電腦視覺作業九

Computer Vision HW9
R04525092 工科碩二 鄭力文
https://github.com/Mike-Zheng/NTU-Computer-Vision-I-

使用語言 C++ with openCV

Write programs to generate the following gradient magnitude images and choose proper thresholds to get the binary edge images:
Roberts operator
Prewitt edge detector
Sobel edge detector
Frei and Chen gradient operator
Kirsch compass operator
Robinson compass operator
Nevatia-Babu 5X5 operator

Functions

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <algorithm>
#include <math.h>
using namespace cv;
using namespace std;

Mat image = imread("lena.bmp");
cv::Size s = image.size();
int rows = s.height;
int cols = s.width;
void runRobertOperator(Mat ,Mat&  ,int );
void runPrewittEdgeDetector(Mat ,Mat&  ,int );
void runFreiandChenGradientOperator(Mat ,Mat&  ,int );
void runKirschCompassOperator(Mat ,Mat&  ,int );
void runRobinsonCompassOperator(Mat ,Mat&  ,int );
void runNevatiaBabu55Operator(Mat ,Mat&  ,int );
```

1. Roberts operator

Figure 7.21 Masks used for the Roberts operators.

```
void runRobertOperator(Mat image,Mat&  imageHandling,int threshold){
    Mat tempImage=image.clone();
    int rows = tempImage.rows;
    int cols = tempImage.cols;
    int r1[2][2]={-1,0,0,1};
    int r2[2][2]={0,-1,1,0};
    int r11,r12,r21,r22,g;

    for (int y=0;y<rows-1;y++){
        for(int x=0;x<cols-1;x++){
            r11=image.at<Vec3b>(x,y)[0];
            r12=image.at<Vec3b>(x+1,y+1)[0];
            r21=image.at<Vec3b>(x+1,y)[0];
            r22=image.at<Vec3b>(x,y+1)[0];
            g=(r1[0][0]*r11+r1[1][1]*r12)*(r1[0][0]*r11+r1[1][1]*r12)+(r2[0][1]*r21+r2[1][0]*r22
                )*(r2[0][1]*r21+r2[1][0]*r22);
            if(g>threshold*threshold){
                tempImage.at<Vec3b>(x,y)[0]=0;
                tempImage.at<Vec3b>(x,y)[1]=0;
                tempImage.at<Vec3b>(x,y)[2]=0;
            }
            else{
                tempImage.at<Vec3b>(x,y)[0]=255;
                tempImage.at<Vec3b>(x,y)[1]=255;
                tempImage.at<Vec3b>(x,y)[2]=255;
            }

        }
    }
    imageHandling=tempImage;

}
```

2. Prewitt 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.

```
void runPrewittEdgeDetector(Mat image,Mat&  imageHandling,int threshold){
    Mat tempImage=image.clone();
    int rows = tempImage.rows;
    int cols = tempImage.cols;
    int mask1[3][3]={-1,-1,-1,0,0,0,1,1,1};
    int mask2[3][3]={-1,0,1,-1,0,1,-1,0,1};
    int p1,p2,p11,p12,p13,p14,p15,p16,p21,p22,p23,p24,p25,p26,g;


    for (int y=0;y<rows-1;y++){
        for(int x=0;x<cols-1;x++){
            p11=image.at<Vec3b>(x,y)[0];
            p12=image.at<Vec3b>(x+1,y)[0];
            p13=image.at<Vec3b>(x+2,y)[0];
            p14=image.at<Vec3b>(x,y+2)[0];
            p15=image.at<Vec3b>(x+1,y+2)[0];
            p16=image.at<Vec3b>(x+2,y+2)[0];
            p21=image.at<Vec3b>(x,y)[0];
            p22=image.at<Vec3b>(x+2,y)[0];
            p23=image.at<Vec3b>(x,y+1)[0];
            p24=image.at<Vec3b>(x+2,y+1)[0];
            p25=image.at<Vec3b>(x,y+2)[0];
            p26=image.at<Vec3b>(x+2,y+2)[0];
            p1=p11*mask1[0][0]+p12*mask1[0][1]+p13*mask1[0][2]+p14*mask1[2][0]+p15*mask1[2][1]+
                p16*mask1[2][2];
            p2=p21*mask2[0][0]+p22*mask2[0][2]+p23*mask2[1][0]+p24*mask2[1][2]+p25*mask2[2][0]+
                p26*mask2[2][2];

