1.Prewitt

prewitt算子对噪声有抑制作用，抑制噪声的原理是通过像素平均，但是像素平均相当于对图像的低通滤波，低通滤波会造成高频的信息丢失，从而使图像模糊，无论这种程度或大或小，这种操作后的结果是存在的，所以Prewitt算子对边缘的定位不如Roberts算子。

Prewitt 算子：

-1	-1	-1
0	0	0
1	1	1

垂直

-1	0	1
-1	0	1
-1	0	1

水平

0	1	1
-1	0	1
-1	-1	0

-1	-1	0
-1	0	1
0	1	1

对角线

https://blog.csdn.net/Chaolei3

2.sobel

sobel算子是prewitt算子的改进形式，改进之处在于sobel算子认为，邻域的像素对当前像素产生的影响不是等价的，所以距离不同的像素具有不同的权值，对算子结果产生的影响也不同。一般来说，距离越远，产生的影响越小。正因为Sobel算子对于像素的位置的影响做了**加权**，与Prewitt算子、Roberts算子相比因此效果更好。相比较Prewitt算子，Sobel模板能够较好的抑制（平滑）噪声。 sobel要比prewitt更能准确检测图像边缘。

http://www.cnblogs.com/ronny/p/3387575.html

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

该算子包含两组3x3的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。如果以A代表原始图像，Gx及Gy分别代表经横向及纵向边缘检测的图像灰度值，其公式如下：

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

具体计算如下：

Gx = (-1)*f(x-1, y-1) + 0*f(x,y-1) + 1*f(x+1,y-1)

+(-2)*f(x-1,y) + 0*f(x,y)+2*f(x+1,y)

+(-1)*f(x-1,y+1) + 0*f(x,y+1) + 1*f(x+1,y+1)

= [f(x+1,y-1)+2*f(x+1,y)+f(x+1,y+1)]-[f(x-1,y-1)+2*f(x-1,y)+f(x-1,y+1)]

Gy =1* f(x-1, y-1) + 2*f(x,y-1)+ 1*f(x+1,y-1)

+0*f(x-1,y) 0*f(x,y) + 0*f(x+1,y)

+(-1)*f(x-1,y+1) + (-2)*f(x,y+1) + (-1)*f(x+1, y+1)

= [f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)]-[f(x-1, y+1) + 2*f(x,y+1)+f(x+1,y+1)]

其中f(a,b), 表示图像(a,b)点的灰度值；

图像的每一个像素的横向及纵向灰度值通过以下公式结合，来计算该点灰度的大小：

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

通常，为了提高效率　使用不开平方的近似值：

$$|G| = |Gx| + |Gy|$$

如果梯度G大于某一阀值　则认为该点(x,y)为边缘点。

然后可用以下公式计算梯度方向：

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Sobel算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。对噪声具有平滑作用，提供较为精确的边缘方向信息，边缘定位精度不够高。当对精度要求不是很高时，是一种较为常用的边缘检测方法。

Sobel边缘检测

Soble边缘检测算法比较简，实际应用中效率比canny边缘检测效率要高，但是边缘不如Canny检测的准确，但是很多实际应用的场合，sobel边缘却是首选，尤其是对效率要求较高，而对细纹理不太关心的时候。

Soble边缘检测通常带有方向性，可以只检测竖直边缘或垂直边缘或都检测。

所以我们先定义两个梯度方向的系数：

```
kx=0;ky=1;% horizontal
kx=1;ky=0;% vertical
kx=1;ky=1;% both
```

然后我们来计算梯度图像，我们知道边缘点其实就是图像中灰度跳变剧烈的点，所以先计算梯度图像，然后将梯度图像中较亮的那一部分提取出来就是简单的边缘部分。

Sobel算子用了一个3*3的滤波器来对图像进行滤波从而得到梯度图像，这里面不再详细描述怎样进行滤波及它们的意义等。

竖起方向的滤波器：y_mask=op = [-1 -2 -1;0 0 0;1 2 1]/8;

水平方向的滤波器：op的转置：x_mask=op';

定义好滤波器后，我们就开始分别求垂直和竖起方向上的梯度图像。用滤波器与图像进行卷积即可：

```
bx = abs(filter2(x_mask,a));
by = abs(filter2(y_mask,a));
```

上面bx为水平方向上的梯度图像，by为垂直方向上的梯度图像。为了更清楚的说明算法过程，下面给出一张示例图像的梯度图像。

而最终的梯度图像为：b = kx*bx.*bx + ky*by.*by;当然这里如果定义了检测方向，就会得到对应上面的图像。

这里值得注意的是为了计算效率并没有对b开平方。**而涉及滤波等操作时对图像边界的处理是值得注意的一个地方**。我们这里应该将梯度图像的四周1像素点都设置为0。

得到梯度图像后，我们需要的是计算阈值，这是Sobel算法很核心的一部分，也是对效果影响较大的地方，同理讲到canny边缘检测时，用到的双阈值法也是canny算法的核心。

