資工所碩一 R07922164 林群崴

本次作業採用 **C++配合 OpenCV** 完成

```
int convolve(Mat a, Mat b) {

     int result = 0;

     for (int i = 0; i < a.rows; i++) {
          for (int j = 0; j < a.cols; j++) {
               result += int(a.at<uchar>(i, j)) * int(b.at<char>(b.rows - i - 1,
b.cols - j - 1));
          }
     }
     return result;
}
```

1. Robert's Operator

Figure 7.21 Masks used for the Roberts operators.

```
Mat Roberts(Mat img) {
     Mat mask1(2, 2, CV_8S, Scalar(0));
     Mat mask2(2, 2, CV_8S, Scalar(0));
     mask1.at<char>(0, 0) = 1;
     mask1.at<char>(1, 1) = -1;
     mask2.at<char>(0, 1) = 1;
     mask2.at<char>(1, 0) = -1;
     Mat Gx(img.rows - 1, img.cols - 1, CV_32S, Scalar(0));
     Mat Gy(img.rows - 1, img.cols - 1, CV_32S, Scalar(0));
     for (int i = 0; i < img.rows - 1; i++) {
          for (int j = 0; j < img.rows - 1; j++) {
               Rect r(j, i, 2, 2);
               Gx.at<int>(i, j) = convolve(img(r), mask1);
               Gy.at<int>(i, j) = convolve(img(r), mask2);
          }
     }
     Mat G1(img.rows - 1, img.cols - 1, CV_32S, Scalar(0));
```

```
    Mat G2(img.rows - 1, img.cols - 1, CV_32S, Scalar(0));
    pow(Gx, 2, G1);
    pow(Gy, 2, G2);
    Mat G(img.rows - 1, img.cols - 1, CV_32S, Scalar(0));
    G = G1 + G2;
    for (int i = 0; i < img.rows - 1; i++) {
        for (int j = 0; j < img.rows - 1; j++) {
            G.at<int>(i, j) = sqrt(G.at<int>(i, j));
            if (G.at<int>(i, j) > 12) G.at<int>(i, j) = 0;
            else G.at<int>(i, j) = 255;
        }
    }
    return G;
}
```

採用 12 作為 threshold value

2. Prewitt's Edge Detector

-1	-1	-1
1	1	1

p_1

-1		1
-1		1
-1		1

p_2

Figure 7.22 Prewitt edge detector masks.

```
Mat Prewitt(Mat img) {
    Mat mask1(3, 3, CV_8S, Scalar(0));
    Mat mask2(3, 3, CV_8S, Scalar(0));
    mask1.at<char>(0, 0) = -1;
    mask1.at<char>(1, 0) = -1;
```

```
    mask1.at<char>(2, 0) = -1;
    mask1.at<char>(0, 2) = 1;
    mask1.at<char>(1, 2) = 1;
    mask1.at<char>(2, 2) = 1;
    mask2.at<char>(0, 0) = -1;
    mask2.at<char>(0, 1) = -1;
    mask2.at<char>(0, 2) = -1;
    mask2.at<char>(2, 0) = 1;
    mask2.at<char>(2, 1) = 1;
    mask2.at<char>(2, 2) = 1;
    Mat Gx(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    Mat Gy(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    for (int i = 0; i < img.rows - 2; i++) {
        for (int j = 0; j < img.rows - 2; j++) {
            Rect r(j, i, 3, 3);
            Gx.at<int>(i, j) = convolve(img(r), mask1);
            Gy.at<int>(i, j) = convolve(img(r), mask2);
        }
    }
    Mat G1(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    Mat G2(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    pow(Gx, 2, G1);
    pow(Gy, 2, G2);
    Mat G(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    G = G1 + G2;
    for (int i = 0; i < img.rows - 2; i++) {
        for (int j = 0; j < img.rows - 2; j++) {
            G.at<int>(i, j) = sqrt(G.at<int>(i, j));
            if (G.at<int>(i, j) > 24) G.at<int>(i, j) = 0;
            else G.at<int>(i, j) = 255;
        }
    }
    return G;
}
```