            g=p1*p1+p2*p2;


            if(g>threshold*threshold){
                tempImage.at<Vec3b>(x,y)[0]=0;
                tempImage.at<Vec3b>(x,y)[1]=0;
                tempImage.at<Vec3b>(x,y)[2]=0;
            }
            else{
                tempImage.at<Vec3b>(x,y)[0]=255;
                tempImage.at<Vec3b>(x,y)[1]=255;
                tempImage.at<Vec3b>(x,y)[2]=255;
            }

        }
    }
    imageHandling=tempImage;

}
```

3. Sobel edge detector

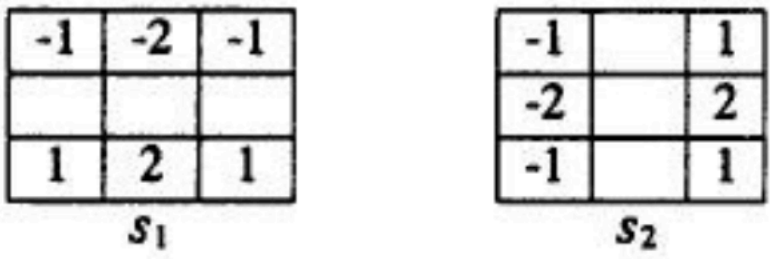

Figure 7.23 Sobel edge detector masks.

```
void runSobelEdgeDetector(Mat image,Mat&  imageHandling,int threshold){
    Mat tempImage=image.clone();
    int rows = tempImage.rows;
    int cols = tempImage.cols;
    int mask1[3][3]={-1,-2,-1,0,0,0,1,2,1};
    int mask2[3][3]={-1,0,1,-2,0,2,-1,0,1};

    int p1,p2,p11,p12,p13,p14,p15,p16,p21,p22,p23,p24,p25,p26,g;


    for (int y=0;y<rows-1;y++){
        for(int x=0;x<cols-1;x++){
            p11=image.at<Vec3b>(x,y)[0];
            p12=image.at<Vec3b>(x+1,y)[0];
            p13=image.at<Vec3b>(x+2,y)[0];
            p14=image.at<Vec3b>(x,y+2)[0];
            p15=image.at<Vec3b>(x+1,y+2)[0];
            p16=image.at<Vec3b>(x+2,y+2)[0];
            p21=image.at<Vec3b>(x,y)[0];
            p22=image.at<Vec3b>(x+2,y)[0];
            p23=image.at<Vec3b>(x,y+1)[0];
            p24=image.at<Vec3b>(x+2,y+1)[0];
            p25=image.at<Vec3b>(x,y+2)[0];
            p26=image.at<Vec3b>(x+2,y+2)[0];
            p1=p11*mask1[0][0]+p12*mask1[0][1]+p13*mask1[0][2]+p14*mask1[2][0]+p15*mask1[2][1]+
                p16*mask1[2][2];
            p2=p21*mask2[0][0]+p22*mask2[0][2]+p23*mask2[1][0]+p24*mask2[1][2]+p25*mask2[2][0]+
                p26*mask2[2][2];

            g=p1*p1+p2*p2;


            if(g>threshold*threshold){
                tempImage.at<Vec3b>(x,y)[0]=0;
                tempImage.at<Vec3b>(x,y)[1]=0;
                tempImage.at<Vec3b>(x,y)[2]=0;
            }
            else{
                tempImage.at<Vec3b>(x,y)[0]=255;
                tempImage.at<Vec3b>(x,y)[1]=255;
                tempImage.at<Vec3b>(x,y)[2]=255;
            }

        }
    }
    imageHandling=tempImage;