同样这里，我并不太多的介绍算法原理，相关文献可以直接百度。阈值也可以通过自己期待的效果进行自定义阈值，如果没有，则我们计算默认阈值。

```
scale = 4;
cutoff = scale*mean2(b);
thresh = sqrt(cutoff);
```

其中mean2函数是求图像所有点灰度的平均值。scale是一个系数。

有了阈值后，我们就可以地得到的梯度图像进行二值化。二值化过程，不做详细说明，遍历图像中的像素点，灰度大于阈值的置为白点，灰度小于阈值的则置为黑点。

其实很多介绍Soble算法的文章介绍到这里就结束了，印象中OpenCv的算法也是到此步为止。但是我们注意到上面的边缘图像，边缘较粗，如果我们在做边界跟踪或轮廓提取时，上面图像并不是我们期望的结果。

所以下面要介绍一个很重要的算法，用**非极大值抑制算法**对边缘进行1像素化。本人在开始的时候也一直以为这里用一个普通的细化算法就可以了，可是总得到到想要的结果，最后查找matlab早期版本的源码才找到方法，其实跟canny算法里原理差不多。

我们需要遍历刚才得到的梯度图像二值化结果b,对于b内的任意一点(i,j)，如果满足下面条件，则保持白点，否则置为黑点。条件简单的说即是：如果是竖直边缘，则它的梯度值要比左边和右边的点都大；如果是水平连续，则该点的梯度值要比上边和下边的都大。

bx(i,j)>kx*by(i,j) && b(i,j-1)<b(i,j) && b(i,j+1)<b(i,j)

或者

by(i,j)>ky*bx(i,j) && b(i-1,j)<b(i,j) &&b (i+1,j)<b(i,j)

经过这样的非极大值抑制我们就可以得到比较理想的边缘图像。

```
bool Sobel(const Mat& image,Mat& result,int TYPE)
 2 {
 3     if(image.channels()!=1)
 4         return false;
 5     // 系数设置
 6     int kx(0);
 7     int ky(0);
 8     if( TYPE==SOBEL_HORZ ){
 9         kx=0;ky=1;
10     }
11     else if( TYPE==SOBEL_VERT ){
12         kx=1;ky=0;
13     }
14     else if( TYPE==SOBEL_BOTH ){
15         kx=1;ky=1;
16     }
17     else
18         return false;
19 
20     // 设置mask
21     float mask[3][3]={{1,2,1},{0,0,0},{-1,-2,-1}};
22     Mat y_mask=Mat(3,3,CV_32F,mask)/8;
23     Mat x_mask=y_mask.t(); // 转置
24 
25     // 计算x方向和y方向上的滤波
26     Mat sobelX,sobelY;
27     filter2D(image,sobelX,CV_32F,x_mask);
28     filter2D(image,sobelY,CV_32F,y_mask);
29     sobelX=abs(sobelX);
30     sobelY=abs(sobelY);
31     // 梯度图
32     Mat gradient=kx*sobelX.mul(sobelX)+ky*sobelY.mul(sobelY);
33 
34     // 计算阈值
35     int scale=4;
36     double cutoff=scale*mean(gradient)[0];
37 
38     result.create(image.size(),image.type());
39     result.setTo(0);
40     for(int i=1;i<image.rows-1;i++)
41     {
```

```
42        float* sbxPtr=sobelX.ptr<float>(i);
43        float* sbyPtr=sobelY.ptr<float>(i);
44        float* prePtr=gradient.ptr<float>(i-1);
45        float* curPtr=gradient.ptr<float>(i);
46        float* lstPtr=gradient.ptr<float>(i+1);
47        uchar* rstPtr=result.ptr<uchar>(i);
48        // 阈值化和极大值抑制
49        for(int j=1;j<image.cols-1;j++)
50        {
51            if( curPtr[j]>cutoff && (
52                (sbxPtr[j]>kx*sbyPtr[j] && curPtr[j]>curPtr[j-1] && curPtr[j]>curPtr[j+1]) ||
53                (sbyPtr[j]>ky*sbxPtr[j] && curPtr[j]>prePtr[j] && curPtr[j]>lstPtr[j]) ))
54                rstPtr[j]=255;
55        }
56    }
57
58    return true;
59 }
```

3.canny

https://www.cnblogs.com/love6tao/p/5152020.html

Canny算子求边缘点具体算法步骤如下：

1. 用高斯滤波器平滑图像.

2. 用一阶偏导有限差分计算梯度幅值和方向

3. 对梯度幅值进行非极大值抑制

4. 用双阈值算法检测和连接边缘.

