HW 9: Edge Detection

Source Code

All questions are written in Python code, please refer to the file “main.py”.
All images will be stored in the folder “res” (automatically create a new folder).
In accordance with the **FAQ** of course website:

- All parts of the question are written from scratch, except for plotting images

Answer

1. Robert’s Operator

```python
def robert(source, threshold):
    padded = padding(source, 1)
    result = np.zeros(source.shape, dtype=int)
    for i in range(1, padded.shape[0]-1):
        for j in range(1, padded.shape[1]-1):
            box = []
            for x in range(2):
                for y in range(2):
                    xdest = i + x
                    ydest = j + y
                    box.append(padded[xdest][ydest])
            r1 = -int(box[0]) + int(box[3])
            r2 = -int(box[1]) + int(box[2])
            gradient = ((r1**2) + (r2**2)) ** 0.5
            if gradient < threshold:
                result[i-1][j-1] = 255
    return result
```

2. Prewitt's Edge Detector

```python
def prewitt(source, threshold):
    padded = padding(source, 1)
    result = np.zeros(source.shape, dtype=int)
    for i in range(1, padded.shape[0]-1):
        for j in range(1, padded.shape[1]-1):
            box = []
            for x in range(3):
                for y in range(3):
                    xdest = i + x - 1
                    ydest = j + y - 1
                    box.append(padded[xdest][ydest])
            p1 = -int(box[0])-int(box[1])-int(box[2])+int(box[6])+int(box[7])+int(box[8])
            p2 = -int(box[0])-int(box[3])-int(box[6])+int(box[2])+int(box[5])+int(box[8])
            gradient = ((p1**2) + (p2**2)) ** 0.5
            if gradient < threshold:
                result[i-1][j-1] = 255
    return result
```

3. Sobel's Edge Detector

```python
def sobel(source, threshold):
    padded = padding(source, 1)
    result = np.zeros(source.shape, dtype=int)
    for i in range(1, padded.shape[0]-1):
        for j in range(1, padded.shape[1]-1):
            box = []
            for x in range(3):
                for y in range(3):
                    xdest = i + x - 1
                    ydest = j + y - 1
                    box.append(padded[xdest][ydest])
            p1 = -int(box[0])-(int(box[1]) * 2)-int(box[2])+int(box[6])+(int(box[7]) * 2)+int(box[8])
            p2 = -int(box[0])-(int(box[3]) * 2)-int(box[6])+int(box[2])+(int(box[5]) * 2)+int(box[8])
            gradient = ((p1**2) + (p2**2)) ** 0.5
            if gradient < threshold:
                result[i-1][j-1] = 255
    return result
```

4. Frei and Chen's Gradient Operator

```python
def freiandchen(source, threshold):
    padded = padding(source, 1)
    result = np.zeros(source.shape, dtype=int)
    for i in range(1, padded.shape[0]-1):
        for j in range(1, padded.shape[1]-1):
            box = []
            for x in range(3):
                for y in range(3):
                    xdest = i + x - 1
                    ydest = j + y - 1
                    box.append(padded[xdest][ydest])
            p1 = -int(box[0])-(int(box[1]) * (2**0.5))-int(box[2])+int(box[6])+(int(box[7]) * (2**0.5))+int(box[8])
            p2 = -int(box[0])-(int(box[3]) * (2**0.5))-int(box[6])+int(box[2])+(int(box[5]) * (2**0.5))+int(box[8])
            gradient = ((p1**2) + (p2**2)) ** 0.5
            if gradient < threshold:
                result[i-1][j-1] = 255
    return result
```

