

计算机视觉基础实验报告（二）

1811361 物联网 郭宇

一、实验要求

对于一张给定的图片，实现canny算子对其进行边缘检测。(实现canny算法)

二、实验环境

opencv4.5.4

visual studio 2019 x64

三、实验步骤

canny算法的实现可以分为以下几个步骤：高斯滤波去噪音，计算方向梯度和方向角，计算梯度融合幅值，局部非最大值抑制，双阈值处理，连接边缘。其中高斯滤波可以直接使用函数完成，不再赘述。

3.0 事先声明的一些变量

```
1 Mat imageOriginal = imread("C:/Users/joker/Desktop/test/sikadi.jpg"); //初始图片
2 Mat imageGray; //灰度转换后图片
3 Mat imageGaussian; //高斯模糊后图片
4 Mat imageGradientY; //Y方向差分值
5 Mat imageGradientX; //X方向差分值
6 Mat imageGradient; //梯度计算
7 Mat imageNMS; //非极大值抑制
8 Mat imageLowThreshold; //弱边缘
9 Mat imageHighThreshold; //强边缘
10 Mat imageCanny; //canny算法图像
11 Mat imageResult; //输出结果图像
12 int lowThreshold = 70; //弱边缘
13 int highThreshold = 40; //强边缘
```

3.1 计算方向梯度和方向角

```
1 void GenerateGradient(const Mat& imageSource, Mat& imageSobelX, Mat& imageSobelY, double*& pointDirection) {
2     pointDirection = new double[(imageSource.rows - 1) * (imageSource.cols - 1)];
3
4     imageSobelX = Mat::zeros(imageSource.size(), CV_32SC1);
5     imageSobelY = Mat::zeros(imageSource.size(), CV_32SC1);
6
7     int step = imageSource.step;
8     int stepXY = imageSobelX.step;
9     int rowCount = imageSource.rows;
10    int columnCount = imageSource.cols;
```

```

11
12     for (int i = 1; i < (rowCount - 1); i++) {
13         const uchar* pixelsPreviousRow = imageSource.ptr<uchar>(i - 1);
14         const uchar* pixelsThisRow = imageSource.ptr<uchar>(i);
15         const uchar* pixelsNextRow = imageSource.ptr<uchar>(i + 1);
16         uchar* pixelsThisRow_x = imageSobelX.ptr<uchar>(i);
17         uchar* pixelsThisRow_y = imageSobelY.ptr<uchar>(i);
18         for (int j = 1, k = 0; j < (columnCount - 1); j++, k++) {
19             //通过指针遍历图像上每一个像素
20             double gradY = pixelsPreviousRow[j + 1] + pixelsThisRow[j + 1] *
21 2 + pixelsNextRow[j + 1] -
22             pixelsPreviousRow[j - 1] - pixelsThisRow[j - 1] * 2 -
23             pixelsNextRow[j - 1];
24             double gradX = pixelsNextRow[j - 1] + pixelsNextRow[j] * 2 +
25             pixelsNextRow[j + 1] -
26             pixelsPreviousRow[j - 1] - pixelsPreviousRow[j] * 2 -
27             pixelsPreviousRow[j + 1];
28             pixelsThisRow_x[j * (stepXY / step)] = static_cast<uchar>
29             (abs(gradX));
30             pixelsThisRow_y[j * (stepXY / step)] = static_cast<uchar>
31             (abs(gradY));
32             if (gradX != 0) {
33                 pointDirection[k] = atan(gradY / gradX) * 57.3 + 90; // (- PI
34             / 2, PI / 2)转换到(0, 180)
35             }
36             else {
37                 pointDirection[k] = 180;
38             }
39         }
40     }
41     convertScaleAbs(imageSobelX, imageSobelX);
42     convertScaleAbs(imageSobelY, imageSobelY);
43 }

```

对于每一个像素，通过遍历他周围的像素即可求得X和Y方向的方向梯度，用atan()求其对应的角度。

3.2 计算梯度融合幅值

```

1 void CombineGradient(const Mat& imageGradX, const Mat& imageGradY, Mat&
2 SobelAmpXY) {
3     SobelAmpXY = Mat::zeros(imageGradX.size(), CV_32FC1);
4     for (int i = 0; i < SobelAmpXY.rows; i++) {
5         const uchar* pixelsThisRow_x = imageGradX.ptr<uchar>(i);
6         const uchar* pixelsThisRow_y = imageGradY.ptr<uchar>(i);
7         float* pixelsThisRow_xy = SobelAmpXY.ptr<float>(i);
8         for (int j = 0; j < SobelAmpXY.cols; j++) {
9             const uchar xj = pixelsThisRow_x[j];
10            const uchar yj = pixelsThisRow_y[j];
11            pixelsThisRow_xy[j] = static_cast<float>(sqrt(xj * xj + yj *
12            yj));
13        }
14    }
15    convertScaleAbs(SobelAmpXY, SobelAmpXY);
16 }

