**经典边缘检测算子比较**

**一 各种经典边缘检测算子原理简介**

图像的边缘对人的视觉具有重要的意义，一般而言，当人们看一个有边缘的物体时，首先感觉到的便是边缘。灰度或结构等信息的突变处称为边缘。边缘是一个区域的结束，也是另一个区域的开始，利用该特征可以分割图像。需要指出的是，检测出的边缘并不等同于实际目标的真实边缘。由于图像数据时二维的，而实际物体是三维的，从三维到二维的投影必然会造成信息的丢失，再加上成像过程中的光照不均和噪声等因素的影响，使得有边缘的地方不一定能被检测出来，而检测出的边缘也不一定代表实际边缘。图像的边缘有方向和幅度两个属性，沿边缘方向像素变化平缓，垂直于边缘方向像素变化剧烈。边缘上的这种变化可以用微分算子检测出来，通常用一阶或两阶导数来检测边缘，如下图所以。不同的是一阶导数认为最大值对应边缘位置，而二阶导数则以过零点对应边缘位置。

（a）图像灰度变化 （b）一阶导数 （c）二阶导数

基于一阶导数的边缘检测算子包括Roberts算子、Sobel算子、Prewitt算子等，在算法实现过程中，通过（Roberts算子）或者模板作为核与图像中的每个像素点做卷积和运算，然后选取合适的阈值以提取边缘。拉普拉斯边缘检测算子是基于二阶导数的边缘检测算子，该算子对噪声敏感。一种改进方式是先对图像进行平滑处理，然后再应用二阶导数的边缘检测算子，其代表是LOG算子。前边介绍的边缘检测算子法是基于微分方法的，其依据是图像的边缘对应一阶导数的极大值点和二阶导数的过零点。Canny算子是另外一类边缘检测算子，它不是通过微分算子检测边缘，而是在满足一定约束条件下推导出的边缘检测最优化算子。

**1 Roberts（罗伯特）边缘检测算子**

景物的边缘总是以图像中强度的突变形式出现的，所以景物边缘包含着大量的信息。由于景物的边缘具有十分复杂的形态，因此，最常用的边缘检测方法是所谓的“梯度检测法”。

设是图像灰度分布函数；是图像边缘的梯度值；是梯度的方向。则有  （1）

（n=1,2,...）

 （2）

式（1）与式（2）可以得到图像在（x,y）点处的梯度大小和梯度方向。

将式（1）改写为：

 （3）

称为Roberts边缘检测算子。式中对等的平方根运算使该处理类似于人类视觉系统的发生过程。事实上Roberts边缘检测算子是一种利用局部差分方法寻找边缘的算子，Robert梯度算子所采用的是对角方向相邻两像素值之差，所以用差分代替一阶偏导，算子形式可表示如下：

 （4）

上述算子对应的两个模板如图（A）所示。实际应用中，图像中的每个像素点都用这两个模板进行卷积运算，为避免出现负值，在边缘检测时常提取其绝对值。

|  |  |
| --- | --- |
| 1 | 0 |
| 0 | -1 |

|  |  |
| --- | --- |
| 0 | 1 |
| -1 | 0 |

（a） （b）

图（A）Robert算子模板

**2 Sobel（索贝尔）边缘检测算子**

该算子是由两个卷积核与对原图像进行卷积运算而得到的。其数学表达式为：

（5）

实际上Sobel边缘算子所采用的算法是先进行加权平均，然后进行微分运算，我们可以用差分代替一阶偏导，算子的计算方法如下：

 （6）

Sobel算子垂直方向和水平方向的模板如图（B）所示，前者可以检测出图像中的水平方向的边缘，后者则可以检测图像中垂直方向的边缘。实际应用中，图像中的每一个像素点都用这两个卷积核进行卷积运算，取其最大值作为输出。运算结果是一幅体现边缘幅度的图像。

|  |  |  |
| --- | --- | --- |
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -3 | 0 | 1 |

（a） （b）

图（B）Sobel算子模板

**3 Prewitt（普瑞维特）边缘检测算子**

Prewitt边缘检测算子就是一种利用局部差分平均方法寻找边缘的算子，它体现了三对像素点像素值之差的平均概念，因为平均能减少或消除噪声，为此我们可以先求平均，再求差分，即利用所谓的平均差分来求梯度。用差分代替一阶偏导可得算子形式如下：

