CV HW9

資工四 陳光裕 B07902072

Usage:

python3 main.py

Robert's Operator with threshold 12 :

```python
def roberts(img, threshold):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    pad_img = padding(img, 1)
    op = [(0, 0), (0, 1), (1, 0), (1, 1)]
    r1 = [(-1, 0), (0, 1)]
    r2 = [(0, -1), (1, 0)]
    for i in range(1, width+1):
        for j in range(1, height+1):
            grad_r1 = 0
            grad_r2 = 0
            for x, y in op:
                grad_r1 += r1[x][y]*pad_img[i+x][j+y]
                grad_r2 += r2[x][y]*pad_img[i+x][j+y]
            grad = math.sqrt((grad_r1**2) + (grad_r2**2))
            if grad >= threshold:
                new_img[i-1, j-1] = 0
            else:
                new_img[i-1, j-1] = 255
    return new_img
```

Prewitt’s Edge Detector with threshold 24 :

```
def prewitt(img, threshold):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    pad_img = padding(img, 1)
    op = generate_op(3)
    p1 = [(-1, -1, -1), (0, 0, 0), (1, 1, 1)]
    p2 = [(-1, 0, 1), (-1, 0, 1), (-1, 0, 1)]
    for i in range(1, width+1):
        for j in range(1, width+1):
            grad_p1 = 0
            grad_p2 = 0
            for x, y in op:
                grad_p1 += p1[x+1][y+1]*pad_img[i+x][j+y]
                grad_p2 += p2[x+1][y+1]*pad_img[i+x][j+y]
            grad = math.sqrt((grad_p1**2) + (grad_p2**2))
            if grad >= threshold:
                new_img[i-1, j-1] = 0
            else:
                new_img[i-1, j-1] = 255
    return new_img
```

Sobel's Edge Detector with threshold 38 :

```
def sobel(img, threshold):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    pad_img = padding(img, 1)
    op = generate_op(3)
    s1 = [(-1, -2, -1), (0, 0, 0), (1, 2, 1)]
    s2 = [(-1, 0, 1), (-2, 0, 2), (-1, 0, 1)]
    for i in range(1, width+1):
        for j in range(1, width+1):
            grad_s1 = 0
            grad_s2 = 0
            for x, y in op:
                grad_s1 += s1[x+1][y+1]*pad_img[i+x][j+y]
                grad_s2 += s2[x+1][y+1]*pad_img[i+x][j+y]
            grad = math.sqrt((grad_s1**2) + (grad_s2**2))
            if grad >= threshold:
                new_img[i-1, j-1] = 0
            else:
                new_img[i-1, j-1] = 255
    return new_img
```

Frei and Chen’s Gradient Operator with threshold 30 :

```python
def frei_chen(img, threshold):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    pad_img = padding(img, 1)
    op = generate_op(3)
    f1 = [(-1, -math.sqrt(2), -1), (0, 0, 0), (1, math.sqrt(2), 1)]
    f2 = [(-1, 0, 1), (-math.sqrt(2), 0, math.sqrt(2)), (-1, 0, 1)]
    for i in range(1, width+1):
        for j in range(1, width+1):
            grad_f1 = 0
            grad_f2 = 0
            for x, y in op:
                grad_f1 += f1[x+1][y+1]*pad_img[i+x][j+y]
                grad_f2 += f2[x+1][y+1]*pad_img[i+x][j+y]
            grad = math.sqrt((grad_f1**2) + (grad_f2**2))
            if grad >= threshold:
                new_img[i-1, j-1] = 0
            else:
                new_img[i-1, j-1] = 255
    return new_img
```

Kirsch’s Compass Operator with threshold 135 :

```
def kirsch(img, threshold):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    pad_img = padding(img, 1)
    op = generate_op(3)
    k0 = [(-3, -3, 5), (-3, 0, 5), (-3, -3, 5)]
    k1 = [(-3, 5, 5), (-3, 0, 5), (-3, -3, -3)]
    k2 = [(5, 5, 5), (-3, 0, -3), (-3, -3, -3)]
    k3 = [(5, 5, -3), (5, 0, -3), (-3, -3, -3)]
    k4 = [(5, -3, -3), (5, 0, -3), (5, -3, -3)]
    k5 = [(-3, -3, -3), (5, 0, -3), (5, 5, -3)]
    k6 = [(-3, -3, -3), (-3, 0, -3), (5, 5, 5)]
    k7 = [(-3, -3, -3), (-3, 0, 5), (-3, 5, 5)]
    for i in range(1, width+1):
        for j in range(1, width+1):
            grad_k0 = 0
            grad_k1 = 0
            grad_k2 = 0
            grad_k3 = 0
            grad_k4 = 0
            grad_k5 = 0
            grad_k6 = 0
            grad_k7 = 0
            for x, y in op:
                grad_k0 += k0[x+1][y+1]*pad_img[i+x][j+y]
                grad_k1 += k1[x+1][y+1]*pad_img[i+x][j+y]
                grad_k2 += k2[x+1][y+1]*pad_img[i+x][j+y]
                grad_k3 += k3[x+1][y+1]*pad_img[i+x][j+y]
                grad_k4 += k4[x+1][y+1]*pad_img[i+x][j+y]
                grad_k5 += k5[x+1][y+1]*pad_img[i+x][j+y]
                grad_k6 += k6[x+1][y+1]*pad_img[i+x][j+y]
                grad_k7 += k7[x+1][y+1]*pad_img[i+x][j+y]
            grad = max(grad_k0, grad_k1, grad_k2, grad_k3, grad_k4, grad_k5, grad_k6, grad_k7)
            if grad >= threshold:
                new_img[i-1, j-1] = 0
            else:
                new_img[i-1, j-1] = 255
    return new_img
```