}
```

4. Frei and Chen gradient operator

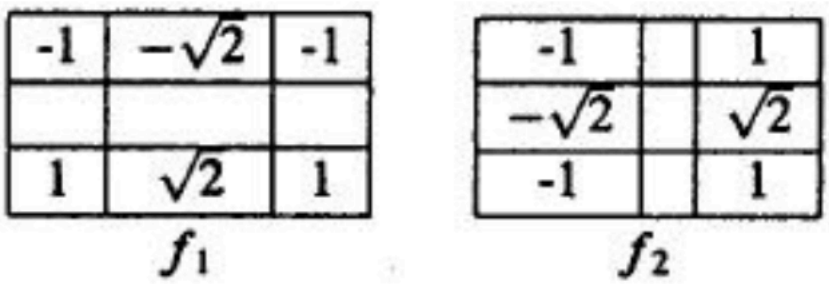

Figure 7.24 Frei and Chen gradient masks.

```
void runFreiandChenGradientOperator(Mat image,Mat&  imageHandling,int threshold){
    Mat tempImage=image.clone();
    int rows = tempImage.rows;
    int cols = tempImage.cols;
    double mask1[3][3]={-1,-sqrt(2),-1,0,0,0,1,sqrt(2),1};
    double mask2[3][3]={-1,0,1,-sqrt(2),0,sqrt(2),-1,0,1};

    double p1,p2,p11,p12,p13,p14,p15,p16,p21,p22,p23,p24,p25,p26,g;


    for (int y=0;y<rows-1;y++){
        for(int x=0;x<cols-1;x++){
            p11=image.at<Vec3b>(x,y)[0];
            p12=image.at<Vec3b>(x+1,y)[0];
            p13=image.at<Vec3b>(x+2,y)[0];
            p14=image.at<Vec3b>(x,y+2)[0];
            p15=image.at<Vec3b>(x+1,y+2)[0];
            p16=image.at<Vec3b>(x+2,y+2)[0];
            p21=image.at<Vec3b>(x,y)[0];
            p22=image.at<Vec3b>(x+2,y)[0];
            p23=image.at<Vec3b>(x,y+1)[0];
            p24=image.at<Vec3b>(x+2,y+1)[0];
            p25=image.at<Vec3b>(x,y+2)[0];
            p26=image.at<Vec3b>(x+2,y+2)[0];
            p1=p11*mask1[0][0]+p12*mask1[0][1]+p13*mask1[0][2]+p14*mask1[2][0]+p15*mask1[2][1]+
                p16*mask1[2][2];
            p2=p21*mask2[0][0]+p22*mask2[0][2]+p23*mask2[1][0]+p24*mask2[1][2]+p25*mask2[2][0]+
                p26*mask2[2][2];

            g=p1*p1+p2*p2;


            if(g>threshold*threshold){
                tempImage.at<Vec3b>(x,y)[0]=0;
                tempImage.at<Vec3b>(x,y)[1]=0;
                tempImage.at<Vec3b>(x,y)[2]=0;
            }
            else{
                tempImage.at<Vec3b>(x,y)[0]=255;
                tempImage.at<Vec3b>(x,y)[1]=255;
                tempImage.at<Vec3b>(x,y)[2]=255;
            }

        }
    }
    imageHandling=tempImage;

}
```

5. Kirsch compass operator

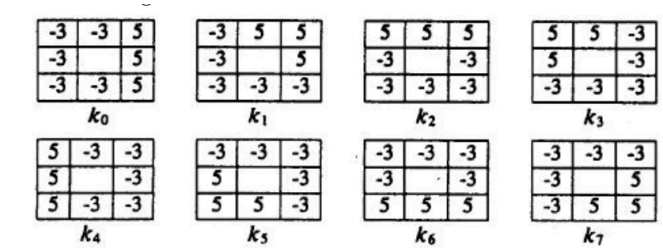

Figure 7.25 Kirsch compass masks.