 （7）

Prewitt边缘检测算子的两个模板如图（C）所示，它的使用方法同Sobel算子一样，图像中的每个点都用这两个核进行卷积，取得最大值作为输出。Prewitt算子也产生一幅边缘图像。

|  |  |  |
| --- | --- | --- |
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

（a） （b）

图（C）Prewitt算子模板

**4 Laplacian（拉普拉斯）边缘检测算子**

对于阶跃状边缘，其二阶导数在边缘点出现过零交叉，即边缘点两旁的二阶导数取异号，据此可以通过二阶导数来检测边缘点。拉普拉斯边缘检测算子正是对二维函数进行二阶导数运算的标量算子，它的定义是：

 （8）

用差分代替二阶偏导时，与前述三个一阶导数算子不同，拉普拉斯算子的形式可表示如下：



（9）

拉普拉斯边缘检测算子的模板如图（D）所示，模板的基本特征是中心位置的系数为正，其余位置的系数为负，且模板的系数之和为零。它的使用方法是用图中的两个点阵之一作为卷积核，与原图像进行卷积运算即可。拉普拉斯算子又是一个线性的移不变算子，它的传递函数在频域空间的原点为零，因此，一个经拉普拉斯滤波过的图像具有零平均灰度。拉普拉斯检测模板的特点是各向同性，对孤立点及线端的检测效果好，但边缘方向信息丢失，对噪声敏感，整体检测效果不如梯度算子。因此，它很少直接用于边缘检测。但注意到与Sobel算子相比，对图像进行处理时，拉普拉斯算子能使噪声成分得到加强，对噪声更敏感。

|  |  |  |
| --- | --- | --- |
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

（a） （b）

图（D）Laplace算子模板

**5 Marr-Hildreth（马尔）边缘检测算子**

实际应用中，由于噪声的影响，对噪声敏感的边缘检测点检测算法（如拉普拉斯算子法）可能会把噪声当边缘点检测出来，而真正的边缘点会被噪声淹没而未检测出。为此Marr和Hildreth提出了马尔算子，因为是基于高斯算子和拉普拉斯算子的，所以也称高斯-拉普拉斯（Laplacian of Gaussian,LoG）边缘检测算子，简称LoG算子。该方法是先采用高斯算子对原图像进行平滑又降低了噪声，孤立的噪声点和较小的结构组织将被滤除由于平滑会导致边缘的延展，因此在边缘检测时仅考虑那些具有局部最大值的点为边缘点，这一点可以用拉普拉斯算子将边缘点转换成零交叉点，然后通过零交叉点的检测来实现边缘检测。所谓零交叉点就是：如果一个像素处的值小于一，而此像素8-连通的各个像素都是大于 （是一个正数），那么这个像素就是零交叉点。这样还能克服拉普拉斯算子对噪声敏感的缺点，减少了噪声的影响。二维高斯函数为

 （10）

则连续函数的LoG边缘检测算子定义为





 （11）

 （12）

其中是标准差。算子是一个轴对称函数，其横截面如图（E）所示。由于它相当的平滑，能减少噪声的影响，所以当边缘模糊或噪声较大时，利用检测过零点能提供较可靠的边缘位置。

H（x,y）

r

0

图（E）H（x,y）的截面图

LoG算子在（x,y）空间具有以原点为中心旋转的对称性，LoG滤波器具有如下三个显著特点：

* 该滤波器中的高斯函数部分对图像具有平滑作用，可有效地消除尺度远小于高斯分布因子的噪声信号。
* 高斯函数在空域和频域内都具有平滑作用。
* 该滤波器采用拉普拉斯算子可以减少计算量。

