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

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

3.1 计算方向梯度和方向角

```
void GenerateGradient(const Mat& imageSource, Mat& imageSobelX, Mat& imageSobelY, double*& pointDirection) {
    pointDirection = new double[(imageSource.rows - 1) * (imageSource.cols - 1)];

    imageSobelX = Mat::zeros(imageSource.size(), CV_32SC1);
    imageSobelY = Mat::zeros(imageSource.size(), CV_32SC1);

    int step = imageSource.step;
    int stepXY = imageSobelX.step;
    int rowCount = imageSource.rows;
    int columnCount = imageSource.cols;
```

```

    for (int i = 1; i < (rowCount - 1); i++) {
        const uchar* pixelsPreviousRow = imageSource.ptr<uchar>(i - 1);
        const uchar* pixelsThisRow = imageSource.ptr<uchar>(i);
        const uchar* pixelsNextRow = imageSource.ptr<uchar>(i + 1);
        uchar* pixelsThisRow_x = imageSobelX.ptr<uchar>(i);
        uchar* pixelsThisRow_y = imageSobelY.ptr<uchar>(i);
        for (int j = 1, k = 0; j < (columnCount - 1); j++, k++) {
            //通过指针遍历图像上每一个像素
            double gradY = pixelsPreviousRow[j + 1] + pixelsThisRow[j + 1] * 2 + pixelsNextRow[j + 1] -
                pixelsPreviousRow[j - 1] - pixelsThisRow[j - 1] * 2 - pixelsNextRow[j - 1];
            double gradX = pixelsNextRow[j - 1] + pixelsNextRow[j] * 2 + pixelsNextRow[j + 1] -
                pixelsPreviousRow[j - 1] - pixelsPreviousRow[j] * 2 - pixelsPreviousRow[j + 1];
            pixelsThisRow_x[j * (stepXY / step)] = static_cast<uchar>(abs(gradX));
            pixelsThisRow_y[j * (stepXY / step)] = static_cast<uchar>(abs(gradY));
            if (gradX != 0) {
                pointDirection[k] = atan(gradY / gradX) * 57.3 + 90;// (- PI / 2, PI / 2)转换到(0, 180)
            }
            else {
                pointDirection[k] = 180;
            }
        }
    }
    convertScaleAbs(imageSobelX, imageSobelX);
    convertScaleAbs(imageSobelY, imageSobelY);
}
```

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

```
void CombineGradient(const Mat& imageGradX, const Mat& imageGradY, Mat& SobelAmpXY) {
    SobelAmpXY = Mat::zeros(imageGradX.size(), CV_32FC1);
    for (int i = 0; i < SobelAmpXY.rows; i++) {
        const uchar* pixelsThisRow_x = imageGradX.ptr<uchar>(i);
        const uchar* pixelsThisRow_y = imageGradY.ptr<uchar>(i);
        float* pixelsThisRow_xy = SobelAmpXY.ptr<float>(i);
        for (int j = 0; j < SobelAmpXY.cols; j++) {
            const uchar xj = pixelsThisRow_x[j];
            const uchar yj = pixelsThisRow_y[j];
            pixelsThisRow_xy[j] = static_cast<float>(sqrt(xj * xj + yj * yj));
        }
    }
    convertScaleAbs(SobelAmpXY, SobelAmpXY);
}
```

