一种扩展的边缘检测算法

邓沌华[1]

(1.　　湖北经济学院计算机学院，湖北 武汉 430205)

(2.

摘要：Sobel 算法是常用的图像处理的边缘检测算法，具有实现简单、速度快的优点，但是只采用两个方向模板，使得 Sobel 算子对于纹理复杂的图像，边缘检测效果较差。本文实现了一种四向加权微分算法作为对 Sobel 算法的扩展，不但产生较好的检测效果，而且对噪声具有平滑作用，可以提供较为精确的边缘定位信息。

关键词：图像处理，边缘检测，Sobel 算子，四向加权微分算法

1．边缘检测算法概述

边缘，指的是图像局部亮度变化最显著的部分，是图像分割的第一步，属于检测图像局部变化显著变化的最基本的运算。图像的大部分主要信息都存在于图像的边缘中，主要表现为图像局部特征的不连续性，是图像中灰度变化比较强烈的地方，也即通常所说的信号发生奇异变化的地方。

边缘检测算法是数字图像处理的基础，广泛应用在图像处理、图像分析、模式识别、计算机视觉以及人类视觉等领域。

经典的经典的边缘检测方法如：Roberts，Sobel，Prewitt，Kirsch，Laplace 等方法，这些算法本质上都是利用边缘处的一阶导数取极值、二阶导数在阶梯状边缘处呈零交叉或在屋顶状边缘处取极值的微分算法。具体实现方法都是通过对原始图像中象素的小邻域构造边缘检测算子，进行一阶微分或二阶微分运算，求得梯度最大值或二阶导数的过零点，最后选取适当的阀值提取边界。近年来，随着数学和人工智能技术的发展，各种类型的边缘检测算法不断涌现，如神经网络、遗传算法、数学形态学等理论运用到图像的边缘检测中。

由于这些算法涉及梯度的运算，因此均存在对噪声敏感、计算量大等缺点。具体来说也就是存在着检测精度、边缘定位精度和抗噪声等方面的矛盾及对于不同的算法边缘检测结果的精度却没有统一的衡量标准，所以至今都还不能取得令人满意的效果。

由此可见，实现边缘检测虽然有很多不同的方法，也一直是图像处理中的研究热点，但是如果希望找到一种抗噪强、定位准、不漏检、不误检的检测算法是很苦难的。经典的算法主要 用梯度算子，最简单的梯度算子是 Roberts 算子，比较常用的有 Prewitt 算子和 Sobel 算子，其中 Sobel 算子效果较好，但是经典 Sobel 算子存在边缘定位不准的缺点，本文描述的就是一种对 Sobel 算法的扩充实现。

2．Sobel 算子的理论描述

经典的 Sobel 图像边缘检测算法，是在图像空间利用两个方向模板与图像进行邻域卷积来完成的，这两个方向模板一个是检测垂直边缘，一个是检测水平边缘。

算法的基本原理：由于图像边缘附近的亮度变化较大，所以可以把那些在邻域内，灰度

变化超过某个适当阈值 TH 的像素点当作边缘点。Sobel 算法的优点是计算简单，速度快。但由于只采用了两个方向模板，只能检测水平方向和垂直方向的边缘，因此，这种算法对于纹理较复杂的图像，其边缘检测效果欠佳；同时，经典 Sobel 算法认为，凡灰度新值大于或等于阈值的像素点都是边缘点。这种判定依据是欠合理的，会造成边缘点的误判，因为多噪声点的灰度新值也很大。

现在将常用的检测实现公式列出如下：

Roberts 算子：G[i,i]=|f[i,j]-f[i+1,j+1]|+|f[i+1,j]-f[i,j+1]|;

Sobel 算子：G[i,i]=|f[i-1,j+1]+2f[i,j+1]+f[i+1,j+1]-f[i-1,j-1]-2f[i,j-1]-f[i+1,j-1]|
+|f[i-1,j-1]+2f[i-1,j]+f[i-1,j+1]-f[i+1,j-1]-2f[i+1,j]-f[i+1,j+1]|;

拉普拉斯算子：G[I,j]=|f[i+1,j]+f[i-1,j]+f(i,j+1)+f[i,j-1]-4f[i,j]|;

其中 G[i,j]表示处理后(i,j)点的灰度值，f[i,j]表示处理前该点的灰度值。

对于数字图像，可以用一阶差分代替一阶微分；

△xf(x,y)=f(x,y)-f(x-1,y);

△yf(x,y)=f(x,y)-f(x,y-1)

