 convolution 연산 수행
    im_after_convolution = gaussconvolve2d(im_array, 1.6)

    # float를 uint8로 다시 변환 후 res에 저장
    res = im_after_convolution.astype(np.uint8)

    # 결과를 다시 PIL 이미지로 변환한 후 저장
    im_filtered = Image.fromarray(res)

    # 원본 이미지와 블러링된 이미지 보여주기
    img.show()
    im_filtered.show()

    return res
```

Canny Edge Detection을 적용하기 앞서 이미지를 blur 처리하여 noise를 줄여주었다. `HW 02`에서 blur 이미지를 생성하는 코드와 같다. 1번 문제의 결과 이미지는 아래와 같다.

iguana_blurred.bmp

2. Finding the intensity gradient of the image

```
def sobel_filters(img):
    """ Returns gradient magnitude and direction of input img.
    Args:
        img: Grayscale image. Numpy array of shape (H, W).
    Returns:
        G: Magnitude of gradient at each pixel in img.
            Numpy array of shape (H, W).
        theta: Direction of gradient at each pixel in img.
            Numpy array of shape (H, W).
    Hints:
        - Use np.hypot and np.arctan2 to calculate square root and arctan
    """
    # X, Y Sobel filter
    x_filter = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    y_filter = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    # x, y 방향으로 Sobel filter 적용
    # intensity x, y 값 얻기 위해 convolve2d 함수 사용
    Ix = convolve2d(img, x_filter)
    Iy = convolve2d(img, y_filter)

    # Magnitude of gradient 구하기
    # np.hypot(x1, x2)는 sqrt(x1^2 + x2^2)와 동일
    G = np.hypot(Ix, Iy)
    # G를 0에서 255 사이 값으로 매핑
    G = G / G.max() * 255

    # Direction of gradient 구하기
    theta = np.arctan2(Iy, Ix)

    return (G, theta)
```

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

X filter

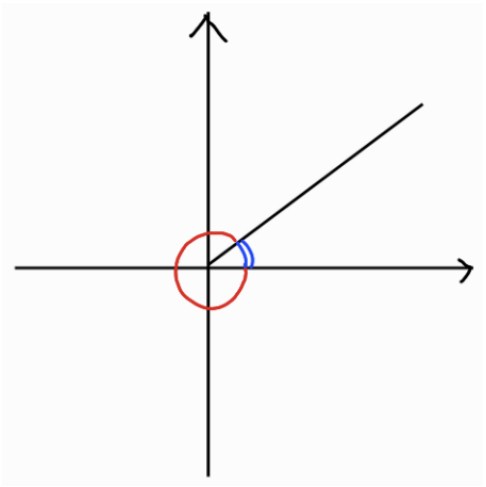
-1	-2	-1
0	0	0
+1	+2	+1

Y filter

convolution 연산을 할 때 `pdf` 파일에 나와있는 sobel filter를 적용하여 x, y의 intensity 값을 구했다.

$$G = \sqrt{I_x^2 + I_y^2}\ ,\ \theta = \tan^{-1}\frac{I_y}{I_x}$$

`Gradient`와 `Theta` 값을 얻기 위해 위 공식을 사용했고 `Gradient`의 경우 0에서 255 사이의 값으로 매핑을 해주었다.



`Theta` 값을 구할 때 `arctan`가 아닌 `arctan2`를 사용하는 이유는 위의 상황에서 더 작은 각(위 사진에서 파란색으로 표시)을 구하기 위해서이다. 이로써 edge의 방향을 더 정확하게 구할 수 있다.

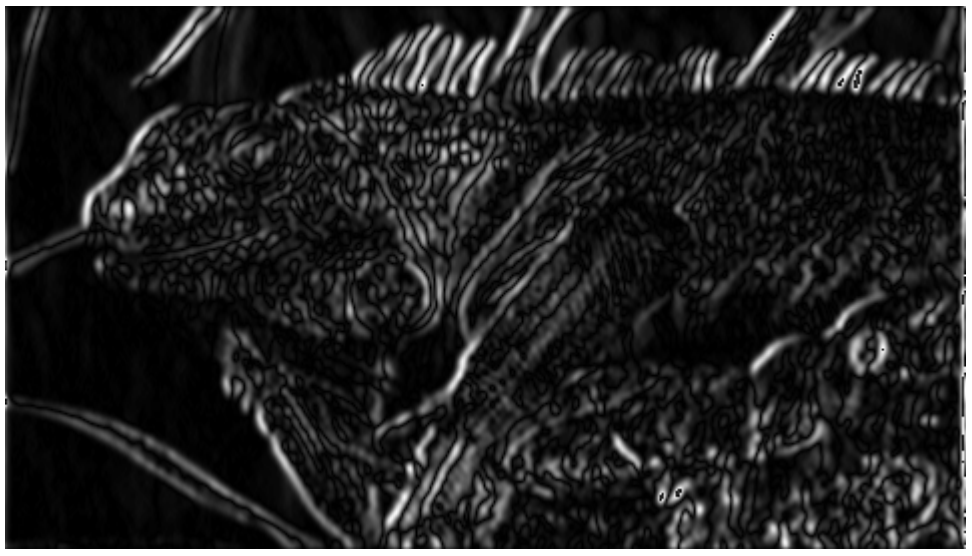
```
[[-1.5437759 -1.5532542 -1.5551726 ... -1.567395  -1.5668901 -1.5646988]
 [-1.5083776 -1.5291537 -1.5337762 ... -1.5620246 -1.5605927 -1.5549246]
 [-1.4601392 -1.4940244 -1.4994888 ... -1.5546687 -1.5515679 -1.5405025]
 ...
 [-1.5042281 -1.5323538 -1.541393  ...  1.5636536  1.5620246  1.5563046]
 [-1.3258177 -1.4464413 -1.487655  ...  1.5641739  1.5631042  1.5587487]
 [ 1.5485778  1.5572836  1.5599272 ...  1.5662302  1.5661235  1.5640397]]
```

arctan을 사용했을 때 theta 값 출력

