异步图书
www.epubit.com

图像局部特征检测和描述

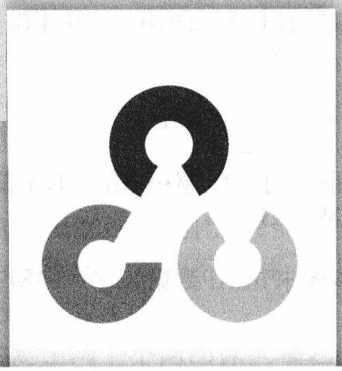
基于OpenCV源码分析的算法与实现

赵春江◎编著

- 对OpenCV所实现的图像特征检测和描述算法进行了全面的讲解，如 Kitchen-Rosenfeld、Canny、Harris、Shi-Tomasi、FAST、MSER、MSCR、SIFT、SURF、BRISK、BRIEF、ORB、FREAK、CenSurE 等。

- 不仅分析了算法的原理和实现方法，还进行了详细的源码解析，并给出具体的程序实现范例，充分体现了理论与实践相结合的特点。

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS



图像局部特征检测和描述

基于OpenCV源码分析的算法与实现

赵春江◎编著

人民邮电出版社
北京

图书在版编目（CIP）数据

图像局部特征检测和描述 ：基于OpenCV源码分析的算法与实现 / 赵春江编著. -- 北京 ：人民邮电出版社, 2018.7
ISBN 978-7-115-46171-1

Ⅰ. ①图… Ⅱ. ①赵… Ⅲ. ①图象处理软件－程序设计 Ⅳ. ①TP391.413

中国版本图书馆CIP数据核字(2018)第051270号

内 容 提 要

在计算机视觉处理中，特征指的是能够解决某种特定任务的信息。图像局部特征在目标识别、目标跟踪、目标匹配、三维重建、图像检索等应用中发挥着重要的作用。它是近 20 年来在计算机视觉领域中研究的热点问题之一。

本书以 OpenCV 2.4.9 为研究工具，对其实现的所有最新的特征检测和描述算法—Kitchen-Rosenfeld、Canny、Harris、Shi-Tomasi、FAST、MSER、MSCR、SIFT、SURF、BRISK、BRIEF、ORB、FREAK、CenSurE、SimpleBlob 等，不仅详细分析了它们的原理和实现方法，还进行了详细的源码解析，并且给出了具体的程序实现范例，充分体现了理论与实践相结合的特点。

本书适合计算机视觉和图像处理领域的工程技术人员阅读，也可供高等院校相关专业师生参考。

◆ 编　　著　赵春江
责任编辑　张　爽
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：18.5
字数：448 千字　　2018 年 7 月第 1 版
印数：1－2 400 册　　2018 年 7 月河北第 1 次印刷

定价：69.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

前　言

图像的局部特征是区别于它的邻域的一种图像模式，它往往与一个图像特性（强度、颜色、纹理等）的变化或多个图像特性的同时变化联系在一起，虽然这种变化并不一定是局部的。相对于局部特征，全局特征在目标识别、目标跟踪、目标匹配等领域并不适用，因为全局特征无法区分前景和背景，而是把这两部分的信息全部混合在一起。

图像的局部特征一般包括边缘、角点和斑点。边缘是由两个不同图像区域之间的边界点所形成的。一般来说，边缘可以是任意形状的，当然也可能包括节点。在实际应用中，边缘往往被定义成图像局部强度变化最剧烈的部分。角点具有点状特性，早期的角点被定义为两个边缘的交点，后来发现只要通过寻找图像梯度的高曲率部分，就能够得到角点，因此角点检测不再需要边缘检测。但是用这种方法得到的角点也有可能是图像的其他部分，如暗背景下的亮斑，所以角点被赋予了更深层次的含义。斑点给出了描述图像区域结构的另一种方式，虽然与角点相比，斑点的点状结构不明显，但斑点检测仍然可以使用检测兴趣点的运算符的方法，而且它可以检测到那些被角点检测认为很平滑的区域。