马尔算子用到的卷积模板一般比较大（典型半径为8-32个像素），不过这些模板可以分解为一维卷积来快速计算。常用的LoG算子是模板，如图（F）。与其他边缘检测算子一样，LoG算子也是先对边缘做出假设，然后再这个假设下寻找边缘像素。但LoG算子对边缘的假设条件最少，因此它的应用范围更广。另外，其他边缘检测算子检测得到的边缘时不连续的，不规则的，还需要连接这些边缘，而LoG算子的结果没有这个缺点。对于LoG算子边缘检测的结果可以通过高斯函数标准偏差来进行调整。即值越大，噪声滤波效果越好，但同时也丢失了重要的边缘信息，影响了边缘检测的性能；值越小，又有可能平滑不完全而留有太多的噪声。因此，在不知道物体尺度和位置的情况下，很难准确确定滤波器的值。一般来说，使用大值的滤波器产生鲁棒边缘，小的值的滤波器产生精确定位的边缘，两者结合，能够检测出图像的最佳边缘。数学上以证明，马尔算子是按零交叉检测阶跃状边缘的最佳算子。但在实际图像中要注意到，高斯滤波的零交叉点不一定全部是边缘点，还需要进一步对其真伪进行检验。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -2 | -4 | -4 | -4 | -2 |
| -4 | 0 | 8 | 0 | -4 |
| -4 | 8 | 24 | 8 | -4 |
| -4 | 0 | 8 | 0 | -4 |
| -2 | -4 | -4 | -4 | -2 |

图（F）LoG算子是模板

**6 Kirsch（凯西）边缘检测算子**

Kirsch边缘检测算子由8个卷积核组成，图像中的每个像素点都用这8个核进行卷积运算，即须求出8个方向的平均差分。像素与不同卷积核卷积运算的结果给出了相应特定边缘方向的响应。从所有方向相应中找出一个最大值，就给出了经过该点的边缘幅度图像的输出值。使输出最大的卷积核的序号既是边缘方向的编码。该算子还可以较好地抑制边缘检测的噪声。假设原始图像的子图像如图（G）所示。

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

图（G）子图像示意图

则边缘的梯度大小为

 （13）

式中

 （14）

式（14）中的下标超过7就用8去除并取余数。k=0,1,...,7实际上就是使用了8个模板，

8个方向的模板如图（H）所示

|  |  |  |
| --- | --- | --- |
| 5 | 5 | 5 |
| 3 | 0 | -3 |
| -3 | -3 | 3 |

|  |  |  |
| --- | --- | --- |
| 3 | 5 | 5 |
| -3 | 0 | 5 |
| -3 | -3 | -3 |

|  |  |  |
| --- | --- | --- |
| -3 | 3 | 5 |
| -3 | 0 | 5 |
| -3 | -3 | 5 |

|  |  |  |
| --- | --- | --- |
| -3 | -3 | -3 |
| -3 | 0 | 5 |
| -3 | 5 | 5 |

|  |  |  |
| --- | --- | --- |
| -3 | -3 | -3 |
| -3 | 0 | 3 |
| 5 | 5 | 5 |

|  |  |  |
| --- | --- | --- |
| -3 | -3 | -3 |
| 5 | 0 | 3 |
| 5 | 5 | 3 |

|  |  |  |
| --- | --- | --- |
| 5 | -3 | -3 |
| 5 | 0 | 3 |
| 5 | 5 | 3 |

|  |  |  |
| --- | --- | --- |
| 5 | 5 | 3 |
| 5 | 0 | 3 |
| -3 | -3 | -3 |

图（H）Kirsch算子的8方向模板

从上面的8个方向模板可以看出，每两个卷积核（模板）之间的夹角为45度。

对于图像T，模块为（k=1,2,...,8），则边缘强度在点（x,y）处为：

 {k=1,2,...,8} （15）

其中，以表示点乘运算。如果取最大值的模作为边缘强度，同时用考虑最大值符号的方法来确定相对应的边缘方向，则考虑到各模板的对称性，只需要前四个模板即可。对于Kirsch算子也有几种不同的形式，如图（I）所示8方向的模板就是一种比较常用的方式。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 1 |
| -1 | 0 | 1 | 1 | 1 |
| -1 | -1 | 0 | 1 | 1 |
| -1 | -1 | -1 | 0 | 1 |
| -1 | -1 | -1 | -1 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -1 | -1 | 0 | 1 | 1 |
| -1 | -1 | 0 | 1 | 1 |
| -1 | -1 | 0 | 1 | 1 |
| -1 | -1 | 0 | 1 | 1 |
| -1 | -1 | 0 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | -1 |
| 1 | 1 | 0 | -1 | -1 |
| 1 | 0 | -1 | -1 | -1 |
| 0 | -1 | -1 | -1 | -1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

