Python+OpenCV图像处理

Sobel算子、Laplacian算子和Canny边缘检测

在对图像进行Sobel算子 、Laplacian算子和Canny 边缘检测时，需要先安装和配置OpenCV。具体方法是在OpenCV官网上下载OpenCV，按照自己需要选择不同的版本，这里我下载的是opencv-3.2.0-vc14，按照E:\Opencv\build\python\2.7\x64的路径查找名称为cv2的PYD文件，复制粘贴到安装的Python27的位置如 C:\Python27\Lib\site-packages里。之后安装pip和numpy，安装成功后要配置pip的环境变量，把文件路径复制到path中。之后就安装好啦。

配置好OpenCV后，把需要处理的图片放到文件夹里，修改程序中的文件读取位置，就可以进行图像处理了。下面介绍一下三种图像处理的方法。

**Sobel算子**

Sobel算子是滤波算子的形式，用于提取边缘，可以利用快速[卷积](http://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF)函数， 简单有效，因此应用广泛。Sobel算子是计算机视觉领域的一种重要处理方法。主要用于获得数字图像的一阶梯度，常见的应用和物理意义是[边缘检测](http://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B)。在技术上，它是一个离散的一阶[差分](http://baike.baidu.com/item/%E5%B7%AE%E5%88%86)算子，用来计算图像亮度函数的一阶梯度之近似值。在图像的任何一点使用此算子，将会产生该点对应的梯度矢量或是其法矢量。

### Sobel算子的核心公式

该算子包含两组3x3的[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)，分别为横向及纵向，将之与图像作平面[卷积](http://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF)，即可分别得出横向及纵向的亮度[差分](http://baike.baidu.com/item/%E5%B7%AE%E5%88%86)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](http://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B)的图像，其公式如下:

[http://c.hiphotos.baidu.com/baike/s%3D250/sign=ac3881323f6d55fbc1c671235d234f40/11385343fbf2b21119bc68a2c88065380dd78e48.jpg](http://baike.baidu.com/pic/Sobel%E7%AE%97%E5%AD%90/11000092/0/11385343fbf2b21119bc68a2c88065380dd78e48?fr=lemma&ct=single)

图像的每一个像素的横向及纵向梯度近似值可用以下的公式结合，来计算梯度的大小。

[http://a.hiphotos.baidu.com/baike/s%3D220/sign=85a0a5cb3901213fcb3349de64e636f8/cdbf6c81800a19d8b48e151733fa828ba61e4616.jpg](http://baike.baidu.com/pic/Sobel%E7%AE%97%E5%AD%90/11000092/0/718e25c7978d5cfed0006077?fr=lemma&ct=single)

可用以下公式计算梯度方向。

[http://e.hiphotos.baidu.com/baike/s%3D220/sign=4662f30e11dfa9ecf92e511552d1f754/023b5bb5c9ea15ce8262711bb6003af33a87b210.jpg](http://baike.baidu.com/pic/Sobel%E7%AE%97%E5%AD%90/11000092/0/969cbf441985fda0b3b7dc71?fr=lemma&ct=single)

在以上例子中，如果以上的角度Θ等于零，即代表图像该处拥有纵向边缘，左方较右方暗。

### Sobel算子描述

在边缘检测中，常用的一种模板是Sobel 算子。Sobel 算子有两个，一个是检测水平边缘的 ；另一个是检测垂直边缘的 。与Prewitt算子相比，Sobel算子对于像素的位置的影响做了加权，可以降低边缘模糊程度，因此效果更好。

Sobel算子另一种形式是各向同性Sobel(Isotropic Sobel)算子，也有两个，一个是检测水平边缘的 ，另一个是检测垂直边缘的 。各向同性Sobel算子和普通Sobel算子相比，它的位置加权系数更为准确，在检测不同方向的边沿时梯度的幅度一致。将Sobel算子矩阵中的所有2改为根号2，就能得到各向同性Sobel的矩阵。