```
[[-0.7853982  -1.3465616  -1.5083776  ... -1.665748   -1.8231143
  -2.362255  ]
 [-0.26916748 -0.87605804 -1.3171222  ... -1.9543239  -2.3761919
  -2.901197  ]
 [-0.13352905 -0.5144513  -1.0516502  ... -2.3053908  -2.726518
  -3.0301914 ]
 ...
 [-0.23374318 -0.68231654 -0.95613337 ...  1.5707964   2.1341126
   2.9024425 ]
 [-0.05549851 -0.20749623 -0.38050637 ...  1.4586856   1.8832879
   2.7823262 ]
 [ 0.6857295   1.1856388   1.315614   ...  1.4842551   1.654711
   2.2991138 ]]
```

arctan2를 사용했을 때 theta 값 출력

2번 문제의 결과 이미지는 아래와 같다.



x_axis_gradient.bmp

y_axis_gradient.bmp

iguana_sobel_gradient.bmp

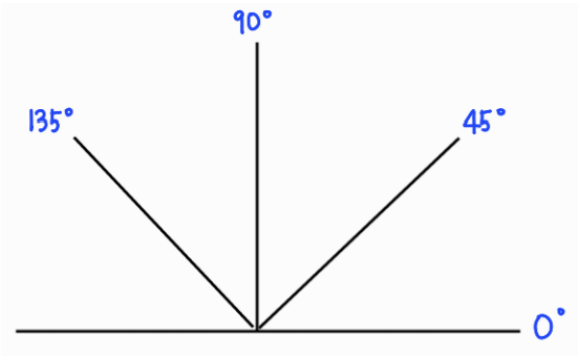
iguana_sobel_theta.bmp

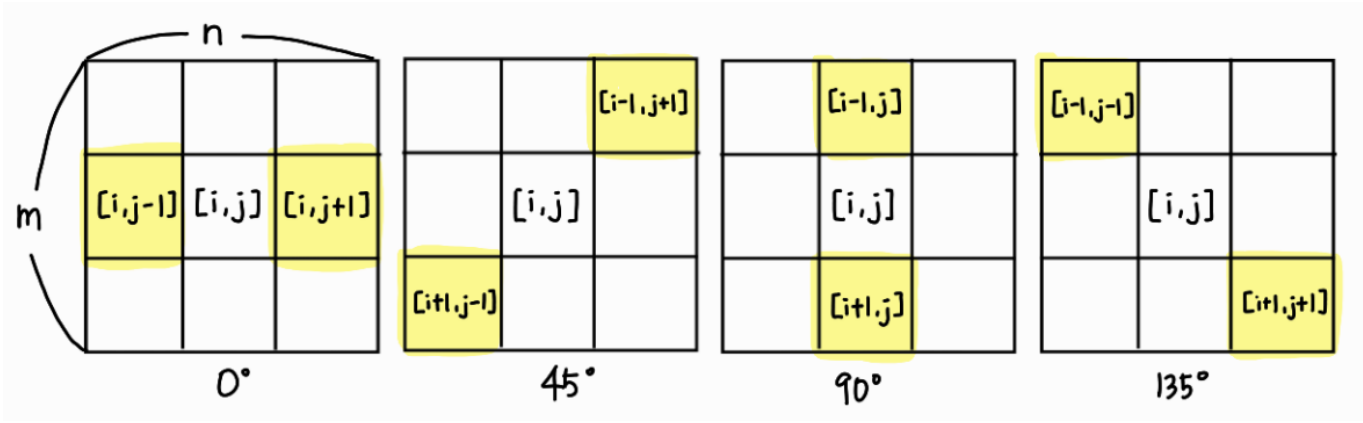
3. Non-Maximum Suppression