图（I）8方向Kirsch算子的前4个

如果对图像进行二值化分析，可以发现图像中非边缘点往往低于阈值，而大多数边缘点则往往高于该阈值。Kirsch方向算子正是利用了这种现象，在进行边缘的检测时，首先用一个中偏低的灰度阈值对梯度图像进行二值化，从而检测出物体与背景，物体与背景被处于阈值之上的边界分开。随着阈值的不断提高，也促使物体和背景的一同增长，当达到接触而又不合并的临界状态时，就可以用接触点来定义边缘。

确定了处于边缘的像素点之后，理想的情形是这些边缘点都是连续的，把各个点连接起来就构成了区域的边缘，各个区域也就分割开来了。但是，在很多情况下，边缘点或边缘线端之间会有间断。所以通常情况下，如果边缘点很稀疏，那么就要用分段线性或高阶样条曲线来拟合这些点，从而形成一条可以用来分割区域的适用边界。总之，虽然Kirsch方向算子的计算开销比较大，但却可以产生出最大的梯度边界。与之类似的另一种比较常用的方向算子是由12个卷积核组成的Nevitia方向算子，其每两个卷积核之间的夹角为30度，该算子充分利用了各位置的权值来调整边缘的方向。

**7 canny（凯尼）边缘检测算子**

前面介绍的边缘检测算子都是局域窗口梯度算子，由于它们对噪声敏感，所以在处理实际图像时效果并不是十分理想。根据边缘检测的有效性和定位的可靠性，Canny研究了最优边缘检测器所需的特性，给出了评价边缘检测性能优劣的三个指标：

1. 高的准确性，在检测的结果里应尽量多的包含真正的边缘，而尽量少的包含假边缘。
2. 高的精确度，检测到的边缘应该在真正的边界上。
3. 单像素宽，要有很高的选择性，对每个边缘有唯一的响应。

针对这三个指标，Canny提出了用于边缘检测的一阶微分滤波器的三个最优化标准则，即最大信噪比准则、最优过零点定位准则和单边缘响应准则。具体如下：

（a）信噪比准则

 （16）

式中，为边缘函数；为带宽为W的低通滤波器的脉冲响应；是高斯噪声的均方差。

（b）定位精确度准则

L为边缘的定位精度，定义如下：

 （17）

式中，和为和的一阶导数；L是对边缘定位精确程度的度量，L越大定位精度越高。

（c）单边缘响应准则

要保证对但边缘只有一个响应，检测算子的脉冲响应导数的零交叉点平均距离应该满足：  （18）

式中，是的二阶导数；是进行边缘检测后的图像。

这三个准则是对前述边缘检测指标的定量描述。抑制噪声和边缘精确定位是无法同时得到满足的，即边缘检测算法通过图像平滑算子去除噪声，势必增加边缘定位的不确定性；反之，若提高边缘检测算子对边缘的敏感性，同时也提高了对噪声的敏感性。因此，在实际应用中只能寄希望于再抑制噪声和提高边缘定位精度之间实现一个合理的折衷。

值得庆幸的是，有一个线性算子可以在抵抗噪声与边缘检测之间获得一个最佳的折衷，这个算子就是高斯函数的一阶导数。高斯函数与原图的卷积达到了抵抗噪声的作用，而求导数，则是检测景物边缘的手段。

对于阶跃形的边缘，Canny推导出的最优边缘检测器的形状与高斯函数的一阶导数类似，因此Canny边缘检测器就是高斯函数的一阶导数构成的。我们知道，高斯函数是圆对称的，因此，Canny算子在边缘方向上是对称的，在垂直于边缘的方向上是反对称的。这就意味着该算子对最急剧变化方向上的边缘特别敏感，但在沿边缘方向上是不敏感的。

设二维高斯函数为  （19）

其中，是高斯函数的分布参数，可用以控制对图像的平滑程度。

最优阶跃边缘检测算子是以卷积为基础的，边缘强度为，而边缘方向为。

从高斯函数的定义可知，该函数是无限拖尾的，在实际应用中，一般情况下是将原始模板截断到有限尺寸N。实验表明，当时，能够获得较好的边缘检测结果。下面给出Canny算子的具体实现。