**Sobel算子的python代码如下：**

# coding=utf-8  
**import** cv2  
**import** numpy **as** np  
**import** os  
  
# 此功能需要在电脑上预先安装opencv  
# path = 'original//1\_22//03//' # 图片读取的目录，更改成自己需要的  
path = 'C:/Users/sls/Desktop/1/'  
pathDir = os.listdir(path)  
**for** eachDir **in** pathDir:  
 imgPath = path + eachDir  
 **print** imgPath  
 img = cv2.imread(imgPath, 0)  
 img = cv2.GaussianBlur(img, (3, 3), 0)  
 canny = cv2.Sobel(img, cv2.CV\_16S, 0, 2) # 第二个参数是图像深度， 第三个第四个参数dx，dy是求导次数，似乎求导次数越多边缘越少  
 cv2.imshow('Canny', canny)  
 cv2.imwrite("C:/Users/sls/Desktop/sobel02/" + eachDir, canny) # 图片输出的目录，更改成自己需要的  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

**原图：**

C:\Users\sls\Desktop\1\Image_000010000_000020000.tif C:\Users\sls\Desktop\1\Image_000000000_000010000.tif C:\Users\sls\Desktop\1\Image_000020000_000030000.tif

**Gx及Gy设置为1和0的处理结果：**

C:\Users\sls\Desktop\sobel\Image_000010000_000020000.tif C:\Users\sls\Desktop\sobel\Image_000000000_000010000.tif C:\Users\sls\Desktop\sobel\Image_000020000_000030000.tif

与原图对比可发现竖条纹明显

**Gx及Gy设置为0和1的处理结果**：

C:\Users\sls\Desktop\sobel01\Image_000010000_000020000.tif C:\Users\sls\Desktop\sobel01\Image_000000000_000010000.tif C:\Users\sls\Desktop\sobel01\Image_000020000_000030000.tif

与原图对比横条纹明显

Gx及Gy设置为2和0的处理结果：

C:\Users\sls\Desktop\sobel2\Image_000010000_000020000.tif C:\Users\sls\Desktop\sobel2\Image_000000000_000010000.tif C:\Users\sls\Desktop\sobel2\Image_000020000_000030000.tif

### Sobel算子处理结果总结

当Gx越大（一般取0，1，2）横条纹越不明显，竖条纹越明显

当Gy越大（一般取0，1，2）横条纹越明显，竖条纹越不明显

**Laplacian算子**

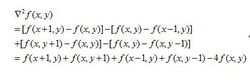
### Laplacian算子定义

Laplacian 算子是n维欧几里德空间中的一个二阶微分算子，定义为梯度grad（）的散度div（）。如果f是二阶[可微](http://baike.baidu.com/item/%E5%8F%AF%E5%BE%AE)的实函数，则f的拉普拉斯算子定义为：

(1) f的拉普拉斯算子也是笛卡儿坐标系xi中的所有非混合二阶[偏导数](http://baike.baidu.com/item/%E5%81%8F%E5%AF%BC%E6%95%B0)求和：

(2) 作为一个二阶[微分算子](http://baike.baidu.com/item/%E5%BE%AE%E5%88%86%E7%AE%97%E5%AD%90)，拉普拉斯算子把C函数映射到C函数，对于k ≥ 2。表达式(1)（或(2)）定义了一个算子Δ : C(R) → C(R)，或更一般地，定义了一个算子Δ : C(Ω) → C(Ω)，对于任何[开集](http://baike.baidu.com/item/%E5%BC%80%E9%9B%86)Ω。