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

```
void NMS(const Mat& imageInput, Mat& imageOutput, double* pointDirection) {
    imageOutput = imageInput.clone();
    int rowCount = imageInput.rows;
    int columnCount = imageInput.cols;
    for (int i = 1; i < rowCount - 1; i++) {
        uchar* pixelsPreviousRow = imageOutput.ptr<uchar>(i - 1);
        uchar* pixelsThisRow = imageOutput.ptr<uchar>(i);
        uchar* pixelsNextRow = imageOutput.ptr<uchar>(i + 1);
        for (int j = 1, k = 0; j < columnCount - 1; j++, k++) {
            double tPD = tan(pointDirection[i * (columnCount - 1) + j]);
            double tPD_180 = tan(180 - pointDirection[i * (columnCount - 1) + j]);

            if (pointDirection[k] <= 45) {
                if (pixelsThisRow[j] <=
                    (pixelsThisRow[j + 1] + (pixelsPreviousRow[j + 1] - pixelsThisRow[j + 1]) * tPD) ||
                    (pixelsThisRow[j] <=
                        (pixelsThisRow[j - 1] + (pixelsNextRow[j - 1] - pixelsThisRow[j - 1]) * tPD))) {
                    pixelsThisRow[j] = 0;
                }
            }
            else if (pointDirection[k] <= 90) {
                if (pixelsThisRow[j] <=
                    (pixelsPreviousRow[j] + (pixelsPreviousRow[j + 1] - pixelsPreviousRow[j]) / tPD) ||
                    pixelsThisRow[j] <= (pixelsNextRow[j] + (pixelsNextRow[j - 1] - pixelsNextRow[j]) / tPD)) {
                    pixelsThisRow[j] = 0;
                }
            }
            else if (pointDirection[k] <= 135) {
                if (pixelsThisRow[j] <=
                    (pixelsPreviousRow[j] + (pixelsPreviousRow[j - 1] - pixelsPreviousRow[j]) / tPD_180) ||
                    pixelsThisRow[j] <= (pixelsNextRow[j] + (pixelsNextRow[j + 1] - pixelsNextRow[j]) / tPD_180)) {
                    pixelsThisRow[j] = 0;
                }
            }
            else if (pointDirection[k] <= 180) {
                if (pixelsThisRow[j] <=
                    (pixelsThisRow[j - 1] + (pixelsPreviousRow[j - 1] - pixelsThisRow[j - 1]) * tPD_180) ||
                    pixelsThisRow[j] <= (pixelsThisRow[j + 1] + (pixelsNextRow[j + 1] - pixelsThisRow[j]) * tPD_180)) {
                    pixelsThisRow[j] = 0;
                }
            }
            else {
```

```
                cout << "Invalid pointDirection: " << pointDirection[k] << endl;
            }
        }
    }
}
```

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。将有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

```
void SplitWithThreshold(const Mat& imageInput, Mat& lowOutput, Mat& highOutput, double lowThreshold, double highThreshold) {
    lowOutput = imageInput.clone();
    highOutput = imageInput.clone();
    int rowCount = imageInput.rows;
    int columnCount = imageInput.cols;
    for (int i = 0; i < rowCount; i++) {
        const uchar* pixelsThisRow = imageInput.ptr<uchar>(i);
        uchar* pixelsThisRow_low = lowOutput.ptr<uchar>(i);
        uchar* pixelsThisRow_high = highOutput.ptr<uchar>(i);
        for (int j = 0; j < columnCount; j++) {
            uchar pixel = pixelsThisRow[j];
            if (pixel >= highThreshold) {
                pixelsThisRow_high[j] = 255;
            }
            else {
                pixelsThisRow_high[j] = 0;
                if (pixel <= lowThreshold) {
                    pixelsThisRow_low[j] = 0;
                }
                else {
                    pixelsThisRow_low[j] = 255;
                }
            }
        }
    }
}
```