利用高斯函数的可分性，将的两个滤波卷积模板分解为两个一维的行列滤波器：

 （20）

 （21）

其中 ，



可见， ，k为常数。

然后把这两个模板分别与进行卷积，得到

 （22）

令  （23）

则反映边缘强度，为垂直于边缘的方向。

根据Canny的定义，中心边缘点为算子与图像的卷积在边缘梯度方向上的区域中的最大值。这样，就可以在每一点的梯度方向上判断此点强度是否为其领域的最大值来确定该点是否为边缘点。当一个像素满足以下三个条件时，则被认为是图像的边缘点。

1）该点的边缘强度大于沿该点梯度方向的两个相邻像素点的边缘强度；

2）与该点梯度方向上相邻两点方向差小于45°；

3）以该点为中心的领域中的边缘强度极大值小于某个阈值。

此外，如果1）和2）同时被满足，那么在梯度方向上的相邻像素就从候选边缘点中取消，条件3）相当于区域梯度最大值组成的阈值图像与边缘点进行匹配，这一过程消除了许多虚假的边缘点。

Canny边缘检测算子步骤如下：

Step1：用高斯滤波器对图像进行滤波消噪；

Step2：用一阶偏导的有限差分来计算梯度的幅值和方向；

Step3：对梯度幅值进行非极大值抑制；

Step4：用双阈值算法检测和连接边缘。

**二 各种经典边缘检测算子性能比较**

由上述边缘算子产生的边缘图像看来很相似。它们看起来像一个绘画者从图片中做出的线条画。Roberts算子是算子，对具有陡峭的低噪声图像响应最好。Sobel算子， Prewitt算子，Kirsch算子都是算子，对灰度渐变和噪声较多的图像处理得较好。使用两个掩模板组成边缘检测器时，通常取较大的幅度作为输出值。这使得它们对边缘的走向有些敏感。取它们的平方和的开方可以获得性能更一致的全方位响应。这与真实的梯度值更接近。值得注意的是的Sobel算子和 Prewitt边缘算子可扩展成八个方向，并且可以像使用Kirsch算子一样获得边缘方向图。

**1 MATLAB程序仿真**

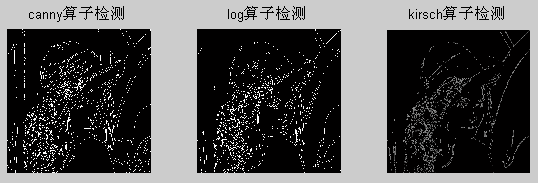
为了分析上述几种算子的效果，我们用MATLAB对lenna原始图像以及分别加入高斯噪声和椒盐噪声后的lenna图像进行检测。仿真实验结果如下（程序见附录）：