4.Roberts

Reboerts算子是一种利用局部差分来寻找边缘的算子，Roberts 梯度算子所采用的是对角方向相邻两像素值之差，算子形式如下：

Gx = f(i,j) - f(i-1,j-1)　　(2.3.3)

Gy = f(i-1,j) - f(i,j-1)　　(2.3.4)

|G(x,y)| = sprt(Gx^2-Gy^2)　　(2.3.5)

*Roberts*梯度算子对应的卷积模版为：

$$Gx=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\qquad Gy=\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}\qquad (2.3.6)$$

用以上两个卷积算子与图像运算后，可求出图像的梯度幅值 G (x,y)，然后选择适当的阈值 τ ，若 G (x,y)＞τ，则 (i ,j)为边缘点，否则，判断 (i ,j)为非边缘点。由此得到一个二值图像 { g (i,j)}，即边缘图像。

Roberts 算子采用的是用对角线方向上相邻两像素的差近似梯度幅值来检测边缘，它的定位精度高，对于水平和垂直方向的边缘，检测效果较好，而对于有一定

倾角的斜边缘，检测效果则不理想，存在着许多的漏检。另外，在含噪声的情况下，Roberts 算子不能有效的抑制噪声，容易产生一些伪边缘。因此，该算子适合于对低噪声且具有陡峭边缘的图像提取边缘。

Roberts算子采用对角线方向相邻两像素之差近似梯度幅值检测边缘。检测水平和垂直边缘的效果好于斜向边缘，定位精度高，对噪声敏感

5.Laplace算子

拉普拉斯算子是不依赖于边缘方向的二阶导数算子，它是一个标量而不是向量，具有旋转不变即各向同性的性质。若只关心边缘点的位置而不需要了解一其周围的实际灰度差时，一般选择该算子提取图像的边缘。Laplace算子的定义为：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad (2.3.9)$$

$$\frac{\partial^2 f}{\partial x^2} = f(x, y+1) - 2f(x, y) + f(x, y-1) \qquad (2.3.10)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x+1, y) - 2f(x, y) + f(x-1, y) \qquad (2.3.11)$$

将这两个式子合并，可以得到近似Laplace算子的模版：

$$\nabla^2 f \approx \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad (2.3.12)$$

当Laplace算子输出出现过零点时就表明有边缘存在，其中忽略无意义的过零点(均匀零区)。原则上，过零点的位置精度可以通过线性内插方法精确到子像素分辨率。**但是拉普拉斯算子在图像边缘检测中并不常用。主要原因有**：任何包含有二阶导数的算子比只包含有一阶导数的算子**更易受噪声的影响**，一阶导数很小的局部峰值也能导致二阶导数过零点，所以Laplace算子对噪声具有无法接受的敏感性； Laplace算子的幅值产生双边元，这是复杂的分割不希望有的结果;最后，**Laplace算子不能检测边缘的方向。为了避免噪声的影响，必须采用特别有效的滤波方法。所以，人们提出了改进的LOG算子。**

6.LOG算子（高斯拉普拉斯算子）

LOG算子基本思想是：先在一定的范围内做平滑滤波，然后再利用差分算子来检测在相应尺度上的边缘。

该方法的特点是先用高斯滤波器与图像进行卷积，既平滑了图像又降低了噪声，使孤立的噪声点和较小的结构组织被滤除。然而由于对图像的平滑会导致边缘的延展，因此只考虑那些具有局部梯度极大值的点作为边缘点，这可以用二阶导数的零交叉来实现。拉普拉斯函数可用作二维二阶导数的近似，因为它是一种标量算子。为了避免检测出非显著的边缘，所以应该选择一阶导数大于某一阈值的零交叉点来作为边缘点。实际应用中，常用的LOG算子的模版为：

$$\begin{bmatrix} -2 & -4 & -4 & -4 & -2 \\ -4 & 0 & 8 & 0 & -4 \\ -4 & 8 & 24 & 8 & -4 \\ -4 & 0 & 8 & 0 & -4 \\ -2 & -4 & -4 & -4 & -2 \end{bmatrix}$$

说明：高斯平滑运算不但可以滤除噪声，还会导致图像中的边缘和其它尖锐不连续部分模糊，而模糊程度取决于空间尺度因子σ 的大小。σ 越大，高斯滤波对噪声的滤除效果越好，但同时也会丢失重要的边缘信息，影响到边缘检测器的性能。如果σ 较小，又可能导致平滑作用不完全而留有较多的噪声。因此在实际应用中，要根据情况选择适当的σ。

不同算子的比较

Robert算子定位比较精确，但由于不包括平滑，所以对于噪声比较敏感。

Prewitt算子和Sobel算子都是一阶的微分算子，而前者是平均滤波，后者是加权平均滤波且检测的图像边缘可能大于2个像素。这两者对灰度渐变低噪声的图像有较好的检测效果，但是对于混合多复杂噪声的图像，处理效果就不理想了。

LOG滤波器方法通过检测二阶导数过零点来判断边缘点。LOG滤波器中的a正比于低通滤波器的宽度，a越大，平滑作用越显著，去除噪声越好，但图像的细节也损失越大，边缘精度也就越低。所以在边缘定位精度和消除噪声级间存在着矛盾，应该根据具体问题对噪声水平和边缘点定位精度要求适当选取。

讨论和比较了几种常用的边缘检测算子。梯度算子计算简单,但精度不高,只能检测出图像大致的轮廓,而对于比较细的边缘可能会忽略。Prewitt 和Sobel 算子比Roberts 效果要好一些。LOG 滤波器和Canny 算子的检测效果优于梯度算子,能够检测出图像较细的边缘部分。不同的系统,针对不同的环境条件和要求,选择合适的算子来对图像进行边缘检测。