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

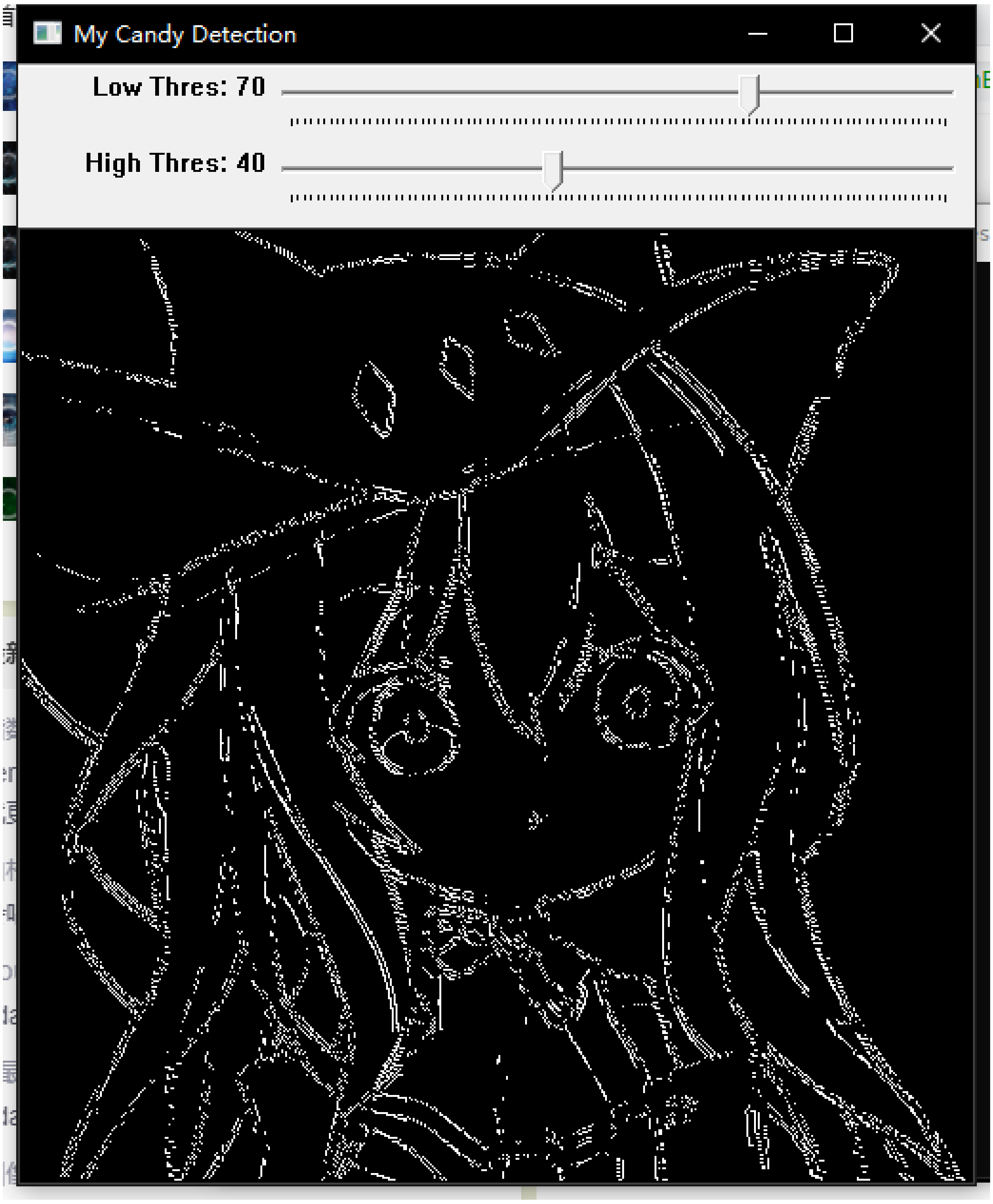
```
void LinkEdge(Mat& imageOutput, const Mat& lowThresImage, const Mat& highThresImage) {
    imageOutput = highThresImage.clone();
    int rowCount = imageOutput.rows;
    int columnCount = imageOutput.cols;
    // 为计算方便，牺牲图像四周1像素宽的一圈
    for (int i = 1; i < rowCount - 1; i++) {
        uchar* pixelsPreviousRow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