Lenna原始图像

（一）各算子对lenna原始图像的处理效果如下所示：



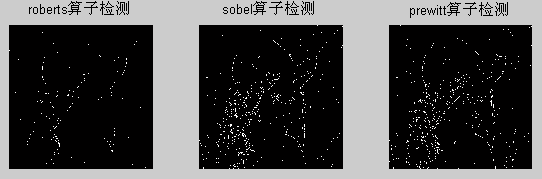


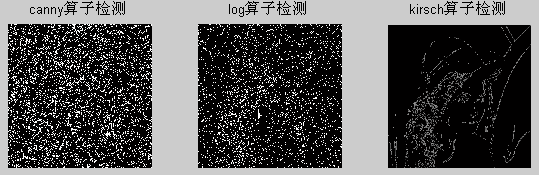
（二）对lenna原始图像加入高斯噪声：



加入高斯噪声的lenna图像

各算子对加入高斯噪声的lenna图像的处理效果如下所示：



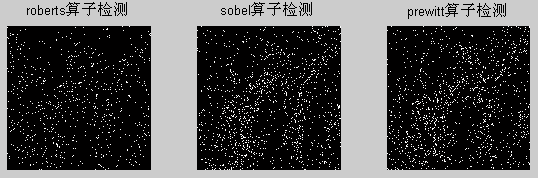


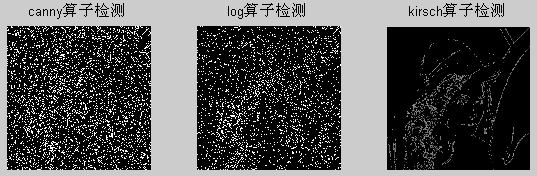
（三）对lenna原始图像加入椒盐噪声：



加入椒盐噪声的lenna图像

各算子对加入椒盐噪声的lenna图像的处理效果如下所示：





**2实验效果比较**

由于Roberts算子是利用图像的两个对角线的相邻像素之差进行梯度幅值的检测，所以求得的是在差分点处梯度幅值的近似值，并且检测水平和垂直方向边缘的性能好于斜线方向的边缘，检测精度比较高，但容易丢失一部分边缘，同时由于没经过图像平滑计算，因此不能抑制噪声，但该算子对具有陡峭的低噪声图像响应最好。

Prewitt算子和Sobel算子都是对图像进行差分和滤波运算，仅在平滑部分的权值选择上有些差异，因此两者均对噪声具有一定的抑制能力，但这种抗噪能力是通过像素平均来实现的，所以图像产生了一定的模糊，而且还会检测出一些伪边缘，所以检测精度比较低，该算子比较适合用于图像边缘灰度值比较尖锐且图像噪声比较小的情况。

LoG算子首先通过高斯函数对图像进行平滑处理，因此对噪声的抑制作用比较明显，但同时也可能将原有的边缘也平滑了，造成某些边缘无法检测到，比外高斯分布因子的选择对图像边缘检测效果有较大的影响，越大，检测到的图像细节越丰富，但抗噪能力下降，从而出现伪边缘，反之则抗噪能力提高，但边缘精度下降，易丢失许多真边缘，因此，对于不同图像应选择不同参数。

Canny算子也采用高斯函数对图像进行平滑处理，因此具有较强的去噪能力，但同样存在容易平滑掉一些边缘信息，其后所采用的一阶微分算子的方向性较LoG算子要好，因此边缘定位精度较高。该算子与其它边缘检测算子的不同之处在于，它使用2种不同的阈值分别检测强边缘和弱边缘，并且仅当弱边缘相连时才将弱边缘包含在输出图像中，因此这种方法较其它方法而言不容易被噪声“填充”更容易检测出真正的弱边缘。通过对lenna图的仿真实验结果可以看出，该算子在上述几种边缘检测算子当中效果最好。边缘定位准确，连续性较好，虚假边缘少且边缘均具有单像素宽度。

表一 各种经典边缘检测算子的优缺点比较

|  |  |
| --- | --- |
| 算子 | 优缺点比较 |
| Roberts | 对具有陡峭的低噪声的图像处理效果较好，但利用Roberts算子提取边缘的结果是边缘比较粗，因此边缘定位不是很准确。 |
| Sobel | 对灰度渐变和噪声较多的图像处理效果比较好，Sobel算子对边缘定位比较准确。 |
| Kirsch | 对灰度渐变和噪声较多的图像处理效果较好。 |
| Prewitt | 对灰度渐变和噪声较多的图像处理效果较好。 |
| Laplacian | 对图像中的阶跃性边缘点定位准确，对噪声非常敏感，丢失一部分边缘的方向信息，造成一些不连续的检测边缘。 |
| LoG | LoG算子经常出现双边缘像素边界，而且该检测方法对噪声比较敏感，所以很少用LoG算子检测边缘，而是用来判断边缘像素是位于图像的明区还是暗区。 |
| Canny | 此方法不容易受噪声的干扰，能够检测到真正的弱边缘。在edge函数中，最有效的边缘检测方法是Canny方法。该方法的优点在于使用两种不同的阈值分别检测强边缘和弱边缘，并且仅当弱边缘与强边缘相连时，才将弱边缘包含在输出图像中。因此，这种方法不容易被噪声“填充”，跟容易检测出真正的弱边缘。 |