对于阶跃状边缘，[导数](http://baike.baidu.com/item/%E5%AF%BC%E6%95%B0)在边缘点出现零交叉，即边缘点两旁[二阶导数](http://baike.baidu.com/item/%E4%BA%8C%E9%98%B6%E5%AF%BC%E6%95%B0)取异号。据此，对[数字图像](http://baike.baidu.com/item/%E6%95%B0%E5%AD%97%E5%9B%BE%E5%83%8F){f（i，j）}的每个像素，取它关于x轴方向和y轴方向的[二阶差分](http://baike.baidu.com/item/%E4%BA%8C%E9%98%B6%E5%B7%AE%E5%88%86)之和，表示为



### Laplacian算子运算模板

函数的拉普拉斯算子也是该函数的黑塞矩阵的迹,可以证明，它具有各向同性，即与坐标轴方向无关，坐标轴旋转后梯度结果不变。如果邻域系统是4 邻域，Laplacian 算子的模板为：

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

如果邻域系统是8 邻域，Laplacian 算子的模板为：

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

Laplacian 算子对噪声比较敏感，所以图像一般先经过平滑处理，因为平滑处理也是用模板进行的，所以，通常的分割算法都是把Laplacian 算子和平滑算子结合起来生成一个新的模板。

**Laplacian算子的python代码如下：**

# coding=utf-8  
**import** cv2  
**import** numpy **as** np  
**import** os  
# path = 'original//1\_22//03//' # 图片读取的目录，更改成自己需要的  
path = 'C:/Users/sls/Desktop/1/'  
pathDir = os.listdir(path)  
**for** eachDir **in** pathDir:  
 imgPath = path + eachDir  
 **print** imgPath  
 img = cv2.imread(imgPath, 0)  
 img = cv2.GaussianBlur(img, (3, 3), 0)  
 canny = cv2.Laplacian(img, cv2.CV\_16S) # 第二个参数是图像深度  
 # cv2.imshow('Canny', canny)  
 cv2.imwrite("C:/Users/sls/Desktop/laplacian/" + eachDir, canny) # 图片输出的目录，更改成自己需要的  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

**原图：**

C:\Users\sls\Desktop\1\Image_000010000_000020000.tif C:\Users\sls\Desktop\1\Image_000000000_000010000.tif C:\Users\sls\Desktop\1\Image_000020000_000030000.tif

**Laplacian 算子处理后的图片：**

**C:\Users\sls\Desktop\laplacian\Image_000010000_000020000.tif C:\Users\sls\Desktop\laplacian\Image_000000000_000010000.tif C:\Users\sls\Desktop\laplacian\Image_000020000_000030000.tif**

### Laplacian算子处理结果总结

处理过后可以发现横竖条纹都比较明显了

Canny边缘检测

Canny 的目标是找到一个最优的[边缘检测](http://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B)算法，最优边缘检测的含义是：

好的检测- 算法能够尽可能多地标识出图像中的实际边缘。

好的定位- 标识出的边缘要尽可能与实际图像中的实际边缘尽可能接近。

最小响应- 图像中的边缘只能标识一次，并且可能存在的图像噪声不应标识为边缘。

为了满足这些要求 Canny 使用了[变分法](http://baike.baidu.com/item/%E5%8F%98%E5%88%86%E6%B3%95)，这是一种寻找满足特定功能的函数的方法。最优检测使用四个[指数函数](http://baike.baidu.com/item/%E6%8C%87%E6%95%B0%E5%87%BD%E6%95%B0)项的和表示，但是它非常近似于[高斯函数](http://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E5%87%BD%E6%95%B0)的一阶[导数](http://baike.baidu.com/item/%E5%AF%BC%E6%95%B0)。

**Canny边缘检测的步骤**

### 去噪声

任何[边缘检测](http://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B)算法都不可能在未经处理的[原始数据](http://baike.baidu.com/item/%E5%8E%9F%E5%A7%8B%E6%95%B0%E6%8D%AE)上很好地处理，所以第一步是对原始数据与高斯平滑模板作[卷积](http://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF)，得到的图像与原始图像相比有些轻微的模糊（blurred）。这样，单独的一个像素噪声在经过高斯平滑的图像上变得几乎没有影响。