採用 24 作為 threshold value

3. Sobel's Edge Detector

-1	-2	-1
1	2	1

s_1

-1		1
-2		2
-1		1

s_2

Figure 7.23 Sobel edge detector masks.

```
Mat Sobel(Mat img) {
    Mat mask1(3, 3, CV_8S, Scalar(0));
    Mat mask2(3, 3, CV_8S, Scalar(0));

    mask1.at<char>(0, 0) = -1;
    mask1.at<char>(1, 0) = -2;
    mask1.at<char>(2, 0) = -1;
    mask1.at<char>(0, 2) = 1;
    mask1.at<char>(1, 2) = 2;
    mask1.at<char>(2, 2) = 1;

    mask2.at<char>(0, 0) = -1;
    mask2.at<char>(0, 1) = -2;
    mask2.at<char>(0, 2) = -1;
    mask2.at<char>(2, 0) = 1;
    mask2.at<char>(2, 1) = 2;
    mask2.at<char>(2, 2) = 1;
```

```
    Mat Gx(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    Mat Gy(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));

    for (int i = 0; i < img.rows - 2; i++) {
        for (int j = 0; j < img.rows - 2; j++) {
            Rect r(j, i, 3, 3);
            Gx.at<int>(i, j) = convolve(img(r), mask1);
            Gy.at<int>(i, j) = convolve(img(r), mask2);
        }
    }
    Mat G1(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    Mat G2(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    pow(Gx, 2, G1);
    pow(Gy, 2, G2);
    Mat G(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    G = G1 + G2;
    for (int i = 0; i < img.rows - 2; i++) {
        for (int j = 0; j < img.rows - 2; j++) {
            G.at<int>(i, j) = sqrt(G.at<int>(i, j));
            if (G.at<int>(i, j) > 38) G.at<int>(i, j) = 0;
            else G.at<int>(i, j) = 255;
        }
    }
    return G;
}
```

採用 38 作為 threshold value

4. Frei and Chen's Gradient Operator

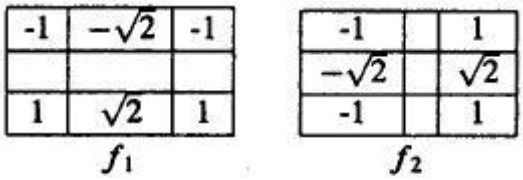

Figure 7.24 Frei and Chen gradient masks.

```
Mat Frei(Mat img) {
     Mat mask1(3, 3, CV_64F, Scalar(0));
     Mat mask2(3, 3, CV_64F, Scalar(0));
     mask1.at<double>(0, 0) = -1;
     mask1.at<double>(0, 1) = -sqrt(2);
     mask1.at<double>(0, 2) = -1;
     mask1.at<double>(2, 0) = 1;
     mask1.at<double>(2, 1) = sqrt(2);
     mask1.at<double>(2, 2) = 1;
     mask2.at<double>(0, 0) = -1;
     mask2.at<double>(0, 1) = -0;
     mask2.at<double>(0, 2) = 1;
     mask2.at<double>(1, 0) = -sqrt(2);
     mask2.at<double>(1, 2) = sqrt(2);
     mask2.at<double>(2, 0) = -1;
     mask2.at<double>(2, 1) = 0;
     mask2.at<double>(2, 2) = 1;
     Mat Gx(img.rows - 2, img.cols - 2, CV_64F, Scalar(0));
     Mat Gy(img.rows - 2, img.cols - 2, CV_64F, Scalar(0));

     for (int i = 0; i < img.rows - 2; i++) {
          for (int j = 0; j < img.rows - 2; j++) {
               Rect r(j, i, 3, 3);
               Gx.at<double>(i, j) = convolve2(img(r), mask1);
               Gy.at<double>(i, j) = convolve2(img(r), mask2);
          }
     }
     Mat G1(img.rows - 2, img.cols - 2, CV_64F, Scalar(0));
     Mat G2(img.rows - 2, img.cols - 2, CV_64F, Scalar(0));
     pow(Gx, 2, G1);
```

```
        pow(Gy, 2, G2);
        Mat G(img.rows - 2, img.cols - 2, CV_64F, Scalar(0));
        G = G1 + G2;
        for (int i = 0; i < img.rows - 2; i++) {
            for (int j = 0; j < img.rows - 2; j++) {
                G.at<double>(i, j) = sqrt(G.at<double>(i, j));
                if (G.at<double>(i, j) > 30) G.at<double>(i, j) = 0;
                else G.at<double>(i, j) = 255;
            }
        }
        return G;
}
```

採用 30 作為 threshold value

5. Kirsch's Compass Operator

$$M_1 = \begin{bmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix},$$

$$M_3 = \begin{bmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{bmatrix}, \quad M_4 = \begin{bmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{bmatrix},$$

$$M_5 = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad M_6 = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