```

计算图像梯度能够得到图像的边缘，因为梯度是灰度变化明显的地方，而边缘也是灰度变化明显的地方。当然这一步只能得到可能的边缘。因为灰度变化的地方可能是边缘，也可能不是边缘。这一步就有了所有可能是边缘的集合。

3.3 局部非极大值抑制

```
1 void NMS(const Mat& imageInput, Mat& imageOutput, double* pointDirection) {
2     imageOutput = imageInput.clone();
3     int rowCount = imageInput.rows;
4     int columnCount = imageInput.cols;
5     for (int i = 1; i < rowCount - 1; i++) {
6         uchar* pixelsPreviousRow = imageOutput.ptr<uchar>(i - 1);
7         uchar* pixelsThisRow = imageOutput.ptr<uchar>(i);
8         uchar* pixelsNextRow = imageOutput.ptr<uchar>(i + 1);
9         for (int j = 1, k = 0; j < columnCount - 1; j++, k++) {
10             double tPD = tan(pointDirection[i * (columnCount - 1) + j]);
11             double tPD_180 = tan(180 - pointDirection[i * (columnCount - 1)
+ j]);
12
13             if (pointDirection[k] <= 45) {
14                 if (pixelsThisRow[j] <=
15                     (pixelsThisRow[j + 1] + (pixelsPreviousRow[j + 1] -
pixelsThisRow[j + 1]) * tPD) ||
16                     (pixelsThisRow[j] <=
17                     (pixelsThisRow[j - 1] + (pixelsNextRow[j - 1] -
pixelsThisRow[j - 1]) * tPD))) {
18                     pixelsThisRow[j] = 0;
19                 }
20             }
21             else if (pointDirection[k] <= 90) {
22                 if (pixelsThisRow[j] <=
23                     (pixelsPreviousRow[j] + (pixelsPreviousRow[j + 1] -
pixelsPreviousRow[j]) / tPD) ||
24                     pixelsThisRow[j] <= (pixelsNextRow[j] + (pixelsNextRow[j
- 1] - pixelsNextRow[j]) / tPD)) {
25                     pixelsThisRow[j] = 0;
26                 }
27             }
28             else if (pointDirection[k] <= 135) {
29                 if (pixelsThisRow[j] <=
30                     (pixelsPreviousRow[j] + (pixelsPreviousRow[j - 1] -
pixelsPreviousRow[j]) / tPD_180) ||
31                     pixelsThisRow[j] <= (pixelsNextRow[j] + (pixelsNextRow[j
+ 1] - pixelsNextRow[j]) / tPD_180)) {
32                     pixelsThisRow[j] = 0;
33                 }
34             }
35             else if (pointDirection[k] <= 180) {
36                 if (pixelsThisRow[j] <=
37                     (pixelsThisRow[j - 1] + (pixelsPreviousRow[j - 1] -
pixelsThisRow[j - 1]) * tPD_180) ||
38                     pixelsThisRow[j] <= (pixelsThisRow[j + 1] +
(pixelsNextRow[j + 1] - pixelsThisRow[j]) * tPD_180)) {
39                     pixelsThisRow[j] = 0;
40                 }
41             }
42             else {
```

```

43         cout << "Invalid pointDirection: " << pointDirection[k] <<
endl;
44     }
45 }
46 }
47 }

```

常灰度变化的地方都比较集中，将局部范围内的梯度方向上，灰度变化最大的保留下来，其它的不保留，这样可以剔除掉一大部分的点。将多个像素宽的边缘变成一个单像素宽的边缘。即“胖边缘”变成“瘦边缘”。

3.4 双阈值处理

```

1 void SplitwithThreshold(const Mat& imageInput, Mat& lowOutput, Mat&
highOutput, double lowThreshold, double highThreshold) {
2     lowOutput = imageInput.clone();
3     highOutput = imageInput.clone();
4     int rowCount = imageInput.rows;
5     int columnCount = imageInput.cols;
6     for (int i = 0; i < rowCount; i++) {
7         const uchar* pixelsThisRow = imageInput.ptr<uchar>(i);
8         uchar* pixelsThisRow_low = lowOutput.ptr<uchar>(i);
9         uchar* pixelsThisRow_high = highOutput.ptr<uchar>(i);
10        for (int j = 0; j < columnCount; j++) {
11            uchar pixel = pixelsThisRow[j];
12            if (pixel >= highThreshold) {
13                pixelsThisRow_high[j] = 255;
14            }
15            else {
16                pixelsThisRow_high[j] = 0;
17                if (pixel <= lowThreshold) {
18                    pixelsThisRow_low[j] = 0;
19                }
20                else {
21                    pixelsThisRow_low[j] = 255;
22                }
23            }
24        }
25    }
26 }