Robinson’s Compass Operator with threshold 43 :

```python
def robinson(img, threshold):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    pad_img = padding(img, 1)
    op = generate_op(3)
    r0 = [(-1, 0, 1), (-2, 0, 2), (-1, 0, 1)]
    r1 = [(0, 1, 2), (-1, 0, 1), (-2, -1, 0)]
    r2 = [(1, 2, 1), (0, 0, 0), (-1, -2, -1)]
    r3 = [(2, 1, 0), (1, 0, -1), (0, -1, -2)]
    r4 = [(1, 0, -1), (2, 0, -2), (1, 0, -1)]
    r5 = [(0, -1, -2), (1, 0, -1), (2, 1, 0)]
    r6 = [(-1, -2, -1), (0, 0, 0), (1, 2, 1)]
    r7 = [(-2, -1, 0), (-1, 0, 1), (0, 1, 2)]
    for i in range(1, width+1):
        for j in range(1, width+1):
            grad_r0 = 0
            grad_r1 = 0
            grad_r2 = 0
            grad_r3 = 0
            grad_r4 = 0
            grad_r5 = 0
            grad_r6 = 0
            grad_r7 = 0
            for x, y in op:
                grad_r0 += r0[x+1][y+1]*pad_img[i+x][j+y]
                grad_r1 += r1[x+1][y+1]*pad_img[i+x][j+y]
                grad_r2 += r2[x+1][y+1]*pad_img[i+x][j+y]
                grad_r3 += r3[x+1][y+1]*pad_img[i+x][j+y]
                grad_r4 += r4[x+1][y+1]*pad_img[i+x][j+y]
                grad_r5 += r5[x+1][y+1]*pad_img[i+x][j+y]
                grad_r6 += r6[x+1][y+1]*pad_img[i+x][j+y]
                grad_r7 += r7[x+1][y+1]*pad_img[i+x][j+y]
            grad = max(grad_r0, grad_r1, grad_r2, grad_r3, grad_r4, grad_r5, grad_r6, grad_r7)
            if grad >= threshold:
                new_img[i-1, j-1] = 0
            else:
                new_img[i-1, j-1] = 255
    return new_img
```

Nevatia-Babu 5x5 Operator with threshold 12500 :

```python
def nevatia_babu(img, threshold):
    width, height = img.shape[:2]
    new_img = np.zeros((width, height))
    pad_img = padding(img, 2)
    op = generate_op(5)
    deg0 = [(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)]
    deg30 = [(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)]
    deg60 = [(100, 100, 100, 32, -100), (100, 100, 92, -78, -100), (100, 100, 0, -100, -100), (100, 78, -92, -100, -100), (100, -32, -100, -100, -100)]
    degm90 = [(-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)]
    degm60 = [(-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)]
    degm30 = [(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(2, width+2):
        for j in range(2, width+2):
            grad_deg0 = 0
            grad_deg30 = 0
            grad_deg60 = 0
            grad_degm90 = 0
            grad_degm60 = 0
            grad_degm30 = 0
            for x, y in op:
                grad_deg0 += deg0[x+2][y+2]*pad_img[i+x][j+y]
                grad_deg30 += deg30[x+2][y+2]*pad_img[i+x][j+y]
                grad_deg60 += deg60[x+2][y+2]*pad_img[i+x][j+y]
                grad_degm90 += degm90[x+2][y+2]*pad_img[i+x][j+y]
                grad_degm60 += degm60[x+2][y+2]*pad_img[i+x][j+y]
                grad_degm30 += degm30[x+2][y+2]*pad_img[i+x][j+y]
            grad = max(grad_deg0, grad_deg30, grad_deg60, grad_degm90, grad_degm60, grad_degm30)
            if grad >= threshold:
                new_img[i-2, j-2] = 0
            else:
                new_img[i-2, j-2] = 255
    return new_img
```

產生 NxN 的 filter 陣列 & padding(recursion) :

```python
def generate_op(size):
    a = []
    for i in range(-size//2+1, size//2+1):
        for j in range(-size//2+1, size//2+1):
            a.append((i, j))
    return a

def padding(img, expand):
    width, height = img.shape[:2]
    new_img = np.zeros((width+2, height+2))
    for i in range(width):
        for j in range(height):
            new_img[i+1,j+1] = img[i,j]
    new_img[0,0] = img[0,0]
    new_img[width+1, 0] = img[width-1, 0]
    new_img[width+1, width+1] = img[width-1, width-1]
    new_img[0, width+1] = img[0, width-1]
    for i in range(1, width+2):
        new_img[0, i] = new_img[1, i]
        new_img[width+1, i] = new_img[width, i]
        new_img[i, 0] = new_img[i, 1]
        new_img[i, width+1] = new_img[i, width]
    if expand == 1:
        return new_img
    else:
        return padding(new_img, expand-1)
```