```
        uchar* pixelsThisRow = imageOutput.ptr<uchar>(i);
        uchar* pixelsNextRow = imageOutput.ptr<uchar>(i + 1);
        for (int j = 1; j < columnCount - 1; j++) {
            if (pixelsThisRow[j] == 255) {
                GoAhead(i, j, pixelsPreviousRow, pixelsThisRow, pixelsNextRow, lowThresImage, imageOutput);
            }
            if (pixelsNextRow[j - 1] == 255) {
                GoAhead(i + 1, j - 1, pixelsThisRow, pixelsNextRow, imageOutput.ptr<uchar>(i + 1), lowThresImage,
                    imageOutput);
            }
            if (pixelsNextRow[j] == 255) {
                GoAhead(i + 1, j, pixelsThisRow, pixelsNextRow, imageOutput.ptr<uchar>(i + 1), lowThresImage,
                    imageOutput);
            }
            if (pixelsNextRow[j + 1] == 255) {
                GoAhead(i + 1, j + 1, pixelsThisRow, pixelsNextRow, imageOutput.ptr<uchar>(i + 1), lowThresImage,
                    imageOutput);
            }
        }
    }
}
```

```
void GoAhead(int i, int j, uchar* pixelsPreviousRow, uchar* pixelsThisRow, uchar* pixelsNextRow, const Mat& lowThresImage, Mat& imageOutput) {
    // 判断左下方、右方、下方和右下方是否接续
    if (pixelsThisRow[j + 1] != 255 && pixelsNextRow[j + 1] != 255 && pixelsNextRow[j] != 255 &&
        pixelsNextRow[j - 1] != 255) {
        // 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
        const uchar* pixelsPreviousRow_low = lowThresImage.ptr<uchar>(i - 1);
        const uchar* pixelsThisRow_low = lowThresImage.ptr<uchar>(i);
        const uchar* pixelsNextRow_low = lowThresImage.ptr<uchar>(i + 1);
        // 左上
        if (pixelsPreviousRow_low[j - 1] == 255) {
            pixelsPreviousRow[j - 1] = 255;
            if (i != 0 && j != 0) {
                GoAhead(i - 1, j - 1, imageOutput.ptr<uchar>(i - 1), pixelsPreviousRow, pixelsThisRow, lowThresImage,
                    imageOutput);
            }
        }
        // 上
        if (pixelsPreviousRow_low[j] == 255) {
            pixelsPreviousRow[j] = 255;
            if (i != 0) {
                GoAhead(i - 1, j, imageOutput.ptr<uchar>(i - 1), pixelsPreviousRow, pixelsThisRow, lowThresImage,
                    imageOutput);
            }
        }
        // 右上
```

```
        if (pixelsPreviousRow_low[j + 1] == 255) {
            pixelsPreviousRow[j + 1] = 255;
            if (i != 0 && j != imageOutput.cols) {
                GoAhead(i - 1, j + 1, imageOutput.ptr<uchar>(i - 1), pixelsPreviousRow, pixelsThisRow, lowThresImage,
                    imageOutput);
            }
        }
        // 左
        if (pixelsThisRow_low[j - 1] == 255) {
            pixelsThisRow[j - 1] = 255;
            if (i != 0 && j != 0) {
                GoAhead(i - 1, j - 1, imageOutput.ptr<uchar>(i - 1), pixelsPreviousRow, pixelsThisRow, lowThresImage,
                    imageOutput);
            }
        }
        // 右
        if (pixelsThisRow_low[j + 1] == 255) {
            pixelsThisRow[j + 1] = 255;
        }
        // 左下
        if (pixelsNextRow_low[j - 1] == 255) {
            pixelsNextRow[j - 1] = 255;
        }
        // 下
        if (pixelsNextRow_low[j] == 255) {
            pixelsNextRow[j] = 255;
        }
        // 右下
        if (pixelsNextRow_low[j + 1] == 255) {
            pixelsNextRow[j + 1] = 255;
        }
    }
}
```

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。