```
def non_max_suppression(G, theta):
    """ Performs non-maximum suppression.
    This function performs non-maximum suppression along the direction
    of gradient (theta) on the gradient magnitude image (G).
    Args:
        G: gradient magnitude image with shape of (H, W).
        theta: direction of gradients with shape of (H, W).
    Returns:
        res: non-maxima suppressed image.
    """
    # 라디안을 도(degree)로 바꾸기
    angle = theta * (180 / np.pi)
    # sobel filter에서 arctan2 사용했기 때문에 angle은 -180도에서 180도 사이의 값을 가짐
    # angle이 음수라면 180 더해주기
```

```
    angle = np.where(angle < 0, angle + 180, angle)

    # G의 행의 개수, 열의 개수 확인
    m, n = G.shape
    # non-maximum suppression을 적용한 이미지인 res 초기화
    res = np.zeros((m, n), dtype=np.float32)

    direction = np.array([0, 45, 90, 135])

    # 제일 끝(edge)은 이웃 픽셀이 충분하지 않으므로 NMS 연산 적용하지 않음
    for i in range(1, m - 1):
        for j in range(1, n - 1):
            # angle이 0, 45, 90, 135 중 어디에 가장 가까운지 찾기
            # 이때 argmin()은 direction의 특정 index 반환
            closest_direction =  direction[np.abs(direction - angle[i, j]).argmin()]

            # direction이 0도라면 (왼쪽 픽셀) vs (현재 픽셀), (오른쪽 픽셀) vs (현재 픽셀)의 G(gradien
            if (closest_direction == 0):
                pixel1 = G[i, j - 1]
                pixel2 = G[i, j + 1]
            # direction이 45도라면
            elif (closest_direction == 45):
                pixel1 = G[i - 1, j + 1]
                pixel2 = G[i + 1, j - 1]
            # direction이 90도라면
            elif (closest_direction == 90):
                pixel1 = G[i - 1, j]
                pixel2 = G[i + 1, j]
            # direction이 135도라면
            elif (closest_direction == 135):
                pixel1 = G[i - 1, j - 1]
                pixel2 = G[i + 1, j + 1]

            # 현재 pixel의 G가 두 이웃 픽셀의 G보다 크거나 같다면 값 유지
            if (pixel1 <= G[i, j]) and (pixel2 <= G[i, j]):
                res[i, j] = G[i, j]
            # 현재 pixel의 G가 두 이웃 픽셀의 G보다 작다면 값을 0으로 만들어 중요하지 않은 edge 삭제
            else:
                res[i, j] = 0

    return res
```

라디안을 60분법으로 바꾼 뒤 음수라면 180을 더해서 양수로 만들어 주고 이를 `angle` 변수에 저장한다. `angle`의 값들이 0, 45, 90, 135 중 어디에 가장 가까운지 구하기 위해 두 값의 차가 가장 적은 `direction`의 index를 찾고 index에 해당하는 값(0, 45, 90, 135 중 하나)을 `closest_direction`에 저장했다.





`closest_direction`이 0도, 45도, 90도, 135도일 때 비교해야 하는 두 개의 픽셀(위 사진에서 노란색으로 표시)의 `G` 값을 `pixel1`, `pixel2`에 저장한다. 이때 제일 테두리에 있는 픽셀은 비교할 이웃 픽셀이 충분하지 않으므로 이 작업을 진행하지 않는다.

`pixel1`, `pixel2`와 비교했을 때 `G` 값이 local maximum이라면 그 값을 유지하고, 아니라면 0으로 만들어 중요하지 않은 edge를 삭제한다. 3번 문제의 결과 이미지는 아래와 같다.

iguana_non_max_suppression.bmp

4. Double threshold