$$M_7 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}, \quad M_8 = \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}, \quad M_9 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
Mat Kirsch(Mat img) {
    Mat k0(3, 3, CV_8S, Scalar(0));
    Mat k1(3, 3, CV_8S, Scalar(0));
    Mat k2(3, 3, CV_8S, Scalar(0));
    Mat k3(3, 3, CV_8S, Scalar(0));
    Mat k4(3, 3, CV_8S, Scalar(0));
    Mat k5(3, 3, CV_8S, Scalar(0));
    Mat k6(3, 3, CV_8S, Scalar(0));
    Mat k7(3, 3, CV_8S, Scalar(0));
    k0.at<char>(0, 0) = -3; k0.at<char>(0, 1) = -3; k0.at<char>(0, 2) = 5;
    k0.at<char>(1, 0) = -3; k0.at<char>(1, 1) = 0; k0.at<char>(1, 2) = 5;
    k0.at<char>(2, 0) = -3; k0.at<char>(2, 1) = -3; k0.at<char>(2, 2) = 5;
    k1.at<char>(0, 0) = -3; k1.at<char>(0, 1) = 5; k1.at<char>(0, 2) = 5;
    k1.at<char>(1, 0) = -3; k1.at<char>(1, 1) = 0; k1.at<char>(1, 2) = 5;
    k1.at<char>(2, 0) = -3; k1.at<char>(2, 1) = -3; k1.at<char>(2, 2) = -3;
    k2.at<char>(0, 0) = 5; k2.at<char>(0, 1) = 5; k2.at<char>(0, 2) = 5;
    k2.at<char>(1, 0) = -3; k2.at<char>(1, 1) = 0; k2.at<char>(1, 2) = -3;
    k2.at<char>(2, 0) = -3; k2.at<char>(2, 1) = -3; k2.at<char>(2, 2) = -3;
    k3.at<char>(0, 0) = 5; k3.at<char>(0, 1) = 5; k3.at<char>(0, 2) = -3;
    k3.at<char>(1, 0) = 5; k3.at<char>(1, 1) = 0; k3.at<char>(1, 2) = -3;
    k3.at<char>(2, 0) = -3; k3.at<char>(2, 1) = -3; k3.at<char>(2, 2) = -3;
    k4.at<char>(0, 0) = 5; k4.at<char>(0, 1) = -3; k4.at<char>(0, 2) = -3;
    k4.at<char>(1, 0) = 5; k4.at<char>(1, 1) = 0; k4.at<char>(1, 2) = -3;
    k4.at<char>(2, 0) = 5; k4.at<char>(2, 1) = -3; k4.at<char>(2, 2) = -3;
    k5.at<char>(0, 0) = -3; k5.at<char>(0, 1) = -3; k5.at<char>(0, 2) = -3;
    k5.at<char>(1, 0) = 5; k5.at<char>(1, 1) = 0; k5.at<char>(1, 2) = -3;
    k5.at<char>(2, 0) = 5; k5.at<char>(2, 1) = 5; k5.at<char>(2, 2) = -3;
    k6.at<char>(0, 0) = -3; k6.at<char>(0, 1) = -3; k6.at<char>(0, 2) = -3;
    k6.at<char>(1, 0) = -3; k6.at<char>(1, 1) = 0; k6.at<char>(1, 2) = -3;
    k6.at<char>(2, 0) = 5; k6.at<char>(2, 1) = 5; k6.at<char>(2, 2) = 5;
    k7.at<char>(0, 0) = -3; k7.at<char>(0, 1) = -3; k7.at<char>(0, 2) = -3;
    k7.at<char>(1, 0) = -3; k7.at<char>(1, 1) = 0; k7.at<char>(1, 2) = 5;
    k7.at<char>(2, 0) = -3; k7.at<char>(2, 1) = 5; k7.at<char>(2, 2) = 5;
    Mat G(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    for (int i = 0; i < img.rows - 2; i++) {
        for (int j = 0; j < img.rows - 2; j++) {
            Rect r(j, i, 3, 3);
            int g[7];
```

```
                g[0] = convolve(img(r), k0);
                g[1] = convolve(img(r), k1);
                g[2] = convolve(img(r), k2);
                g[3] = convolve(img(r), k3);
                g[4] = convolve(img(r), k4);
                g[5] = convolve(img(r), k5);
                g[6] = convolve(img(r), k6);
                int _max = convolve(img(r), k7);
                for (int k = 1; k < 7; k++) {
                     _max = max(_max,g[k]);
                }
                G.at<int>(i, j) = (_max > 135) ? 0 : 255;
           }
     }
     return G;
}
```

採用 135 作為 threshold value

6. Robinson's Compass Operator

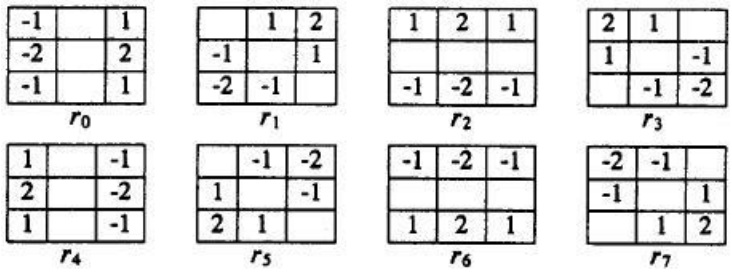

Figure 7.26 Robinson compass masks.