求梯度时对于平方和运算及开方运算，可以用两个分量的绝对值之和表示，即：

G[f(x,y)]={[△xf(x,y)] +[△yf(x,y)] } |△xf(x,y)|+|△yf(x,y)|;

Sobel 梯度算子是先做成加权平均，再微分，然后求梯度，即：

△xf(x,y)= f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)- f(x-1,y-1) - 2f(x,y-1) - f(x+1,y-1);

△yf(x,y)= f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1)- f(x+1,y-1) - 2f(x+1,y) - f(x+1,y+1);

G[f(x,y)]=|△xf(x,y)|+|△yf(x,y)|;

上述各式中的像素之间的关系见表 1：

表 1

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

边缘检测一般是通过判断一像素与其周围像素的关系进行的,可构造一定大小(如 3 ×3)的模板与原图像进行卷积得到边缘图像。

Sobel 边缘检测算子使用两个如下有向算子(一个水平的，一个是垂直的)，每一个逼近一个偏导数：

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

图 1 sobel 算子的水平和垂直方向模板

$D_1f(x,y)=\{f(x+1,y-1)+2f(x+1,y)+f(x+1,y+1)\}-\{f(x-1,y-1)+2f(x-1,y)+f(x-1,y+1)\}$

$D_2f(x,y)=\{f(x-1,y+1)+2f(x,y+1)+f(x+1,y+1)\}-\{f(x-1,y-1)+2f(x,y-1)+f(x+1,y-1)\}$

如果用 Sobel 算子检测图像 M 的边缘的话，可以先分别用水平算子和垂直算子对图像进行卷积，得到的是两个矩阵，在不考虑边界的情形下也是和原图像同样大 小的图像 M1，M2，他们分别表示图像 M 中相同位置处的两个偏导数。然后把 M1，M2 对应位置的两个数平方后相加得到一个新的矩阵 G，G 表示 M 中各个像素 的灰度的梯度值(一个逼近)。这样就可以通过阀值处理得到边缘图像。

通常物体的边缘是连续而光滑的，且边缘具有方向和幅度两个特征，而噪声是随机的。沿任一边缘点走向总能找到另一个边缘点，且这两个边缘点之间的灰度差和方向差相近。而

噪声却不同，在一般情况下，沿任一噪声点很难找到与其灰度值和方差相似的噪声点。 基于这一思想，就可以将噪声点和边缘点区分开来。

Sobel 算子利用像素的左、右、上、下邻域的灰度加权算法，根据在边缘点处达到极值这一原理进行边缘检测。该方法不但产生较好的检测效果，而且对噪声具有平滑作用，可以提供较为精确的边缘方向信息。但是，在抗噪声好的同时也存在检测到伪边缘，定位精度不高的缺点。

为解决这个问题，现在成熟的解决方案一般是通过对图像处理过程之前，增加预处理过程来解决问题。一般而言，对图片进行预处理，是为了突出图片的边缘线条部分，那么再经 Sobel 算子运算后的边缘线条将会精确得多，而 Sobel 算子的噪声抑制作用也得到保存。

预处理虽然能够解决噪声问题，提高定位精度，但是并不能完全解决图像边缘像素点的检测精度问题；这是因为，除了水平和垂直两方向外，图像的边缘还有其它的方向，如 135^o 和 45^o 等，而经典的 Sobel 算子只考虑两个方向，即 0^o 和 90^o 的方向模板，因此，为了增加算子在某一像素点检测边缘的精度，可将方向模板由 2 个增加为多个即再在经典的方向模板的基础上增加 2 个以上方向模板，如图 2 所示。

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

0^0

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

90^0

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

180^0

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

270^0

图 2　Sobel 算子的四方向模板

180 度方向算子：

$D_3f(x,y)=\{f(x-1,y-1)+2f(x-1,y)+f(x-1,y+1)\}-\{f(x+1,y-1)+2f(x+1,y)+f(x+1,y+1)\}$

270 度方向算子：

$D_4f(x,y)=\{f(x-1,y-1)+2f(x,y-1)+f(x+1,y-1)\}-\{f(x-1,y+1)+2f(x,y+1)+f(x+1,y+1)\}$