```
void runKirschCompassOperator(Mat image,Mat&  imageHandling,int threshold){
    Mat tempImage=image.clone();
    int rows = tempImage.rows;
    int cols = tempImage.cols;
    int k0[3][3] = {-3,-3,5,-3,0,5,-3,-3,5};
    int k1[3][3] = {-3,5,5,-3,0,5,-3,-3,-3};
    int k2[3][3] = {5,5,5,-3,0,-3,-3,-3,-3};
    int k3[3][3] = {5,5,-3,5,0,-3,-3,-3,-3};
    int k4[3][3] = {5,-3,-3,5,0,-3,5,-3,-3};
    int k5[3][3] = {-3,-3,-3,5,0,-3,5,5,-3};
    int k6[3][3] = {-3,-3,-3,-3,0,-3,5,5,5};
    int k7[3][3] = {-3,-3,-3,-3,0,5,-3,5,5};
    int g ,temp_g;
    int y0,y1,y2,y3,y4,y5,y6,y7,y8;
    for (int y=0;y<rows;y++){
        for(int x=0;x<cols;x++){
            y0=image.at<Vec3b>(x,y)[0];
            if(y<512)y1=image.at<Vec3b>(x+1,y)[0];else y1=0;
            if(y!=0) y2=image.at<Vec3b>(x,y-1)[0]; else y2=0;
            if(x!=0) y3=image.at<Vec3b>(x-1,y)[0]; else y3=0;
            if(x<512) y4=image.at<Vec3b>(x,y+1)[0]; else y4=0;
            if(x<512&&y<512) y5=image.at<Vec3b>(x+1,y+1)[0]; else y5=0;
            if(y!=0&&x<512) y6=image.at<Vec3b>(x+1,y-1)[0]; else y6=0;
            if(y!=0&&x!=0) y7=image.at<Vec3b>(x-1,y-1)[0]; else y7=0;
            if(x!=0&&y<512) y8=image.at<Vec3b>(x-1,y+1)[0]; else y8=0;
            g = 0;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k0);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k1);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k2);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k3);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k4);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k5);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k6);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,k7);
            if ( temp_g>g ) g=temp_g;

            if(g>threshold){
                tempImage.at<Vec3b>(x,y)[0]=0;
                tempImage.at<Vec3b>(x,y)[1]=0;
                tempImage.at<Vec3b>(x,y)[2]=0;
            }
            else{
                tempImage.at<Vec3b>(x,y)[0]=255;
                tempImage.at<Vec3b>(x,y)[1]=255;
                tempImage.at<Vec3b>(x,y)[2]=255;
            }

        }
    }
    imageHandling=tempImage;
```

6. Robinson compass operator

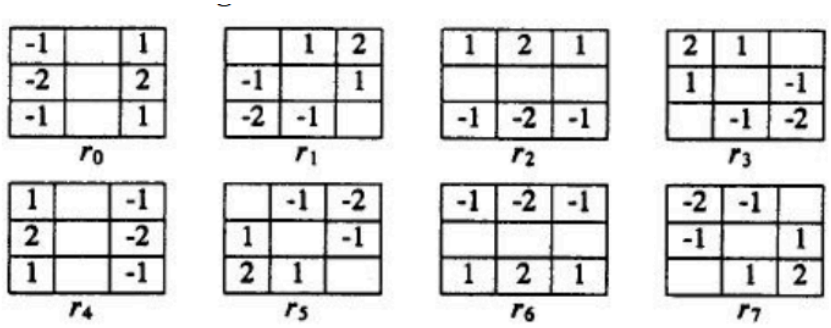

Figure 7.26 Robinson compass masks.