### 寻找图像中的亮度梯度

图像中的边缘可能会指向不同的方向，所以 Canny 算法使用 4 个 mask 检测水平、垂直以及对角线方向的边缘。原始图像与每个 mask 所作的[卷积](http://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF)都存储起来。对于每个点我们都标识在这个点上的最大值以及生成的边缘的方向。这样我们就从原始图像生成了图像中每个点亮度梯度图以及亮度梯度的方向。

### 在图像中跟踪边缘

较高的亮度梯度比较有可能是边缘，但是没有一个确切的值来限定多大的亮度梯度是边缘多大，所以 Canny 使用了[滞后](http://baike.baidu.com/item/%E6%BB%9E%E5%90%8E)阈值。

滞后阈值需要两个阈值——高阈值与低阈值。假设图像中的重要边缘都是连续的曲线，这样我们就可以跟踪给定曲线中模糊的部分，并且避免将没有组成曲线的噪声像素当成边缘。所以我们从一个较大的阈值开始，这将标识出我们比较确信的真实边缘，使用前面导出的方向信息，我们从这些真正的边缘开始在图像中跟踪整个的边缘。在跟踪的时候，我们使用一个较小的阈值，这样就可以跟踪曲线的模糊部分直到我们回到起点。

一旦这个过程完成，我们就得到了一个二值图像，每点表示是否是一个边缘点。

一个获得亚像素精度边缘的改进实现是在梯度方向检测二阶方向导数的过零点

[http://e.hiphotos.baidu.com/baike/s%3D250/sign=ce47308ba8d3fd1f3209a53f004f25ce/aa18972bd40735faaa01ed1c9e510fb30e2408a5.jpg](http://baike.baidu.com/pic/Canny%E7%AE%97%E5%AD%90/8821789/0/aa18972bd40735faaa01ed1c9e510fb30e2408a5?fr=lemma&ct=single)

它在梯度方向的三阶方向[导数](http://baike.baidu.com/item/%E5%AF%BC%E6%95%B0)满足符号条件

[http://b.hiphotos.baidu.com/baike/s%3D250/sign=67061b24cb1349547a1eef61664f92dd/fd039245d688d43fbe4d32577d1ed21b0ff43baa.jpg](http://baike.baidu.com/pic/Canny%E7%AE%97%E5%AD%90/8821789/0/fd039245d688d43fbe4d32577d1ed21b0ff43baa?fr=lemma&ct=single)

其中

[http://c.hiphotos.baidu.com/baike/s%3D220/sign=d94914b3cffc1e17f9bf8b337a91f67c/f603918fa0ec08facbddb00259ee3d6d54fbdab2.jpg](http://baike.baidu.com/pic/Canny%E7%AE%97%E5%AD%90/8821789/0/f603918fa0ec08facbddb00259ee3d6d54fbdab2?fr=lemma&ct=single)

[http://d.hiphotos.baidu.com/baike/s%3D220/sign=4da5a0623d6d55fbc1c671245d234f40/11385343fbf2b211f82149f2ca8065380cd78e7b.jpg](http://baike.baidu.com/pic/Canny%E7%AE%97%E5%AD%90/8821789/0/11385343fbf2b211f82149f2ca8065380cd78e7b?fr=lemma&ct=single)

...