好的局部特征应该具备以下几个特点。

（1）可重复性：具有相同目标的两幅图像，尽管观看的视角和方式不同，但应该在这两幅图像中有高比例的目标特征被分别检测到。

（2）独特性：特征应该具有很大的信息量，以至于能够区别和匹配不同的特征。

（3）局部性：为了减少目标因为被部分遮挡，或几何、光度变形而产生的影响，特征一定要是局部特征。

（4）数量多：目标的特征一定要达到一定的数量，这样才能有理由相信各种基于图像特征应用的结果。

（5）准确性：准确性不仅包括特征的位置，还要包括特征的可能变形。

（6）效率：尤其是在当前计算机视觉向嵌入式、实时性方向发展的背景下，对算法效率提出了更高的要求。

从上面的分析可以看出，特征的可重复性是所有特点中最重要的。这种可重复性可以从两个不同方面去衡量：不变性和鲁棒性。局部特征的不变性指的是特征不随图像的尺度、位移、旋转、视角、光照等因素的变化而变化；鲁棒性指的是特征检测对图像的某种变形不敏感，尽管准确性下降了，但下降得不是很剧烈，而引起图像变形的原因有图像噪声、离散效应、压缩效应、模糊，以及数学建模时的几何、光度偏差等。

特征检测和特征描述是各种基于局部特征应用的两个不同阶段。先进行局部特征的检测，然后把检测到的特征按照一种紧凑的、完整的、归一化的形式表现出来，这种量化的数据描述形式被称为描述符（或描述子）。描述符往往是基于图像处理的知识，把围绕特征周围区域的

相关信息提取出来而最终形成的。因此，特征描述与特征检测具有同样重要的作用。

OpenCV（Open Source Computer Vision）是一个主要应用于实时计算机视觉领域的程序函数库，该函数库包括了超过 500 个优化的图像和视频分析处理的算法。OpenCV 是以 BSD 许可证授权发行的，程序源码完全对用户开放，它既可以直接免费使用，又可以作为二次开发的工具。OpenCV 有 C++、C、Python 和 Java 接口，并且支持 Windows、Linux、Mac OS、iOS 和安卓系统。OpenCV 最初由英特尔公司发起并参与开发，目前由 Willow Garage 公司和 Itseez 公司共同维护。

由于 OpenCV 具有开源、接口丰富、移植性好、算法优化、运算能力强等特点，使它既可以应用于商业项目的开发，又可以作为科研工作者的工具，开发、验证计算机视觉算法。在后一个应用领域中，它的作用已经与 MATLAB 相似，而且目前有迹象表明，在计算机视觉领域，OpenCV 已逐渐取代了 MATLAB 在这个领域的地位，在最近的科技文献中，有大量的算法仿真都是基于 OpenCV 的。而且更重要的是，OpenCV 中一些算法的程序直接就是由该算法的提出者编写完成的。

对于算法开发者来说，阅读文献是必须要经历的一个过程。但对于初学者来说，有些文献晦涩难懂，往往读了几遍后也不知所云，这不仅是因为语言的障碍，背景知识的欠缺，更重要的是由于论文篇幅所限，抑或是从作者的角度来看不重要，导致算法的某些内容没有被详细披露。而即使有幸读懂，但要想自己实现该算法，那又是另一回事，尤其是对于那些编程能力不强的人来说。算法都无法实现，那更谈不上在此基础上的修改及创新了。

如果能够有算法的实现源码，那么在阅读文献的同时，阅读程序代码，这种方法绝对能够起到更好更快地理解和掌握算法的作用。先阅读文献，掌握算法的基本思想、原理、步骤等内容，再阅读源码，理解算法的细节实现部分，然后回过头来再阅读文献，看一看当初对算法的理解是否有偏差。就这样反复多次，一定能够掌握到算法的精髓。OpenCV 开源的特点为我们能够阅读到算法源码提供了绝佳的机会。

图像局部特征的检测和描述是目前计算机视觉领域一个研究热点问题。OpenCV 几乎包括了当今所有的特征检测与描述的算法，从 SIFT、SURF 到 ORB、FREAK。这些代码既实现了优化，又不失可读性。阅读分析这些代码既有助于理解 OpenCV 的编程思想，为我们自己编写基于 OpenCV 的程序打下基础，又可以帮助我们理解特征检测与描述的相关算法，真可谓是一举两得。