```
void runRobinsonCompassOperator(Mat image,Mat&  imageHandling,int threshold){
    Mat tempImage=image.clone();
    int rows = tempImage.rows;
    int cols = tempImage.cols;
    int r0[3][3] = {-1,0,1,-2,0,2,-1,0,1};
    int r1[3][3] = {0,1,2,-1,0,1,-2,-1,0};
    int r2[3][3] = {1,2,1,0,0,0,-1,-2,-1};
    int r3[3][3] = {2,1,0,1,0,-1,0,-1,-2};
    int r4[3][3] = {1,0,-1,2,0,-2,1,0,-1};
    int r5[3][3] = {0,-1,-2,1,0,-1,2,1,0};
    int r6[3][3] = {-1,-2,-1,0,0,0,1,2,1};
    int r7[3][3] = {-2,-1,0,-1,0,1,0,1,2};
    int g ,temp_g;
    int y0,y1,y2,y3,y4,y5,y6,y7,y8;
    for (int y=0;y<rows;y++){
        for(int x=0;x<cols;x++){
            y0=image.at<Vec3b>(x,y)[0];
            if(y<512)y1=image.at<Vec3b>(x+1,y)[0];else y1=0;
            if(y!=0) y2=image.at<Vec3b>(x,y-1)[0]; else y2=0;
            if(x!=0) y3=image.at<Vec3b>(x-1,y)[0]; else y3=0;
            if(x<512) y4=image.at<Vec3b>(x,y+1)[0]; else y4=0;
            if(x<512&&y<512) y5=image.at<Vec3b>(x+1,y+1)[0]; else y5=0;
            if(y!=0&&x<512) y6=image.at<Vec3b>(x+1,y-1)[0]; else y6=0;
            if(y!=0&&x!=0) y7=image.at<Vec3b>(x-1,y-1)[0]; else y7=0;
            if(x!=0&&y<512) y8=image.at<Vec3b>(x-1,y+1)[0]; else y8=0;
            g = 0;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r0);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r1);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r2);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r3);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r4);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r5);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r6);
            if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_3(y,x,y7,y2,y6,y3,y0,y1,y8,y4,y5,r7);
            if ( temp_g>g ) g=temp_g;

            if(g>threshold){
                tempImage.at<Vec3b>(x,y)[0]=0;
                tempImage.at<Vec3b>(x,y)[1]=0;
                tempImage.at<Vec3b>(x,y)[2]=0;
            }
            else{
                tempImage.at<Vec3b>(x,y)[0]=255;
                tempImage.at<Vec3b>(x,y)[1]=255;
                tempImage.at<Vec3b>(x,y)[2]=255;
            }

        }
    }
    imageHandling=tempImage;
```

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.

```
void runNevatiaBabu55Operator(Mat image,Mat&  imageHandling,int threshold){
    Mat tempImage=image.clone();
    int rows = tempImage.rows;
    int cols = tempImage.cols;
    int r0[5][5] = {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,-100};\
    int r1[5][5] = {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};
    int r2[5][5] = {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};
    int r3[5][5] = {-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};
    int r4[5][5] = {-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};
    int r5[5][5] = {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};
    int g ,temp_g;


    for ( int y = 2 ; y<rows-2 ; y++ ) {
        for ( int x=2 ; x<cols-2 ; x++ ){
            g = 0;
            temp_g = get_gradient_manitude_5(x,y,2,2,r0,image); if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_5(x,y,2,2,r1,image); if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_5(x,y,2,2,r2,image); if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_5(x,y,2,2,r3,image); if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_5(x,y,2,2,r4,image); if ( temp_g>g ) g=temp_g;
            temp_g = get_gradient_manitude_5(x,y,2,2,r5,image); if ( temp_g>g ) g=temp_g;
            if(g>threshold){
                tempImage.at<Vec3b>(x,y)[0]=0;
                tempImage.at<Vec3b>(x,y)[1]=0;
                tempImage.at<Vec3b>(x,y)[2]=0;
            }
            else{
                tempImage.at<Vec3b>(x,y)[0]=255;
                tempImage.at<Vec3b>(x,y)[1]=255;
                tempImage.at<Vec3b>(x,y)[2]=255;
            }       }
    }
    imageHandling=tempImage;

}
```

8.結果

Roberts operator

Prewitt edge detector

Sobel edge detector

Frei and Chen gradient operator

Kirsch compass operator

Robinson compass operator

Nevatia-Babu 5X5 operator