Homework 9　　R08921005 黃國郡

Answer

Lena.bmp

Robert's Operator
Thresholds:25:

Prewitt's Edge Detector
Thresholds:100

Sobel's Edge Detector
Thresholds:100:

Frei and Chen's Gradient Operator
Thresholds:100

Kirsch's Compass Operator
Thresholds:300

Robinson's Compass Operator
Thresholds:100

Nevatia-Babu 5x5 Operator
Thresholds:15000

Homework 9 R08921005 黃國郡

Description& Algorithm

Implement following edge detectors with thresholds :

(a) Robert's Operator: 25

Roberts operators: two 2×2 masks to calculate gradient

$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$

$f'(x) \approx f(x+1) - f(x)$

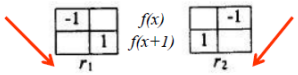


Figure 7.21 Masks used for the Roberts operators.

gradient magnitude: $\sqrt{r_1^2 + r_2^2}$

Where r_1, r_2 are values from first, second masks respectively.

(b) Prewitt's Edge Detector: 100

Prewitt edge detector: two 3×3 masks in row, column directions.

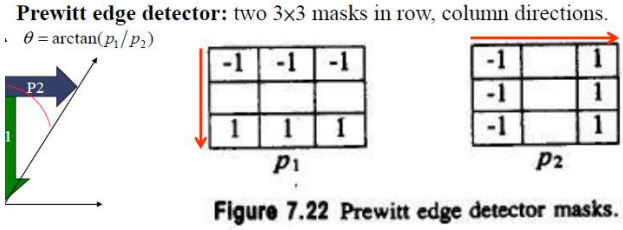


Figure 7.22 Prewitt edge detector masks.

Gradient magnitude: $g = \sqrt{p_1^2 + p_2^2}$

Gradient direction: $\theta = arctan(p_1/p_2)$ clockwise w.r.t. column axis

Where p_1, p_2 are values from first, second masks respectively.

(c) Sobel's Edge Detector: 100

Sobel edge detector: two 3×3 masks in row, column directions.

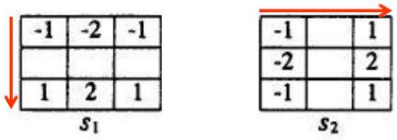


Figure 7.23 Sobel edge detector masks.

Gradient magnitude: $g = \sqrt{s_1^2 + s_2^2}$

Gradient direction: $\theta = arctan(s_1/s_2)$ clockwise w.r.t. column axis

Where s_1, s_2 are values from first, second masks respectively.

(d) Frei and Chen's Gradient Operator: 100

Frei and Chen edge detector: two in a set of nine orthogonal masks (3×3).

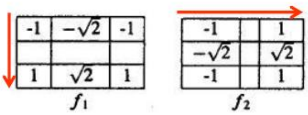


Figure 7.24 Frei and Chen gradient masks.

Gradient magnitude: $g = \sqrt{f_1^2 + f_2^2}$

Gradient direction: $\theta = arctan(f_1/f_2)$ clockwise w.r.t. column axis

Where f_1, f_2 are values from first, second masks respectively.

(e) Kirsch's Compass Operator: 300

Kirsch edge detector : set of eight compass template edge masks.

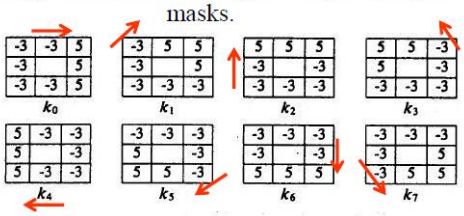
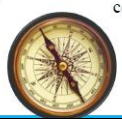


Figure 7.25 Kirsch compass masks.



compass

Gradient magnitude: $g = \max_{n, n=0,\ldots,7} k_n$

Gradient direction: $\theta = 45^\circ \text{ argmax } k_n$

(f) Robinson's Compass Operator: 100

Robinson edge detector : compass template mask set with only 0, ±1, ±2.