5. Kirsch's Compass Operator

```
def kirsch(source, threshold):
    padded = padding(source, 1)
    result = np.zeros(source.shape, dtype=int)
    for i in range(1, padded.shape[0]-1):
        for j in range(1, padded.shape[1]-1):
            box = []
            for x in range(3):
                for y in range(3):
                    xdest = i + x - 1
                    ydest = j + y - 1
                    box.append(padded[xdest][ydest])
            k0 = (-3*(int(box[1])+int(box[0])+int(box[3])+int(box[6]+int(box[7])))) + (5*(int(box[2])+int(box[5])+int(box[8])))
            k1 = (-3*(int(box[8])+int(box[0])+int(box[3])+int(box[6]+int(box[7])))) + (5*(int(box[1])+int(box[2])+int(box[5])))
            k2 = (-3*(int(box[8])+int(box[5])+int(box[3])+int(box[6]+int(box[7])))) + (5*(int(box[0])+int(box[1])+int(box[2])))
            k3 = (-3*(int(box[8])+int(box[5])+int(box[2])+int(box[6]+int(box[7])))) + (5*(int(box[0])+int(box[1])+int(box[3])))
            k4 = (-3*(int(box[8])+int(box[5])+int(box[2])+int(box[1]+int(box[7])))) + (5*(int(box[0])+int(box[3])+int(box[6])))
            k5 = (-3*(int(box[8])+int(box[5])+int(box[2])+int(box[1]+int(box[0])))) + (5*(int(box[3])+int(box[6])+int(box[7])))
            k6 = (-3*(int(box[3])+int(box[5])+int(box[2])+int(box[1]+int(box[0])))) + (5*(int(box[6])+int(box[7])+int(box[8])))
            k7 = (-3*(int(box[3])+int(box[6])+int(box[2])+int(box[1]+int(box[0])))) + (5*(int(box[5])+int(box[7])+int(box[8])))
            gradient = max(k0,k1, k2, k3, k4, k5, k6, k7)
            if gradient < threshold:
                result[i-1][j-1] = 255
    return result
```

6. Robinson's Compass Operator

```python
def robinson(source, threshold):
    padded = padding(source, 1)
    result = np.zeros(source.shape, dtype=int)
    for i in range(1, padded.shape[0]-1):
        for j in range(1, padded.shape[1]-1):
            box = []
            for x in range(3):
                for y in range(3):
                    xdest = i + x - 1
                    ydest = j + y - 1
                    box.append(padded[xdest][ydest])
            r0 = -int(box[0])-(int(box[3]) * 2)-int(box[6])+int(box[2])+(int(box[5]) * 2)+int(box[8])
            r1 = -int(box[3])-(int(box[6]) * 2)-int(box[7])+int(box[1])+(int(box[2]) * 2)+int(box[5])
            r2 = -int(box[6])-(int(box[7]) * 2)-int(box[8])+int(box[0])+(int(box[1]) * 2)+int(box[2])
            r3 = -int(box[5])-(int(box[8]) * 2)-int(box[7])+int(box[1])+(int(box[0]) * 2)+int(box[3])
            r4 = -r0
            r5 = -r1
            r6 = -r2
            r7 = -r3
            gradient = max(r0, r1, r2, r3, r4, r5, r6, r7)
            if gradient < threshold:
                result[i-1][j-1] = 255
    return result
```

7. Nevatia-Babu 5x5 Operator

```python
def nevatiababu(source, threshold):
    padded = padding(source, 2)
    result = np.zeros(source.shape, dtype=int)
    n0_mat = [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]
    n1_mat = [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]
    n2_mat = [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]
    n3_mat = [-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]
    n4_mat = [-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]
    n5_mat = [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, padded.shape[0]-2):
        for j in range(2, padded.shape[1]-2):
            box = []
            for x in range(5):
                for y in range(5):
                    xdest = i + x - 2
                    ydest = j + y - 2
                    box.append(padded[xdest][ydest])
            n0 = n1 = n2 = n3 = n4 = n5 = 0
            for x in range(25):
                n0 += (n0_mat[x] * box[x])
                n1 += (n1_mat[x] * box[x])
                n2 += (n2_mat[x] * box[x])
                n3 += (n3_mat[x] * box[x])
                n4 += (n4_mat[x] * box[x])
                n5 += (n5_mat[x] * box[x])
            gradient = max(n0, n1, n2, n3, n4, n5)
            if gradient < threshold:
                result[i-2][j-2] = 255
    return result
```