```
# NMS 연산으로 나온 픽셀들을 strong, weak, non-relevant 총 세 가지 type의 edge로 구분
def double_thresholding(img):
    """
    Args:
        img: numpy array of shape (H, W) representing NMS edge response.
    Returns:
        res: double_thresholded image.
    """
    diff = np.max(img) - np.min(img)
    T_high = np.min(img) + diff * 0.15
    T_low = np.min(img) + diff * 0.03

    # res 초기화
    res = np.zeros(img.shape, dtype=np.uint8)
    # T_high보다 크면 strong edge
    res = np.where(img > T_high, 255, res)
    # T_low와 T_high 사이면 weak edge
    res = np.where((T_low <= img) & (img <= T_high), 80, res)
    # T_low보다 작으면 non-relevant edge
    res = np.where(img < T_low, 0, res)

    return res
```

4번에서는 NMS 연산으로 나온 픽셀들을 strong, weak, non-relevant 총 세 가지 type의 edge로 구분하는 작업을 했다. `pdf`에 나와 있는 공식을 적용하여 `diff`, `T_high`, `T_low`를 구한 후 `T_high`보다 크면 **255**(strong edge), `T_low`와 `T_high` 사이면 **80**(weak

edge), `T_low`보다 작으면 **0**(non-relevant edge)으로 만들었다. 이를 통해 NMS 연산을 했을 때보다 훨씬 더 정교하게 edge를 검출할 수 있다. 4번 문제의 결과 이미지는 아래와 같다.

iguana_double_thresholding.bmp

5. Edge Tracking by hysteresis

```
# 픽셀을 tracking하여 실제 edge 라인 형성
def hysteresis(img):
    """ Find weak edges connected to strong edges and link them.
    Iterate over each pixel in strong_edges and perform depth first
    search across the connected pixels in weak_edges to link them.
    Here we consider a pixel (a, b) is connected to a pixel (c, d)
    if (a, b) is one of the eight neighboring pixels of (c, d).
    Args:
        img: numpy array of shape (H, W) representing NMS edge response.
    Returns:
        res: hysteresised image.
    """
    m, n = img.shape
    res = np.copy(img)
    visited = [] # 방문한 픽셀의 좌표 저장

    for i in range(m):
        for j in range(n):
            # strong edge라면
            if img[i, j] == 255 and (i, j) not in visited:
                # 방문 표시
                visited.append((i, j))
                # 이웃 픽셀 중에 값이 80인 픽셀이 있다면 255로 바꾸고, 재귀적으로 같은 과정 반복
                dfs(img, res, i, j, visited)

    # weak edge 제거
    res = np.where(res == 80, 0, res)

    return res
```

4번에서 구한 것을 바탕으로 hysteresis를 통해 더 정교하게 edge를 검출할 수 있다. 반복문을 돌면서 만약 픽셀의 값이 255이면서 아직 방문하지 않았다면 방문 표시를 한 후 `dfs` 함수를 호출한다. 해당 픽셀의 이웃 픽셀 중에 값이 80인 픽셀이 있다면(= weak edge가 strong edge와 연결되어 있다면) 값을 255로 바꾸고 재귀적으로 같은 과정을 반복한다. 5번 문제의 결과 이미지는 아래와 같다.

iguana_hysteresis.bmp

결론

Canny Edge Detection의 과정은 아래와 같다.

1. Noise reduction

   원본 이미지를 grayscale로 변환하고 blur 처리하여 noise를 줄인다.

2. Find magnitude and orientation of gradient

   x, y 각각의 방향에 대해 convolution 연산을 해야 하는데, 이때 sobel filter를 적용하여 intensity 값을 얻는다. magnitude는 `np.hypot(Ix, Iy)`로, direction은 `np.arctan2(Iy, Ix)`로 구할 수 있다.

3. Non-maximum suppression

   2번 단계에서 구한 direction이 0도, 45도, 90도, 135도 중 어디에 가장 가까운지 찾고, 각도에 따라 비교할 두 개의 픽셀을 정한다. 해당 픽셀의 magnitude가 두 개의 픽셀의 magnitude보다 크다면 local maximum이므로 edge라고 볼 수 있다. 그렇지 않다면 edge라고 볼 수 없으므로 값을 0으로 만든다.

4. Thresholding and linking (hysteresis)

   두 개의 threshold(low, high)를 정하여 NMS 연산으로 나온 픽셀들을 strong, weak, non-relevant 총 세 가지 type의 edge로 구분한다. edge로 볼 수 있는 것은 strong edge, weak edge 중 strong edge와 연결되어 있는 edge이다. 이는 DFS로 찾을 수 있다.