**三 结论**

目前，边缘检测技术在很多领域中都得到了广泛的应用。如对医学图像的边缘特征提取、车牌识别、人脸检测技术等。本文讨论和比较了几种常用的经典边缘检测算子。可以看出它们都不是具有绝对优势的方法，在图像的抗噪声、图像边缘的定位情况、图像部分边缘是否可以检测出，以及算法运行的速度等方面，各自存在优缺点。所以对于图像的边缘检测，要根据不同的系统，针对不同的环境条件和要求，选择合适的算子。希望本文总结的各种算法的优缺点及适用范围，能为今后人们使用这些算子以及研究更好的边缘检测算子方法提供参考价值。

**附录**

**附录（1）**

各算子对lenna原始图像处理的仿真程序：

function jingdian

I=imread('lenna.jpg');

B1=edge(I,'roberts');

B2=edge(I,'sobel');

B3=edge(I,'prewitt');

B4=edge(I,'canny');

B5=edge(I,'log');

B6=DetectContour(I,0.3);

subplot(2,3,1);

imshow(B1);title('roberts算子检测');

subplot(2,3,2);

imshow(B2);title('sobel算子检测');

subplot(2,3,3);

imshow(B3);title('prewitt算子检测');

subplot(2,3,4);

imshow(B4);title('canny算子检测');

subplot(2,3,5)

imshow(B5);title('log算子检测');

subplot(2,3,6);

imshow(B6);title('kirsch算子检测');

**附录（2）**

各算子对加入高斯噪声lenna图像处理的仿真程序：

function jingdian

I=imread('lenna.jpg');

I1=imnoise(I,'gaussian');

B1=edge(I1,'roberts');

B2=edge(I1,'sobel');

B3=edge(I1,'prewitt');

B4=edge(I1,'canny');

B5=edge(I1,'log');

B6=DetectContour(I1,0.3);

subplot(2,3,1);

imshow(B1);title('roberts算子检测');

subplot(2,3,2);

imshow(B2);title('sobel算子检测');

subplot(2,3,3);

imshow(B3);title('prewitt算子检测');

subplot(2,3,4);

imshow(B4);title('canny算子检测');

subplot(2,3,5)

imshow(B5);title('log算子检测');

subplot(2,3,6);

imshow(B6);title('kirsch算子检测');

**附录（3）**

各算子对加入椒盐噪声lenna图像处理的仿真程序：

function jingdian

I=imread('lenna.jpg');

I2=imnoise(I,'salt & pepper');

B1=edge(I2,'roberts');

B2=edge(I2,'sobel');

B3=edge(I2,'prewitt');

B4=edge(I2,'canny');

B5=edge(I2,'log');

B6=DetectContour(I2,0.3);

subplot(2,3,1);

imshow(B1);title('roberts算子检测');

subplot(2,3,2);

imshow(B2);title('sobel算子检测');

subplot(2,3,3);

imshow(B3);title('prewitt算子检测');

subplot(2,3,4);

imshow(B4);title('canny算子检测');

subplot(2,3,5)

imshow(B5);title('log算子检测');

subplot(2,3,6);

imshow(B6);title('kirsch算子检测');

**附录（4）**

DetectContour函数：

function [contourImage]=DetectContour(I,k)

grayImage = imread('lenna.jpg');

[m n d]=size(grayImage);

work\_img(1:m,1:n)=0;

img=double(grayImage);

r(1:8)=0;

for i=2:m-1

for j=2:n-1

blk=img(i-1:i+1,j-1:j+1);

p0=blk(1,1);p1=blk(1,2);p2=blk(1,3);

p7=blk(2,1);p3=blk(2,3);

p6=blk(3,1);p5=blk(3,2);p4=blk(3,3);

r(1)=(5/8)\*(p0+p1+p2)-(3/8)\*(p3+p4+p5+p6+p7);

r(2)=r(1)+p7-p2; r(3)=r(2)+p6-p1; r(4)=r(3)+p5-p0;

r(5)=r(4)+p4-p7;r(6)=r(5)+p3-p6;r(7)=r(6)+p2-p5; r(8)=r(7)+p1-p4;

maxR=8\*k\*max(r);

if maxR>255

maxR=255;

end

if maxR<0

maxR=0;

end

work\_img(i,j)=255-maxR;

if work\_img(i,j)>150