本书正是基于这个目的而撰写的，它基本上包括了 OpenCV 2.4.9 中实现的所有局部特征检测和描述算法——Kitchen-Rosenfeld、Canny、Harris、Shi-Tomasi、FAST、MSER、MSCR、SIFT、SURF、BRISK、BRIEF、ORB、FREAK、CenSurE、SimpleBlob 等。对于每一种算法，都先从原理开始介绍，然后是 OpenCV 的源码分析，最后给出了一个基于 OpenCV 的具体应用实例。对算法原理的介绍，不仅仅是文献的简单翻译重复，还加入了我们对算法的理解，更重要的是，原理的分析是从实现的角度去介绍，更注重对算法所用到的一些背景知识以及细节上的讲解。在源码分析部分，我们基本上做到了对每一条代码都给出了较详细的注解。应用实例的程序虽然简单，但足以说明问题，读者完全可以在此基础上编写更复杂的程序代码。因此可以说，这本书既突出算法的理论，又包括实践部分。

本书是作者利用工作之余时间撰写完成的，加之作者水平有限，难免会有对算法和源码理解分析不对的地方，权当抛砖引玉，恳请读者批评指正。本书编辑联系和投稿邮箱 zhangtao@ptpress.com.cn。

作者

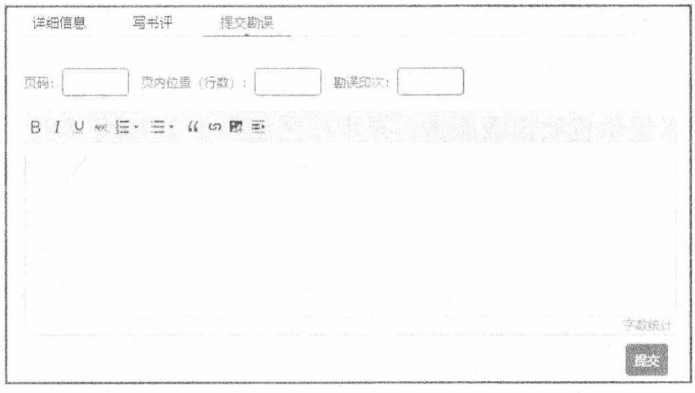
资源与支持

本书由异步社区出品，社区（https://www.epubit.com/）为您提供相关资源和后续服务。

提交勘误

作者和编辑尽最大努力来确保书中内容的准确性，但难免会存在疏漏。欢迎您将发现的问题反馈给我们，帮助我们提升图书的质量。

当您发现错误时，请登录异步社区，按书名搜索，进入本书页面，点击“提交勘误”，输入勘误信息，点击“提交”按钮即可。本书的作者和编辑会对您提交的勘误进行审核，确认并接受后，您将获赠异步社区的 100 积分。积分可用于在异步社区兑换优惠券、样书或奖品。



扫码关注本书

扫描下方二维码，您将会在异步社区微信服务号中看到本书信息及相关的服务提示。

与我们联系

我们的联系邮箱是 contact@epubit.com.cn。

如果您对本书有任何疑问或建议，请您发邮件给我们，并请在邮件标题中注明本书书名，以便我们更高效地做出反馈。

如果您有兴趣出版图书、录制教学视频，或者参与图书翻译、技术审校等工作，可以发邮件给我们；有意出版图书的作者也可以到异步社区在线提交投稿（直接访问 www.epubit.com/selfpublish/submission 即可）。

如果您是学校、培训机构或企业，想批量购买本书或异步社区出版的其他图书，也可以发邮件给我们。

如果您在网上发现有针对异步社区出品图书的各种形式的盗版行为，包括对图书全部或部分内容的非授权传播，请您将怀疑有侵权行为的链接发邮件给我们。您的这一举动是对作者权益的保护，也是我们持续为您提供有价值的内容的动力之源。

