Computer Vision HW9

系級：資工碩一　　姓名：胡嘉祐　　學號：r07922162

1. Principle Code and Algorithm

Part 1 : Robert's Operator

```
def robert (img , threshold):
    m1 = [[-1,0],[0,1]]
    m2 = [[0,-1],[1,0]]
    img_len = len(img)
    r_img = np.zeros ((img_len,img_len,3),dtype='uint8')
    for i in range (img_len):
        for j in range (img_len):
            r_img[i][j]=(255,255,255)
    for i in range (img_len-1):
        for j in range (img_len-1):
            r1 = m1[0][0]*img[i][j][0]+m1[1][1]*img[i+1][j+1][0]
            r2 = m2[0][1]*img[i][j+1][0]+m2[1][0]*img[i+1][j][0]
            g = r1*r1 + r2*r2
            if ( g > threshold*threshold ):
                r_img[i][j]=(0,0,0)
```

Part 2 : **Prewitt's Edge Detector** and **Sobel's Edge Detector** and **Frei and Chen's Gradient Operator**

These operators are similar, only changes the mask.

```
def prewitt_or_sobel (img ,threshold):

    img_len = len(img)
    r_img = np.zeros ((img_len,img_len,3),dtype='uint8')
    for i in range (img_len):
        for j in range (img_len):
            r_img[i][j]=(255,255,255)
    for i in range (1,img_len-1):
        for j in range (1,img_len-1):
            r1=0
            r2=0
            for ii in range (3):
                for jj in range (3):
                    r1+=img[i+ii-1][j+jj-1][0]*m1[ii][jj]
                    r2+=img[i+ii-1][j+jj-1][0]*m2[ii][jj]
            g = r1*r1 + r2*r2
            if ( g > threshold*threshold ):
                r_img[i][j]=(0,0,0)

    return r_img
```

Mask of Prewitt's :

```
m1 = [[-1,-1,-1],[0,0,0],[1,1,1]]
m2 = [[-1,0,1],[-1,0,1],[-1,0,1]]
```

Mask of Sobel's :

```
m1 = [[-1,-2,-1],[0,0,0],[1,2,1]]
m2 = [[-1,0,1],[-2,0,2],[-1,0,1]]
```

Mask of Frei and Chen:

```
root2 = np.sqrt(2)
m1 = [[-1,-root2,-1],[0,0,0],[1,root2,1]]
m2 = [[-1,0,1],[-root2,0,root2],[-1,0,1]]
```

Part 3 : Kirsch's Compass Operator and Robinson's Compass Operator

Kirsch's and Robinson's Compass Operator are similar, only change the mask.

```
def kirsch (img , threshold):

    img_len = len(img)
    r_img = np.zeros ((img_len,img_len,3),dtype='uint8')
    for i in range (img_len):
        for j in range (img_len):
            r_img[i][j]=(255,255,255)
    for i in range (1,img_len-1):
        for j in range (1,img_len-1):

            g = get_manitude_3(img,i,j,k[0])

            for k_index in range (1,8):

                g = max(g,get_manitude_3 (img,i,j,k[k_index]))
            if ( g > threshold ):
                r_img[i][j]=(0,0,0)
    return r_img
```

Mask of Kirsch's :

```
k= [[[-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]]]
```

Mask of Robinson's :

```
k= [[[-1, -2, -1], [0, 0, 0], [1, 2, 1]],
    [[0, -1, -2], [1, 0, -1], [2, 1, 0]],
    [[1, 0, -1], [2, 0, -2], [1, 0, -1]],
    [[2, 1, 0], [1, 0, -1], [0, -1, -2]],
    [[1, 2, 1], [0, 0, 0], [-1, -2, -1]],
    [[0, 1, 2], [-1, 0, 1], [-2, -1, 0]],
    [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]],
    [[-2, -1, 0], [-1, 0, 1], [0, 1, 2]]]
```

```
def get_manitude_3 (img,r,c,k ):
    g = 0
    for ii in range (3):
        for jj in range (3):
            g+=img[r+ii-1][c+jj-1][0]*k[ii][jj]
    return g
```

Part 4 : Nevatia–Babu 5x5 Operator

```
def nevatia_babu (img , threshold):

    k  = [[[100, 100, 0, -100, -100], [100, 100, 0, -100, -100], [
          [[100, 100, 100, 32, -100], [100, 100, 92, -78, -100], [
          [[100, 100, 100, 100, 100], [100, 100, 100, 78, -32], [1
          [[-100, -100, -100, -100, -100], [-100, -100, -100, -100
          [[-100, -100, -100, -100, -100], [32, -78, -100, -100, -
          [[100, -32, -100, -100, -100], [100, 78, -92, -100, -100
    img_len = len(img)
    r_img = np.zeros ((img_len,img_len,3),dtype='uint8')
    for i in range (img_len):
        for j in range (img_len):
            r_img[i][j]=(255,255,255)
    for i in range (2,img_len-2):
        for j in range (2,img_len-2):

            g = get_manitude_5(img,i,j,k[0])

            for k_index in range (1,6):
                g = max(g,get_manitude_5 (img,i,j,k[k_index]))
            if ( g > threshold ):
                r_img[i][j]=(0,0,0)
    return r_img
```

```
def get_manitude_5 (img,r,c,k ):
    g = 0
    for ii in range (5):
        for jj in range (5):
            g+=img[r+ii-2][c+jj-2][0]*k[ii][jj]
    return g
```

Part 5 : Main

The threshold use :

Robert's Operator: 12

Prewitt's Edge Detector: 24

Sobel's Edge Detector: 38

Frei and Chen's Gradient Operator: 30

Kirsch's Compass Operator: 135

Robinson's Compass Operator: 43

Nevatia–Babu 5x5 Operator: 12500

```
def main ():

    ori = cv2.imread(sys.argv[1])
    cv2.imwrite("robert12.png",robert (ori , 12))
    cv2.imwrite("prewitt24.png",prewitt (ori , 24))
    cv2.imwrite("sobel38.png",sobel (ori , 38))
    cv2.imwrite("Frei_Chen30.png",Frei_Chen (ori , 30))
    cv2.imwrite("kirsch135.png",kirsch (ori , 135))
    cv2.imwrite("robinson43.png",robinson (ori , 43))
    cv2.imwrite("nevatia_babu12500.png",nevatia_babu (ori , 12500))
```

2. Result

Robert's Operator: 12

Prewitt's Edge Detector: 24

Sobel's Edge Detector: 38

Frei and Chen's Gradient Operator: 30

Kirsch's Compass Operator: 135

Robinson's Compass Operator: 43

Nevatia–Babu 5x5 Operator: 12500