```

通过非极大值抑制后，仍然有很多的可能边缘点，进一步的设置一个双阈值，即低阈值（low），高阈值（high）。灰度变化大于high的，设置为强边缘像素，低于low的，剔除。在low和high之间的设置为弱边缘。进一步判断，如果其领域内有强边缘像素，保留，如果没有，剔除。

3.5 连接边缘

```

1 void LinkEdge(Mat& imageOutput, const Mat& lowThresImage, const Mat&
highThresImage) {
2     imageOutput = highThresImage.clone();
3     int rowCount = imageOutput.rows;
4     int columnCount = imageOutput.cols;
5     // 为计算方便，牺牲图像四周1像素宽的一圈
6     for (int i = 1; i < rowCount - 1; i++) {
7         uchar* pixelsPreviousRow = imageOutput.ptr<uchar>(i - 1);

```

```

8     uchar* pixelsThisRow = imageOutput.ptr<uchar>(i);
9     uchar* pixelsNextRow = imageOutput.ptr<uchar>(i + 1);
10    for (int j = 1; j < columnCount - 1; j++) {
11        if (pixelsThisRow[j] == 255) {
12            GoAhead(i, j, pixelsPreviousRow, pixelsThisRow,
pixelsNextRow, lowThresImage, imageOutput);
13        }
14        if (pixelsNextRow[j - 1] == 255) {
15            GoAhead(i + 1, j - 1, pixelsThisRow, pixelsNextRow,
imageOutput.ptr<uchar>(i + 1), lowThresImage,
16            imageOutput);
17        }
18        if (pixelsNextRow[j] == 255) {
19            GoAhead(i + 1, j, pixelsThisRow, pixelsNextRow,
imageOutput.ptr<uchar>(i + 1), lowThresImage,
20            imageOutput);
21        }
22        if (pixelsNextRow[j + 1] == 255) {
23            GoAhead(i + 1, j + 1, pixelsThisRow, pixelsNextRow,
imageOutput.ptr<uchar>(i + 1), lowThresImage,
24            imageOutput);
25        }
26    }
27 }
28 }