关于异步社区和异步图书

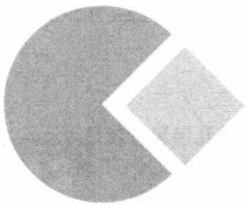
“异步社区”是人民邮电出版社旗下 IT 专业图书社区，致力于出版精品 IT 技术图书和相关学习产品，为作译者提供优质出版服务。异步社区创办于 2015 年 8 月，提供大量精品 IT 技术图书和电子书，以及高品质技术文章和视频课程。更多详情请访问异步社区官网 https://www.epubit.com。

“异步图书”是由异步社区编辑团队策划出版的精品 IT 专业图书的品牌，依托于人民邮电出版社近 30 年的计算机图书出版积累和专业编辑团队，相关图书在封面上印有异步图书的 LOGO。异步图书的出版领域包括软件开发、大数据、AI、测试、前端、网络技术等。

异步社区

微信服务号

目　录



第1章 Kitchen-Rosenfeld角点检测

1.1 原理分析

Kitchen和Rosenfeld认为角点是那些边缘曲线曲率和梯度幅值都很大的点，因此他们提出了使用曲率 k 与梯度幅值 g 的乘积来计算角点响应函数 C 的方法：

$$C = kg = k\left(I_x^2 + I_y^2\right)^{1/2} = \frac{I_{xx}I_y^2 + I_{yy}I_x^2 - 2I_{xy}I_xI_y}{I_x^2 + I_y^2} \tag{1-1}$$

C 的极值所对应的像素即为角点。式（1-1）中，I_x、I_y 为图像 I 的一阶导数，即：

$$I_x = \frac{\partial I}{\partial x},\ I_y = \frac{\partial I}{\partial y} \tag{1-2}$$

式（1-1）中，I_{xx}、I_{yy} 和 I_{xy} 为图像的二阶偏导，即：

$$I_{xx} = \frac{\partial^2 I}{\partial x^2},\ I_{yy} = \frac{\partial^2 I}{\partial y^2},\ I_{xy} = \frac{\partial^2 I}{\partial xy} \tag{1-3}$$