Simplify mask coefficients (Kirsch)

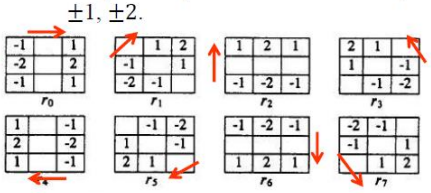


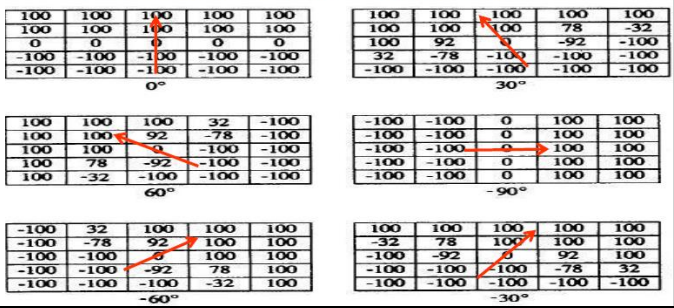
Figure 7.26 Robinson compass masks.

Done by only four masks since negation of each mask is also a mask.

Gradient magnitude and direction same as Kirsch operator.

(g) Nevatia-Babu 5x5 Operator: 15000

Nevatia and Babu: set of six 5×5 compass template masks.



Homework 9　　R08921005 黃國郡

Code description

本次作業使用 python3.7，IDE 使用 Spyder

(a) Main function:

1. 先建立好每一種會用到 neighbor(2*2,3*3,5*5)
2. 將 lena 當作 input 給每一個 operator
3. 將每個 output 存起來

```python
if __name__ == '__main__':
    neighbor_2 = createNeighbor(0, 2)
    neighbor_3 = createNeighbor(-1, 2)
    neighbor_5 = createNeighbor(-2, 3)

    lena = cv2.imread('Lena.bmp', cv2.IMREAD_GRAYSCALE)

    robert = robert_operator(lena, 25)
    cv2.imwrite('Robert.bmp', robert)

    prewitt = prewitt_operator(lena, 100)
    cv2.imwrite('Prewitt.bmp', prewitt)

    sobel = sobel_operator(lena, 100)
    cv2.imwrite('Sobel.bmp', sobel)

    frei_chen = frei_chen_operator(lena, 100)
    cv2.imwrite('Frei and Chen.bmp', frei_chen)

    kirsch = kirsch_operator(lena, 300)
    cv2.imwrite('Kirsch.bmp', kirsch)

    robinson = robinson_operator(lena, 100)
    cv2.imwrite('Robinson.bmp', robinson)

    nevatia_babu = nevatia_babu_operator(lena, 15000)
    cv2.imwrite('Nevatia-Babu.bmp', nevatia_babu)
```

(b) createNeighbor:

給定上下屆，就可以取出區塊的位置標號

```python
def createNeighbor(start, end):
    x = list(range(start, end))
    return list(itertools.product(x, repeat=2))
```

(c) Robert:

1.每個 pixel 的 neighbor 對應 2*2 的 mask ，

2.乘上 mask 上的值並加總

3.最後 mask 的值分別平方開根號

4.得 gradient magnitude。

5.if magnitude < threshold →改成 255

```python
def robert_operator(img, threshold):
    r, c = img.shape
    robert_img = np.zeros(img.shape, dtype=np.uint8)
    r1_mask = [-1, 0, 0, 1]
    r2_mask = [0, -1, 1, 0]
    for i in range(r):
        for j in range(c):
            candidate = []
            for m, n in neighbor_2:
                if 0 <= i+m < r and 0 <= j+n < c:
                    candidate.append(img[i+m][j+n])
                else:
                    candidate.append(0)
            r1 = 0
            r2 = 0
            for k in range(len(candidate)):
                r1 += int(candidate[k]) * r1_mask[k]
                r2 += int(candidate[k]) * r2_mask[k]
            value = math.sqrt((r1**2) + (r2**2))
            if (value < threshold):
                robert_img[i][j] = 255
    return robert_img
```