3．Sobel 扩展：四向加权微分算法

对于一幅数字图像 f(x，y)，利用上述的 8 个方向模板 Sobel 算子对图像中的每个像素计算，取得其中的 最大值作为该点的新值，而该最大值对应的模板所表示的方向为该像素点的方向。若｜f(x，y)－f(x+i，y+j)｜ > TH_2，对于任意 i=0，1，-1；j=0，1，-1 均成立，则可判断点(x，y)为噪声点。下面给出了图像边缘检测程序实现的改进算法的软件实现：

```
    void ExtSobel()
{
  HANDLE BlockHandle;
  LPBITMAPINFOHEADER lpBitmap;
  CDibDoc *pDoc=GetDocument();
```

```
   HDIB hDib;
   unsigned char *hData;
   unsigned char *data;

   hDib=pDoc->m_hDIB;
   BeginWaitCursor();
   lpBitmap=(LPBITMAPTMAPINFOHEADER)GlobalLock((HGLOBAL)hDib);
   hData= lpBitmap +* (LPDWORD)lpBitmap + 256*sizeof(RGBQUAD);
   //得到指向位图像素值的指针
   pDoc->SetModifiedFlag(TRUE);//设修改标志为"TRUE"

BlockHandle=GlobalAlloc(GMEM_SHARE,WIDTHBYTES(lpBitmap->biWidth*8)*lpBitmap->
biHeight);
   //申请存放处理后的像素值的缓冲区
   data=(unsigned char*)GlobalLock((HGLOBAL)BlockHandle);
   AfxGetApp()->BeginWaitCursor();
   int i,j,buf,buf1,buf2,buf3,buf4;
   for( j=0; jbiHeight; j++)//以下循环求(x,y)位置的灰度值
      for( i=0; ibiWidth; i++)
      {
        if(((i-1)>=0)&&((i+1)biWidth)&&((j-1)>=0)&&((j+1)biHeight))
        {//对于图像四周边界处的向素点不处理
          buf1=(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
                +2*(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j))
                +(int)(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1));
          buf1=buf1-(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
                -2*(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j))
                -(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1));
          //0 度方向加权微分
          buf2=(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1))
                +2*(int)(int)*(hData+(i)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1))
                +(int)(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1));
          buf2=buf2-(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
                -2*(int)(int)*(hData+(i)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
                -(int)(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1));
          //90 度方向加权微分
        buf3=(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
                +2*(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j))
                +(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1));
          buf3=buf1-(int)(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
                -2*(int)(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j))
                -(int)(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1));
      //180 度加权微分
          buf4=(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
```

```
                +2*(int)*(hData+(i)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1))
                +(int)(int)*(hData+(i+1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j-1));
        buf4=buf1-(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1))
                -2*(int)(int)*(hData+(i)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1))
                -(int)(int)*(hData+(i-1)*WIDTHBYTES(lpBitmap->biWidth*8)+(j+1));
    //270 度加权微分
            buf=abs(buf1)+abs(buf2)+ABS(buf3)+abs(buf4);//求梯度
        if(buf>255) buf=255;
          if(buf<0){buf=0;
                *(data+i*WIDTHBYTES(lpBitmap->biWidth*8)+j)=(BYTE)buf;
          }

        else *(data+i*lpBitmap->biWidth+j)=(BYTE)0;
        }
        for( j=0; jbiHeight; j++)
          for( i=0; ibiWidth; i++)

*(hData+i*WIDTHBYTES(lpBitmap->biWidth*8)+j)=*(data+i*WIDTHBYTES(lpBitmap->biW
idth*8)+j);
            //处理后的数据写回原缓冲区
              AfxGetApp()->EndWaitCursor();
              GlobalUnlock((HGLOBAL)hDib);
              GlobalUnlock(BlockHandle);
              GlobalFree(date1handle);
              EndWaitCursor();
              Invalidate(TRUE);
  }
```

该算法程序在 Visual C6.0 环境中编译通过并实现。

4. 总结

从经典 Sobel 算法边缘检测实例中可以看出，Soble 算子对噪声确有抑制作用，因此不会出现很多孤立的边缘像素点；但 Sobel 算子对边缘的定位不是很准确，图像的边界宽度往往不止一个像素。而经过本算法改进过的 Sobel 算子的边缘检测处理后得到的边缘图，对边缘的定位较准，边界的线条并不是很粗，且对噪声就不是那么敏感了。

参考文献:

[1] 章毓晋. 图象工程[M]. 北京：清华大学出版社, 1999.

[2] 刑军. 基于 Sobel 算子数字图像的边缘检测. 微机发展[J]. 2005, 15(9):48-52.

[3] 张少军,艾矫健,李忠富,等. 利用图像处理技术测量几何尺寸[J] . 北京科技大学学报,2002 ,24 (3) :284 - 287.

[4] Health A., Sarkar S., Sanocki T., et al.. Comparison of Edge Detectors: A Methodology and Initial Study. Computer Vision and Image Understanding[J]. 1998, 69(1): 38-54.