由于式（1-1）中的分母始终大于 0，它不改变角点响应函数的相对值，因此在实际应用中，往往只计算分子部分。

Kitchen-Rosenfeld角点检测方法的步骤如下：

（1）计算图像的一阶、二阶导数；

（2）利用式（1-1）计算角点响应函数；

（3）对角点响应函数进行非极大值抑制，并设定阈值，得到角点。

由于该方法使用了基于灰度的二阶偏导，所以它对噪声比较敏感。

1.2 源码解析

OpenCV中实现Kitchen-Rosenfeld角点检测的函数为preCornerDetect，该函数只实现了计算角点响应函数 C，并没有完成非极大值抑制。

preCornerDetect函数的原型为：

```
void preCornerDetect(InputArray src, OutputArray dst, int ksize, int border- Type= BORDER_
DEFAULT )
src 为输入单通道 8 位整型或 32 位浮点型图像
dst 为输出 32 位浮点型图像，尺寸大小与 src 相同
ksize 表示孔径尺寸，preCornerDetect 函数内部是使用 Sobel 方法进行导数计算的，因此该孔径尺寸就是 Sobel
算子的尺寸
border-Type 为扩展图像边界的方式
preCornerDetect 函数在 sources/modules/imgproc/src/corner.cpp 文件内定义：
void cv::preCornerDetect( InputArray _src, OutputArray _dst, int ksize, int borderType )
{
    //定义各种矩形变量
    Mat Dx, Dy, D2x, D2y, Dxy, src = _src.getMat();
    //确保输入图像类型的正确性
    CV_Assert( src.type() == CV_8UC1 || src.type() == CV_32FC1 );
    _dst.create( src.size(), CV_32F );
    Mat dst = _dst.getMat();    //得到输出图像
    //利用 Sobel 算子计算图像的一阶、二阶导数
    //Dx 为 I_x，Dy 为 I_y，D2x 为 I_xx，D2y 为 I_yy，Dxy 为 I_xy
    Sobel( src, Dx, CV_32F, 1, 0, ksize, 1, 0, borderType );
    Sobel( src, Dy, CV_32F, 0, 1, ksize, 1, 0, borderType );
    Sobel( src, D2x, CV_32F, 2, 0, ksize, 1, 0, borderType );
    Sobel( src, D2y, CV_32F, 0, 2, ksize, 1, 0, borderType );
    Sobel( src, Dxy, CV_32F, 1, 1, ksize, 1, 0, borderType );
    //定义一个系数 factor，用于限制输出数据的大小
    double factor = 1 << (ksize - 1);
    if( src.depth() == CV_8U )
        factor *= 255;
    factor = 1./(factor * factor * factor);

    Size size = src.size();    //图像尺寸大小
    int i, j;
    //遍历图像的所有像素
    for( i = 0; i < size.height; i++ )
    {
        //分别得到输出图像、图像一阶导数和二阶导数的当前行的首地址指针
        float* dstdata = (float*)(dst.data + i*dst.step);
        const float* dxdata = (const float*)(Dx.data + i*Dx.step);
        const float* dydata = (const float*)(Dy.data + i*Dy.step);
        const float* d2xdata = (const float*)(D2x.data + i*D2x.step);
        const float* d2ydata = (const float*)(D2y.data + i*D2y.step);
        const float* dxydata = (const float*)(Dxy.data + i*Dxy.step);

        for( j = 0; j < size.width; j++ )
        {
            //dx 和 dy 分别为当前像素的水平方向和垂直方向的一阶导数
            float dx = dxdata[j];
            float dy = dydata[j];
            //计算式 1-1 的分子部分
            dstdata[j] = (float)(factor*(dx*dx*d2ydata[j] + dy*dy*d2xdata[j] -
            2*dx*dy*dxydata[j]));
```

```
        }
    }
}
```

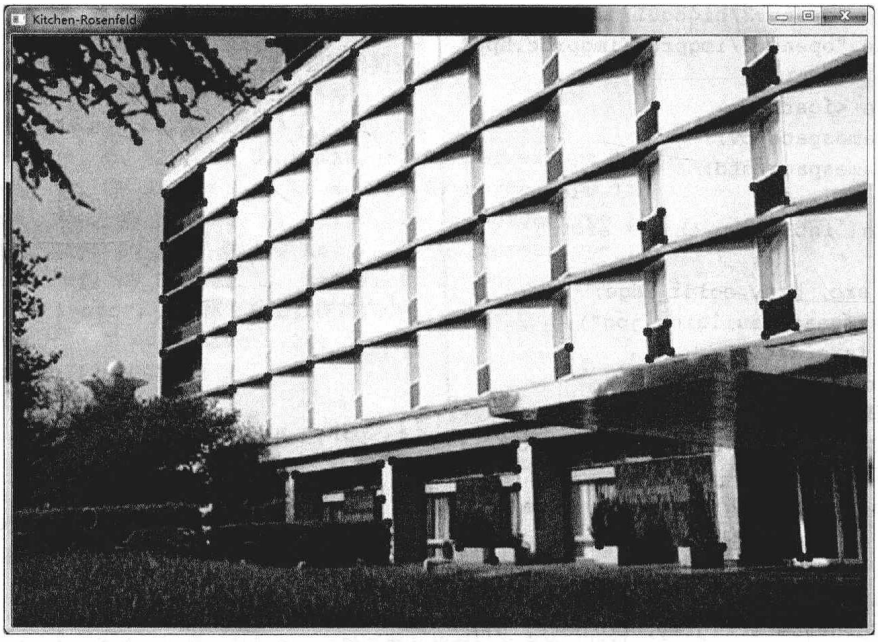
1.3 应用实例

由于 preCornerDetect 函数并没有完成非极大值抑制，因此这一步需要在应用程序中来实现，另外 Kitchen-Rosenfeld 角点检测方法对噪声比较敏感，所以还需要对输入图像进行平滑滤波。

