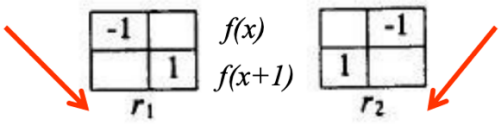
Homework 9

General Edge Detection

R09922063 鄭筠庭 資工所碩一

Code explanation:

(a) **Robert's Operator**: Extend the right and bottom borders by 1 pixel. Use a 4x4 mask to calculate the r1 and r2 values. If the gradient value is larger than the threshold, set the pixel to black (value = 0).

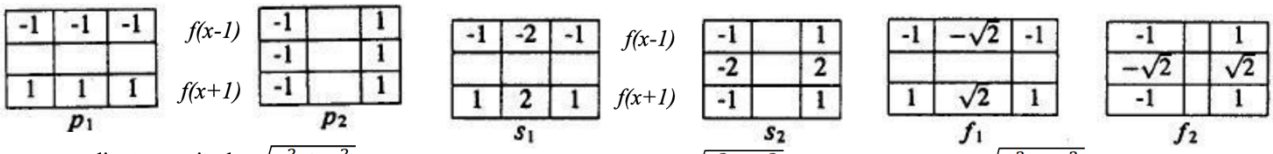


gradient magnitude: $\sqrt{r_1^2 + r_2^2}$

```
def Roberts(originImg, rows, columns, threshold):
    image = np.zeros(originImg.shape, np.uint8)
    imageBordered = cv2.copyMakeBorder(originImg, 0, 1, 0, 1, cv2.BORDER_REFLECT)

    for r in range(rows):
        for c in range(columns):
            r1 = int(imageBordered[r+1,c+1]) - int(imageBordered[r,c])
            r2 = int(imageBordered[r+1,c]) - int(imageBordered[r,c+1])
            gradient = math.sqrt(r1**2 + r2**2)
            if gradient >= threshold:
                image[r,c] = 0
            else:
                image[r,c] = 255
```

(b) Prewitt operator, (c) Sobel operator, (d) Frei and Chen gradient operator:

These three operators share the same function. The only difference is the mask they use to calculate the gradient magnitude. Extend the borders by 1 pixel. Use a 9x9 mask to calculate the s1 and s2 values.

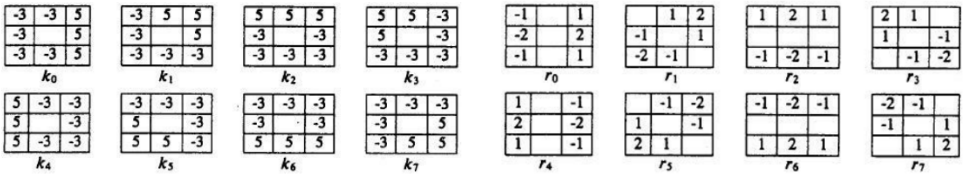


gradient magnitude: $\sqrt{p_1^2 + p_2^2}$

gradient magnitude: $\sqrt{s_1^2 + s_2^2}$

```
for r in range(rows):
    for c in range(columns):
        p1, p2 = 0, 0
        for maskR in range(maskRows):
            for maskC in range(maskColumns):
                p1 += int(imageBordered[r+maskR-1, c+maskC-1]) * mask_1[maskR][maskC]
                p2 += int(imageBordered[r+maskR-1, c+maskC-1]) * mask_2[maskR][maskC]
        gradient = math.sqrt(p1**2 + p2**2)
        if gradient >= threshold:
            image[r,c] = 0
        else:
            image[r,c] = 255
```

(e) Kirsch compass operator, (f) Robinson's Compass Operator:



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

gradient magnitude: $\max_{n,n=0,\ldots,7} r_n$

These two operators share the same function while using a different mask. The mask moves in a counter-clockwise pattern, and generate eight values for each mask. We find the maximum value and compare it with the threshold. If the gradient value is larger than the threshold, set the pixel to black (value = 0).

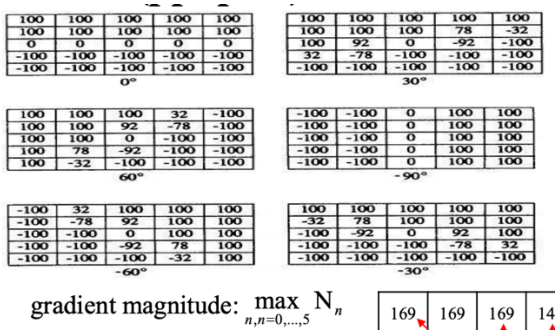
```
for r in range(rows):
    for c in range(columns):
        imageList = [borderImg[r-1,c-1], borderImg[r-1,c] , borderImg[r-1,c+1], borderImg[r,c+1] ,borderImg[r+1
            ,c+1], borderImg[r+1,c], borderImg[r+1,c-1] , borderImg[r,c-1]]
        maxK = 0
        for offset in range(8):
            tempSum = 0
            for num in range(8):
                tempSum += imageList[num] * mask[(num+offset)%8]
            if tempSum > maxK:
                maxK = tempSum

        if maxK >= threshold:
            image[r,c] = 0
        else:
            image[r,c] = 255
```

(e) Nevatia-Babu 5×5 operator:

The six masks are divided into two categories. 0° and 30° are the two main masks.

- 0° → rotate 90° clockwise → -90°
- 30° → rotate 90° clockwise → -60° → flip from left to right → 60° → rotate 90° clockwise → -30°

Use 6 masks to calculate 6 values for each pixel, and find the maximum to compare with the threshold. If the gradient value is larger than the threshold, set the pixel to black (value = 0). If it is smaller, set pixel to white (value = 255)



gradient magnitude: $\max_{n,n=0,\ldots,5} N_n$


```
setMask1 = np.array([[100,100,100,100,100], [100,100,100,100,100], [0,0,0,0,0], [-100,-100,-100,-100,-100], [-
    100,-100,-100,-100,-100]])
setMask2 = np.array([[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]])
```

Result:

(a) Robert's Operator: 25

(b) Prewitt's Edge Detector: 55

(c) Sobel's Edge Detector: 60

(d) Frei and Chen's Gradient: 40

(e) Kirsch's Compass Operator: 135

(f) Robinson's Compass Operator: 43

(g) Nevatia-Babu 5x5 Operator: 12500