(d) Prewitt、Sobel、Frei and Chen:

1. 不同於 Robert’s operator 的地方方在於這三個是用 3*3 的 mask 而 Robert’s operator 是用 2*2 的 mask
2. 這三個 operator 彼此之間 mask 權重不同，可以得到不同的結果。

```python
def frei_chen_operator(img, threshold):
    r, c = img.shape
    frei_chen_img = np.zeros(img.shape, dtype=np.uint8)
    number = math.sqrt(2)
    f1_mask = [-1, -number, -1,
               0,       0,  0,
               1,  number,  1]
    f2_mask = [-1, 0,      1,
               -number, 0, number,
               -1, 0,      1]

    for i in range(r):
        for j in range(c):
            candidate = []
            for m, n in neighbor_3:
                if 0 <= i+m < r and 0 <= j+n < c:
                    candidate.append(img[i+m][j+n])
                else:
                    candidate.append(0)
            f1 = 0
            f2 = 0
            for k in range(len(candidate)):
                f1 += int(candidate[k]) * f1_mask[k]
                f2 += int(candidate[k]) * f2_mask[k]
            value = math.sqrt((f1**2) + (f2**2))
            if (value < threshold):
                frei_chen_img[i][j] = 255
    return frei_chen_img
```

```python
def prewitt_operator(img, threshold):
    r, c = img.shape
    prewitt_img = np.zeros(img.shape, dtype=np.uint8)
    p1_mask = [-1, -1, -1, 0, 0, 0, 1, 1, 1]
    p2_mask = [-1, 0, 1, -1, 0, 1, -1, 0, 1]
    for i in range(r):
        for j in range(c):
            candidate = []
            for m, n in neighbor_3:
                if 0 <= i+m < r and 0 <= j+n < c:
                    candidate.append(img[i+m][j+n])
                else:
                    candidate.append(0)
            p1 = 0
            p2 = 0
            for k in range(len(candidate)):
                p1 += int(candidate[k]) * p1_mask[k]
                p2 += int(candidate[k]) * p2_mask[k]
            value = math.sqrt((p1**2) + (p2**2))
            if (value < threshold):
                prewitt_img[i][j] = 255
    return prewitt_img
```

```python
def sobel_operator(img, threshold):
    r, c = img.shape
    sobel_img = np.zeros(img.shape, dtype=np.uint8)
    s1_mask = [-1, -2, -1,  0, 0, 0,  1, 2, 1]
    s2_mask = [-1,  0,  1, -2, 0, 2, -1, 0, 1]
    for i in range(r):
        for j in range(c):
            candidate = []
            for m, n in neighbor_3:
                if 0 <= i+m < r and 0 <= j+n < c:
                    candidate.append(img[i+m][j+n])
                else:
                    candidate.append(0)
            s1 = 0
            s2 = 0
            for k in range(len(candidate)):
                s1 += int(candidate[k]) * s1_mask[k]
                s2 += int(candidate[k]) * s2_mask[k]
            value = math.sqrt((s1**2) + (s2**2))
            if (value < threshold):
                sobel_img[i][j] = 255
    return sobel_img
```

(e) Kirsch、Robinson:

1. 用 3*3 的 mask 來做 edge detection
2. 用 8 個 3*3 的 mask，每個 mask 罩在每個 pixel 上都會有 gradient magnitude
3. 取 max(gradient magnitude)，和 threshold 做比較來來決定 pixel 值
4. <threshold 設 255

```python
def kirsch_operator(img, threshold):
    r, c = img.shape
    kirsch_img = np.zeros(img.shape, dtype=np.uint8)
    masks = [(-3, -3, 5, -3, 0, 5, -3, -3, 5),
             (-3, 5, 5, -3, 0, 5, -3, -3, -3),
             (5, 5, 5, -3, 0, -3, -3, -3, -3),
             (5, 5, -3, 5, 0, -3, -3, -3, -3),
             (5, -3, -3, 5, 0, -3, 5, -3, -3),
             (-3, -3, -3, 5, 0, -3, 5, 5, -3),
             (-3, -3, -3, -3, 0, -3, 5, 5, 5),
             (-3, -3, -3, -3, 0, 5, -3, 5, 5)]
    for i in range(r):
        for j in range(c):
            value = 0
            for mask in masks:
                candidate = []
                for m, n in neighbor_3:
                    if 0 <= i+m < r and 0 <= j+n < c:
                        candidate.append(img[i+m][j+n])
                    else:
                        candidate.append(0)
                g = 0
                for k in range(len(candidate)):
                    g += int(candidate[k]) * mask[k]
                value = max(value, g)
            if (value < threshold):
                kirsch_img[i][j] = 255

    return kirsch_img
```

```python
def robinson_operator(img, threshold):
    r, c = img.shape
    robinson_img = np.zeros(img.shape, dtype=np.uint8)
    masks = [(-1, 0, 1, -2, 0, 2, -1, 0, 1),
             (0, 1, 2, -1, 0, 1, -2, -1, 0),
             (1, 2, 1, 0, 0, 0, -1, -2, -1),
             (2, 1, 0, 1, 0, -1, 0, -1, -2),
             (1, 0, -1, 2, 0, -2, 1, 0, -1),
             (0, -1, -2, 1, 0, -1, 2, 1, 0),
             (-1, -2, -1, 0, 0, 0, 1, 2, 1),
             (-2, -1, 0, -1, 0, 1, 0, 1, 2)]
    for i in range(r):
        for j in range(c):
            value = 0
            for mask in masks:
                candidate = []
                for m, n in neighbor_3:
                    if 0 <= i+m < r and 0 <= j+n < c:
                        candidate.append(img[i+m][j+n])
                    else:
                        candidate.append(0)
                g = 0
                for k in range(len(candidate)):
                    g += int(candidate[k]) * mask[k]
                value = max(value, g)
            if (value < threshold):
                robinson_img[i][j] = 255

    return robinson_img
```

(f) Nevatia and Babu:

1. 用 6 個 5*5 的 mask 找出哪個 mask 對該 pixel 值有最大的 gradient magnitude
2. 拿該 magnitude 去和 threshold 做比較，
3. < threshold 設 255

```python
def nevatia_babu_operator(img, threshold):
    r, c = img.shape
    nevatia_babu_img = np.zeros(img.shape, dtype=np.uint8)
    masks = [(100, 100, 100, 100, 100, 100, 100, 100, 100, 0, 0, 0, 0, 0, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100),
             (100, 100, 100, 100, 100, 100, 100, 100, 78, -32, 100, 92, 0, -
             92, -100, 32, -78, -100, -100, -100, -100, -100, -100, -100, -100),
             (100, 100, 100, 92, -100, 100, 100, 32, -78, -100, 100, 100, 0, -
             100, -100, 100, 78, -92, -100, -100, 100, -32, -100, -100, -100),
             (-100, -100, 0, 100, 100, -100, -100, 0, 100, 100, -100, -100,
             0, 100, 100, -100, -100, 0, 100, 100, -100, -100, 0, 100, 100),
             (-100, 32, 100, 100, 100, -100, -78, 92, 100, 100, -100, -100, 0,
             -100, 100, -100, -100, -92, 78, 100, -100, -100, -100, 32, 100),
             (100, 100, 100, 100, 100, -32, 78, 100, 100, 100, -100, -92, 0, 92, 100, -100, -100, -100, -78, 32, -100, -100, -100, -100, -100)]
    for i in range(r):
        for j in range(c):
            value = 0
            for mask in masks:
                candidate = []
                for m, n in neighbor_5:
                    if 0 <= i+m < r and 0 <= j+n < c:
                        candidate.append(img[i+m][j+n])
                    else:
                        candidate.append(0)
                g = 0
                for k in range(len(candidate)):
                    g += int(candidate[k]) * mask[k]
                value = max(value, g)
            if (value < threshold):
                nevatia_babu_img[i][j] = 255

    return nevatia_babu_img
```