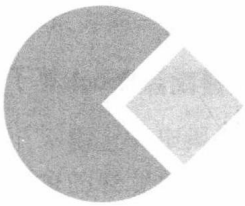
```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
using namespace cv;
using namespace std;

int main( int argc, char** argv )
{
    Mat src, gray,color_edge;
    src=imread("building.jpg");
    if( !src.data )
        return -1;
    //把输入的彩色图像转换成灰度图像
    cvtColor( src, gray, CV_BGR2GRAY );
    //高斯平滑滤波
    GaussianBlur( gray, gray, Size(9, 9), 2, 2 );

    Mat corners, dilated_corners;
    //Kitchen-Rosenfeld 角点检测，得到 corners 变量
    preCornerDetect(gray, corners, 5);
    //使用 3×3 的结构元素进行数学形态学中的膨胀处理，即在 3×3 的邻域内找最大值，结果存入
    //dilated_corners 变量内
    dilate(corners, dilated_corners, Mat());
    //遍历图像的所有像素
    for( int j = 0; j < src.rows ; j++ )
    {
        //每行的首地址指针
        const float* tmp_data = (const float*)dilated_corners.ptr(j);
        const float* corners_data = (const float*)corners.ptr(j);
        for( int i = 0; i < src.cols; i++ )
        {
            //非极大值抑制，并且要满足阈值条件，阈值设为 0.037，膨胀处理的结果如果等于角点，则说明该角点
            //是在 3×3 的邻域内的最大值
            if( tmp_data[i]>0.037 && corners_data[i]==tmp_data[i] )
            {
                //在角点处画一个红色的圆
```

```
                circle( src, Point( i, j ), 5, Scalar(0,0,255), -1, 8, 0 );
            }
        }
    }

    namedWindow("Kitchen-Rosenfeld", CV_WINDOW_AUTOSIZE );
    imshow("Kitchen-Rosenfeld", src );

    waitKey(0);
    return 0;
}
```

图 1-1 所示为得到的角点检测图像。



▲图 1-1　Kitchen-Rosenfeld 角点检测结果



第2章 Canny边缘检测

2.1 理论分析

Canny边缘检测方法是由Canny于1986年提出的一种被公认为效果较好的边缘检测方法。

一个好的边缘检测方法应该满足3项指标：第一项也是最重要的一项为低失误率，即不能漏检也不能错检；第二项为高的位置精度，即标定的边缘像素点与真正的边缘中心之间距离应该为最小；第三项为每个边缘应该有唯一的响应，即得到单像素宽度的边缘。为此，Canny提出了判定边缘检测算子的3个准则：信噪比准则，定位精度准则，单边缘响应准则。因此Canny的贡献不仅是提出了一种边缘检测算子，而且还给出了衡量边缘检测算子好坏的准则。

下面我们就重点介绍Canny算子的实现过程。

该方法共分为4个步骤：平滑处理、梯度检测、非极大值抑制和滞后阈值处理。以下是这4个步骤的详细讲解。

1. 平滑处理

所有的边缘都极易受到噪声的干扰，为了防止因噪声所引起的错误的检测结果，有必要应用平滑滤波的方法滤除噪声。高斯滤波方法是最常用的滤波方法，二维图像应用二维高斯函数，它的定义为：

$$G(u,v)=\frac{1}{2\pi\sigma^2}\mathrm{e}^{-\frac{u^2+v^2}{2\sigma^2}} \tag{2-1}$$

式中，σ表示高斯函数的标准差。只要把输入图像与二维高斯函数进行卷积，即可得到平滑处理后的图像。考虑到数字图像为离散化的形式，我们往往把高斯函数转换为离散化的高斯内核模板的形式，如标准差为1.4的模板尺寸为5×5的归一化高斯内核模板为：

$$K=\frac{1}{159}\times\begin{bmatrix}2&4&5&4&2\\4&9&12&9&4\\5&12&15&12&5\\4&9&12&9&4\\2&4&5&4&2\end{bmatrix} \tag{2-2}$$