```
Mat Robinson(Mat img) {
    Mat k0(3, 3, CV_8S, Scalar(0));
    Mat k1(3, 3, CV_8S, Scalar(0));
    Mat k2(3, 3, CV_8S, Scalar(0));
    Mat k3(3, 3, CV_8S, Scalar(0));
    Mat k4(3, 3, CV_8S, Scalar(0));
    Mat k5(3, 3, CV_8S, Scalar(0));
    Mat k6(3, 3, CV_8S, Scalar(0));
    Mat k7(3, 3, CV_8S, Scalar(0));
    k0.at<char>(0, 0) = -1; k0.at<char>(0, 1) = 0; k0.at<char>(0, 2) = 1;
    k0.at<char>(1, 0) = -2; k0.at<char>(1, 1) = 0; k0.at<char>(1, 2) = 2;
    k0.at<char>(2, 0) = -1; k0.at<char>(2, 1) = 0; k0.at<char>(2, 2) = 1;
    k1.at<char>(0, 0) = 0; k1.at<char>(0, 1) = 1; k1.at<char>(0, 2) = 2;
    k1.at<char>(1, 0) = -1; k1.at<char>(1, 1) = 0; k1.at<char>(1, 2) = 1;
    k1.at<char>(2, 0) = -2; k1.at<char>(2, 1) = -1; k1.at<char>(2, 2) = 0;
    k2.at<char>(0, 0) = 1; k2.at<char>(0, 1) = 2; k2.at<char>(0, 2) = 1;
    k2.at<char>(1, 0) = 0; k2.at<char>(1, 1) = 0; k2.at<char>(1, 2) = 0;
    k2.at<char>(2, 0) = -1; k2.at<char>(2, 1) = -2; k2.at<char>(2, 2) = -1;
    k3.at<char>(0, 0) = 2; k3.at<char>(0, 1) = 1; k3.at<char>(0, 2) = 0;
    k3.at<char>(1, 0) = 1; k3.at<char>(1, 1) = 0; k3.at<char>(1, 2) = -1;
    k3.at<char>(2, 0) = 0; k3.at<char>(2, 1) = -1; k3.at<char>(2, 2) = -2;
    k4.at<char>(0, 0) = 1; k4.at<char>(0, 1) = 0; k4.at<char>(0, 2) = -1;
    k4.at<char>(1, 0) = 2; k4.at<char>(1, 1) = 0; k4.at<char>(1, 2) = -2;
    k4.at<char>(2, 0) = 1; k4.at<char>(2, 1) = 0; k4.at<char>(2, 2) = -1;
    k5.at<char>(0, 0) = 0; k5.at<char>(0, 1) = -1; k5.at<char>(0, 2) = -2;
    k5.at<char>(1, 0) = 1; k5.at<char>(1, 1) = 0; k5.at<char>(1, 2) = -1;
    k5.at<char>(2, 0) = 2; k5.at<char>(2, 1) = 1; k5.at<char>(2, 2) = 0;
    k6.at<char>(0, 0) = -1; k6.at<char>(0, 1) = -2; k6.at<char>(0, 2) = -1;
    k6.at<char>(1, 0) = 0; k6.at<char>(1, 1) = 0; k6.at<char>(1, 2) = 0;
```

```
    k6.at<char>(2, 0) = 1; k6.at<char>(2, 1) = 2; k6.at<char>(2, 2) = 1;
    k7.at<char>(0, 0) = -2; k7.at<char>(0, 1) = -1; k7.at<char>(0, 2) = 0;
    k7.at<char>(1, 0) = -1; k7.at<char>(1, 1) = 0; k7.at<char>(1, 2) = 1;
    k7.at<char>(2, 0) = 0; k7.at<char>(2, 1) = 1; k7.at<char>(2, 2) = 2;
    Mat G(img.rows - 2, img.cols - 2, CV_32S, Scalar(0));
    for (int i = 0; i < img.rows - 2; i++) {
        for (int j = 0; j < img.rows - 2; j++) {
            Rect r(j, i, 3, 3);
            int g[7];
            g[0] = convolve(img(r), k0);
            g[1] = convolve(img(r), k1);
            g[2] = convolve(img(r), k2);
            g[3] = convolve(img(r), k3);
            g[4] = convolve(img(r), k4);
            g[5] = convolve(img(r), k5);
            g[6] = convolve(img(r), k6);
            int _max = convolve(img(r), k7);
            for (int k = 1; k < 7; k++) {
                _max = max(_max, g[k]);
            }
            G.at<int>(i, j) = (_max > 43) ? 0 : 255;
        }
    }
    return G;
}
```

採用 43 作為 threshold value

7. Nevatia-Babu 5x5 Operator

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

0°

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

30°

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

60°

-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

-90°

-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

-60°

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

-30°

Figure 7.27 Nevatia-Babu 5 × 5 compass template masks.

程式碼置於 source code 資料夾中 CV-HW9.cpp

採用 12500 作為 threshold value