```

```

1 void GoAhead(int i, int j, uchar* pixelsPreviousRow, uchar* pixelsThisRow,
uchar* pixelsNextRow, const Mat& lowThresImage, Mat& imageOutput) {
2     // 判断左下方、右方、下方和右下方是否接续
3     if (pixelsThisRow[j + 1] != 255 && pixelsNextRow[j + 1] != 255 &&
pixelsNextRow[j] != 255 &&
4     pixelsNextRow[j - 1] != 255) {
5         // 若不接续，从低阈值图中查找8领域是否接续，并对左上方、上方、右上方和左上方递归
调用自身
6         const uchar* pixelsPreviousRow_low = lowThresImage.ptr<uchar>(i -
1);
7         const uchar* pixelsThisRow_low = lowThresImage.ptr<uchar>(i);
8         const uchar* pixelsNextRow_low = lowThresImage.ptr<uchar>(i + 1);
9         // 左上
10        if (pixelsPreviousRow_low[j - 1] == 255) {
11            pixelsPreviousRow[j - 1] = 255;
12            if (i != 0 && j != 0) {
13                GoAhead(i - 1, j - 1, imageOutput.ptr<uchar>(i - 1),
pixelsPreviousRow, pixelsThisRow, lowThresImage,
14                imageOutput);
15            }
16        }
17        // 上
18        if (pixelsPreviousRow_low[j] == 255) {
19            pixelsPreviousRow[j] = 255;
20            if (i != 0) {
21                GoAhead(i - 1, j, imageOutput.ptr<uchar>(i - 1),
pixelsPreviousRow, pixelsThisRow, lowThresImage,
22                imageOutput);
23            }
24        }
25        // 右上

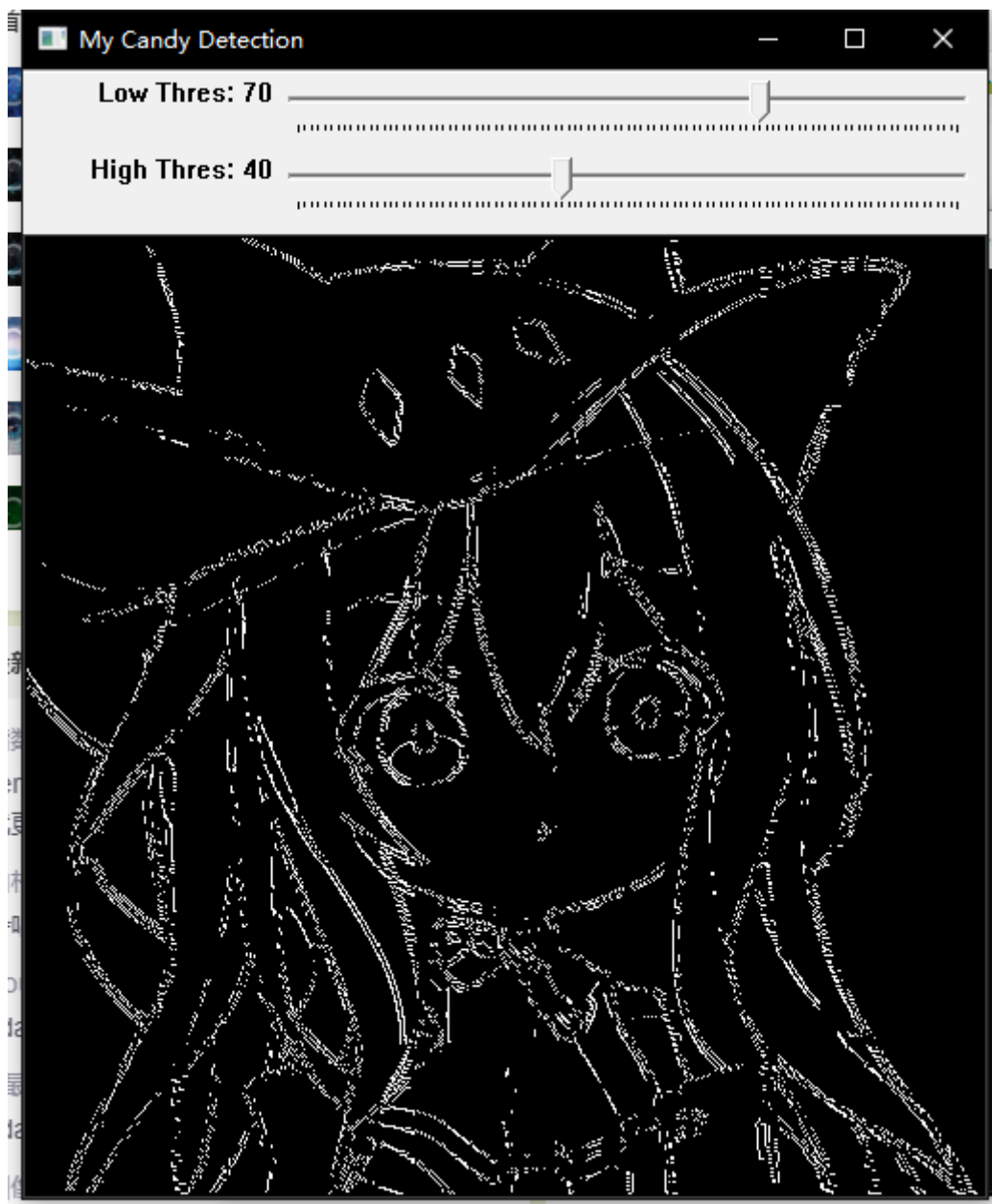
```

```

26         if (pixelsPreviousRow_low[j + 1] == 255) {
27             pixelsPreviousRow[j + 1] = 255;
28             if (i != 0 && j != imageOutput.cols) {
29                 GoAhead(i - 1, j + 1, imageOutput.ptr<uchar>(i - 1),
pixelsPreviousRow, pixelsThisRow, lowThresImage,
30                     imageOutput);
31             }
32         }
33         // 左
34         if (pixelsThisRow_low[j - 1] == 255) {
35             pixelsThisRow[j - 1] = 255;
36             if (i != 0 && j != 0) {
37                 GoAhead(i - 1, j - 1, imageOutput.ptr<uchar>(i - 1),
pixelsPreviousRow, pixelsThisRow, lowThresImage,
38                     imageOutput);
39             }
40         }
41         // 右
42         if (pixelsThisRow_low[j + 1] == 255) {
43             pixelsThisRow[j + 1] = 255;
44         }
45         // 左下
46         if (pixelsNextRow_low[j - 1] == 255) {
47             pixelsNextRow[j - 1] = 255;
48         }
49         // 下
50         if (pixelsNextRow_low[j] == 255) {
51             pixelsNextRow[j] = 255;
52         }
53         // 右下
54         if (pixelsNextRow_low[j + 1] == 255) {
55             pixelsNextRow[j + 1] = 255;
56         }
57     }
58 }

```

四、实验结果



五、实验感想

虽然可以直接调用opencv里自带的函数canny进行边缘检测，但这么写理解了canny的具体的工作流程，也知道了函数里的各个参数都是干什么用的。