2. 梯度检测

梯度是图像灰度值变化剧烈的地方，它可以通过 Roberts 算子、Prewitt 算子、Sobel 算子等最简单的模板检测方法得到。常用的是 Sobel 算子，它是由两个模板组成：

$$S_{GX}=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix},\quad S_{GY}=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \tag{2-3}$$

把这两个模板分别与图像进行卷积运算，则分别得到水平方向的梯度 G_x 和垂直方向的梯度 G_y。最终的梯度幅值 G 往往是由欧几里得距离（L2 范数）求得：

$$G=\sqrt{G_x^2+G_y^2} \tag{2-4}$$

但有时为了简化，梯度值也可由曼哈顿距离（L1 范数）得到：

$$G=|G_x|+|G_y| \tag{2-5}$$

而梯度幅角 θ 为：

$$\theta=\arctan\left(\frac{|G_y|}{|G_x|}\right) \tag{2-6}$$

由式（2-6）得到的角度可以是任意值，但这里我们需要把梯度幅角四舍五入到代表水平方向、垂直方向和 2 个对角线方向的 4 个方向上，即 0°、45°、90° 和 135°。例如，梯度幅角为–22.5°～22.5° 时，将被统一设置为 0°。

3. 非极大值抑制

这一步骤的目的是使边缘细化。由上一步得到边缘图像十分模糊，不符合好的边缘检测中的第三个指标，而非极大值抑制可以抑制那些局部不是梯度幅值最大值的边缘，而保留下来的具有局部最大值的像素点正是灰度值变化最剧烈的地方。这里的局部最大值是由在 3×3 的邻域内的梯度方向上比较梯度值得到的。例如：

（1）当梯度方向为 0° 时，图像的边缘是南-北方向，则在 3×3 的邻域内，当前像素与其左右两侧像素的梯度值进行比较，如果当前像素的梯度幅值最大，则保留，否则剔除；

（2）当梯度方向为 90° 时，图像的边缘是东-西方向，则在 3×3 的邻域内，当前像素与其上下两侧像素的梯度值进行比较，如果当前像素的梯度幅值最大，则保留，否则剔除；

（3）当梯度方向为 135° 时，图像的边缘是东北-西南方向，则在 3×3 的邻域内，当前像素与其左上角和右下角像素的梯度值进行比较，如果当前像素的梯度幅值最大，则保留，否则剔除；

（4）当梯度方向为 45° 时，图像的边缘是东南-西北方向，则在 3×3 的邻域内，当前像素与其右上角和左下角像素的梯度值进行比较，如果当前像素的梯度幅值最大，则保留，否则剔除。

还需要说明的是，梯度方向的符号与非极大值抑制的结果无关，即无论是东-西方向还是

西-东方向，两者是一样的。

4. 滞后阈值处理

由上一步得到的边缘仍有一小部分由于噪声或者颜色变化的影响而不是真正的边缘，这种现象的表现形式是尽管这些边缘的梯度幅值是局部最大值，但与其他边缘比，它们的梯度幅值很小，也就是绝对梯度幅值很小。处理它们也很简单，采用阈值法即可。但 Canny 采用的是双阈值的方法，即设置高、低两个阈值，当梯度幅值大于高阈值时，该边缘为强边缘，当梯度幅值小于低阈值时，该边缘需要被剔除，当梯度值介于高、低阈值之间时，该边缘为弱边缘。

强边缘毫无疑问是需要被保留下来的，而弱边缘则需要采用边缘跟踪的方法来判断其是否为真正的边缘。在弱边缘的 3×3 的领域内，如果有强边缘，则说明该弱边缘是属于这个强边缘的，所以需要被保留，否则被剔除掉。