[http://f.hiphotos.baidu.com/baike/s%3D220/sign=2b35b9bbb9389b503cffe750b534e5f1/838ba61ea8d3fd1f5cd0019f304e251f94ca5fbd.jpg](http://baike.baidu.com/pic/Canny%E7%AE%97%E5%AD%90/8821789/0/838ba61ea8d3fd1f5cd0019f304e251f94ca5fbd?fr=lemma&ct=single)

表示用高斯核平滑原始图像得到的[尺度空间](http://baike.baidu.com/item/%E5%B0%BA%E5%BA%A6%E7%A9%BA%E9%97%B4)表示 L 计算得到的偏[导数](http://baike.baidu.com/item/%E5%AF%BC%E6%95%B0)。用这种方法得到的边缘片断是连续曲线，这样就不需要另外的边缘跟踪改进。[滞后](http://baike.baidu.com/item/%E6%BB%9E%E5%90%8E)阈值也可以用于亚像素[边缘检测](http://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B)。

### Canny边缘检测的参数

Canny 算法包含许多可以调整的参数，它们将影响到算法的计算的时间与实效。

[高斯滤波器](http://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E6%BB%A4%E6%B3%A2%E5%99%A8)的大小：第一步所用的平滑滤波器将会直接影响 Canny 算法的结果。较小的滤波器产生的模糊效果也较少，这样就可以检测较小、变化明显的细线。较大的滤波器产生的模糊效果也较多，将较大的一块图像区域涂成一个特定点的颜色值。这样带来的结果就是对于检测较大、平滑的边缘更加有用，例如彩虹的边缘。

阈值：使用两个阈值比使用一个阈值更加灵活，但是它还是有阈值存在的共性问题。设置的阈值过高，可能会漏掉重要信息；阈值过低，将会把枝节信息看得很重要。很难给出一个适用于所有图像的通用阈值。目前还没有一个经过验证的实现方法。

**Canny边缘检测算子的python代码如下：**

# coding=utf-8  
**import** cv2  
**import** numpy **as** np  
**import** os  
# 此功能需要在电脑上预先安装opencv  
path = 'C:/Users/sls/Desktop/1/' # 图片读取的目录，更改成自己需要的  
pathDir = os.listdir(path)  
**for** eachDir **in** pathDir:  
 imgPath = path + eachDir  
 **print** imgPath  
 img = cv2.imread(imgPath, 0)  
 img = cv2.GaussianBlur(img, (3, 3), 0)  
 canny = cv2.Canny(img, 1, 50) # 此处参数是范围，数值越大越不细致，具体数值根据图片情况更改  
 # cv2.imshow('Canny', canny)  
 cv2.imwrite("C:/Users/sls/Desktop/canny1/" + eachDir, canny) # 图片输出的目录，更改成自己需要的  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

**原图：**

C:\Users\sls\Desktop\1\Image_000010000_000020000.tif C:\Users\sls\Desktop\1\Image_000000000_000010000.tif C:\Users\sls\Desktop\1\Image_000020000_000030000.tif

**高阈值与低阈值分别为300和200时：**

**C:\Users\sls\Desktop\canny\Image_000010000_000020000.tif C:\Users\sls\Desktop\canny\Image_000000000_000010000.tif C:\Users\sls\Desktop\canny\Image_000020000_000030000.tif**

**高阈值与低阈值分别为200和100时：**

**C:\Users\sls\Desktop\canny100\Image_000010000_000020000.tif C:\Users\sls\Desktop\canny100\Image_000000000_000010000.tif C:\Users\sls\Desktop\canny100\Image_000020000_000030000.tif**

**高阈值与低阈值分别为100和50时：**

**C:\Users\sls\Desktop\canny50\Image_000010000_000020000.tif C:\Users\sls\Desktop\canny50\Image_000000000_000010000.tif C:\Users\sls\Desktop\canny50\Image_000020000_000030000.tif**

**高阈值与低阈值分别为50和1时：**

**C:\Users\sls\Desktop\canny1\Image_000010000_000020000.tif C:\Users\sls\Desktop\canny1\Image_000000000_000010000.tif C:\Users\sls\Desktop\canny1\Image_000020000_000030000.tif**

### Canny算子处理结果总结

**对比可以发现阈值设置的过高可能会漏掉重要信息，阈值设置过低将会把枝节信息看的很重要**

**原图 sobel**



**Laplacian** **canny** ‘