2.2 源码解析

Canny 函数的原型为：

```
void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int
apertureSize=3, bool L2gradient=false )
```

image 表示输入图像。

edges 表示输出边缘图像。

threshold1 和 threshold2 表示滞后阈值法中所需要的高、低两个阈值。

apertureSize 表示孔径尺寸，即 Sobel 算子的尺寸大小，OpenCV 是采用 Sobel 算子来计算图像的梯度的，该值默认为 3。

L2gradient 表示在计算梯度幅值时是用式（2-4）的 L2 范数还是用式（2-5）的 L1 范数，该值默认为 false，即采用 L1 范数。

Canny 函数在 sources/modules/imgproc/src/canny.cpp 文件内被定义，它的源码为：

```
void cv::Canny( InputArray _src, OutputArray _dst,
                double low_thresh, double high_thresh,
                int aperture_size, bool L2gradient )
{
    Mat src = _src.getMat();                    //得到输入图像矩阵
    CV_Assert( src.depth() == CV_8U );          //确保输入图像是 8 位二进制的数据格式

    _dst.create(src.size(), CV_8U);             //创建输出边缘图像，尺寸大小与输入图像一致
    Mat dst = _dst.getMat();                    //得到输出边缘图像矩阵
    //调整 aperture_size 数值，满足兼容性
    if (!L2gradient && (aperture_size & CV_CANNY_L2_GRADIENT) == CV_CANNY_L2_GRADIENT)
    {
        //backward compatibility
        aperture_size &= ~CV_CANNY_L2_GRADIENT;
        L2gradient = true;
    }
```

```
    //确保孔径尺寸的大小在 3～7 之间
    if ((aperture_size & 1) == 0||(aperture_size != -1 && (aperture_size < 3 || aperture_size > 7)))
        CV_Error(CV_StsBadFlag, "");
    //确保双阈值中的高阈值大于低阈值，如果不是，则两者交换
    if (low_thresh > high_thresh)
        std::swap(low_thresh, high_thresh);

#ifdef HAVE_TEGRA_OPTIMIZATION
    if (tegra::canny(src, dst, low_thresh, high_thresh, aperture_size, L2gradient))
        return;
#endif

#ifdef USE_IPP_CANNY
    if( aperture_size == 3 && !L2gradient &&
        ippCanny(src, dst, (float)low_thresh, (float)high_thresh) )
        return;
#endif
    //得到输入图像的通道数，可以看出，Canny 函数能够直接处理彩色图像
    const int cn = src.channels();
    //定义两个矩阵 dx 和 dy，分别用于保存图像的水平方向梯度和垂直方向梯度
    Mat dx(src.rows, src.cols, CV_16SC(cn));
    Mat dy(src.rows, src.cols, CV_16SC(cn));
    //调用 Sobel 函数，分别用 Sobel 算子计算水平方向梯度 dx 和垂直方向梯度 dy
    Sobel(src, dx, CV_16S, 1, 0, aperture_size, 1, 0, cv::BORDER_REPLICATE);
    Sobel(src, dy, CV_16S, 0, 1, aperture_size, 1, 0, cv::BORDER_REPLICATE);
    //如果采用 L2 范数的方法计算梯度幅值，则两个阈值要做平方处理，这样求 L2 范数的时候就不用再开根号了
    if (L2gradient)
    {
        //确保高、低阈值的数值不能太大
        low_thresh = std::min(32767.0, low_thresh);
        high_thresh = std::min(32767.0, high_thresh);

        if (low_thresh > 0) low_thresh *= low_thresh;          //平方处理
        if (high_thresh > 0) high_thresh *= high_thresh;       //平方处理
    }
    //对两个阈值进行向下取整处理
    int low = cvFloor(low_thresh);
    int high = cvFloor(high_thresh);

    //定义步长
    ptrdiff_t mapstep = src.cols + 2;
    //开辟一段内存空间，它包括 mag_buf[3]和 map
    AutoBuffer<uchar> buffer((src.cols+2)*(src.rows+2) + cn * mapstep * 3 * sizeof(int));
    //定义 3 个指针数组，用于存储连续 3 行的梯度值
    //mag_buf[1]存放当前行的梯度幅值，mag_buf[0]存放前一行的梯度幅值，mag_buf[2]
    //存放后一行的梯度幅值，它们的长度都为 mapstep 乘以 cn
    int* mag_buf[3];
    mag_buf[0] = (int*)(uchar*)buffer;
    mag_buf[1] = mag_buf[0] + mapstep*cn;
    mag_buf[2] = mag_buf[1] + mapstep*cn;
```