数字图像分析
Canny 边缘检测算法

康昇
SA19010015

日期：2020 年 6 月 13 日

摘　　要

本文使用 Python 实现了 Canny 边缘检测算法，基于现有的工作，对其中非极大值抑制和双阈值选取部分进行了额外优化，并且使用 C++ 及 OpenCV 实现了对应的 C++ 版本，同时基于对比实验，论证了所作出的改进的效果。

关键词： Canny 边缘检测，非极大值抑制

1　背景介绍

1.1　Canny 边缘检测

Canny 边缘检测 [1] 是 Canny 于 1986 年提出的边缘检测算法，其结构及实现过程较为简单，但结果非常优秀，是一个经典的边缘检测算法。该算法主要分为四部分，即高斯模糊，计算图片梯度赋值，非极大值抑制和双阈值选取。

1.2　已有工作

本文首先复现了使用 Python 的 Canny 边缘检测，相比已有工作 [2][3]，本文改进了其中的非极大值抑制和双阈值选取部分的代码，使其更加鲁棒，易于使用。

2　复现过程

2.1　整体框架

本文中通过创建一个`Canny`类来实现 Canny 边缘检测，考虑到 OpenCV 已经有 Canny 边缘检测算法，因此需要在输出结果中添加 OpenCV 实现的 Canny 边缘检测作为对比，又考虑到双阈值选取部分有两个阈值，其设定的值会影响最终结果，因此需要一个 GUI 来动态接受用户设定，并返回检测结果，从而使得用户可以连续调整阈值，获取其想要的最佳效果。

因此`Canny`类的初始化参数应该包括是否创建 GUI，高阈值的低阈值的值，已经高斯模糊的参数，包括方差和模糊核大小。

`Canny`类按照图片读取，高斯模糊，计算梯度，非极大值抑制，双阈值选取，显示最终结果或创建 GUI 来处理图片，之后依次介绍这几个步骤。

2.2 图片读取

图片读取函数较为简单，需要注意的地方有

- 如果图片尺寸较大，则需要调整图片大小，否则 GUI 界面无法全部展示
- 需要保存读入的图片，在最终输出时作为对比
- 如果是彩色图片，需要调整为黑白图片，因为边缘检测一般不考虑颜色

2.3 高斯模糊

高斯模糊的作用是通过平滑来去除噪声，因为噪声造成的亮度图片会引入额外的边缘。假定模糊核大小为 $2k+1$，则模糊核 (i,j) 位置的值为

$$H[i,j] = \frac{1}{2\pi\sigma^2} e^{-\frac{(i-k)^2+(j-k)^2}{2\sigma^2}}$$

。

为简化代码，本文中在时域实现高斯模糊，即直接将高斯模糊核与原图像做卷积，而非在频域做乘积后再转回频域。原彩色图像，转化得到的灰度图及高斯模糊后的图像如图1所示。

(a) 原图　　**(b)** 灰度图　　**(c)** 高斯模糊图

图 1: 原图，灰度图及高斯模糊处理后的图像

2.4 计算梯度

计算梯度时，可以采用多种算子。本文中采用了 Sobel 算子。Sobel 算子为

$$H_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, H_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

将 Sobel 算子与原图做卷积，此处可以直接调用之前计算高斯模糊时设计的卷积函数。原图像 x 方向和 y 方向的梯度。同时本文中还需要每个像素位置梯度的幅值，即 $m = \sqrt{d_x^2 + d_y^2}$。

对得到的梯度图和幅值图归一化，x，y 方向梯度图及幅值如图2所示。可以看到，幅值图2c已经接近本文中想要的结果，但是幅值图边界检测到的边界较粗，而且有许多较小的梯度需要去除。因此引入之后的步骤，即非极大值抑制和双阈值选取。

(a) x 方向梯度图　　**(b)** y 方向梯度图　　**(c)** 幅值图

图 2: 原图，灰度图及高斯模糊处理后的图像

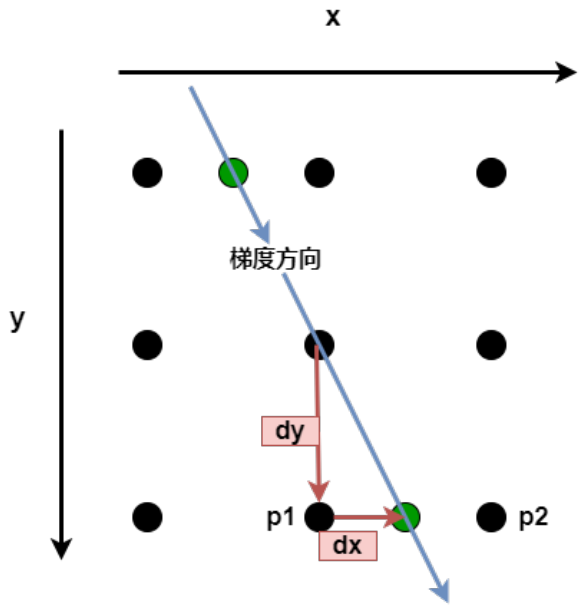


图 3: 插值过程示意图

2.5　非极大值抑制

因为幅值图2c得到的边界较粗，因此本文中考虑将其边界变细。边界较粗的原因是，一般来说，图像从一个灰度变化到另一个灰度的过程中，灰度值不会突变，而是在一定范围内渐变。因此在梯度方向，本文中只需要寻找一个最大的值，保留这个值，其余值置为 0 即可。但梯度方向不可能一直沿着坐标轴变化的方向，因此需要沿着梯度方向进行差值。

如图3所示，为了得到梯度方向的梯度，需要对 p_1 和 p_2 进行差值，得到绿色的点 p，假设 x 方向的梯度为 d_x，y 方向的梯度为 d_y，则

$$p = p_1 * \frac{d_x}{d_y} + p_2 * (1 - \frac{d_x}{d_y})$$

其他情况可类推得到。

经过非极大值抑制处理后的图像如图4所示。可以看到此时虽然边缘较细，但是有许多灰色部分，即幅值梯度较小的部分，而且本文中所需要的边缘是一个二值化图像，所以本文中需要引入最后一个步骤，即双阈值选取。

图 4: 对梯度幅值图进行非极大值抑制的结果

2.6 双阈值选取

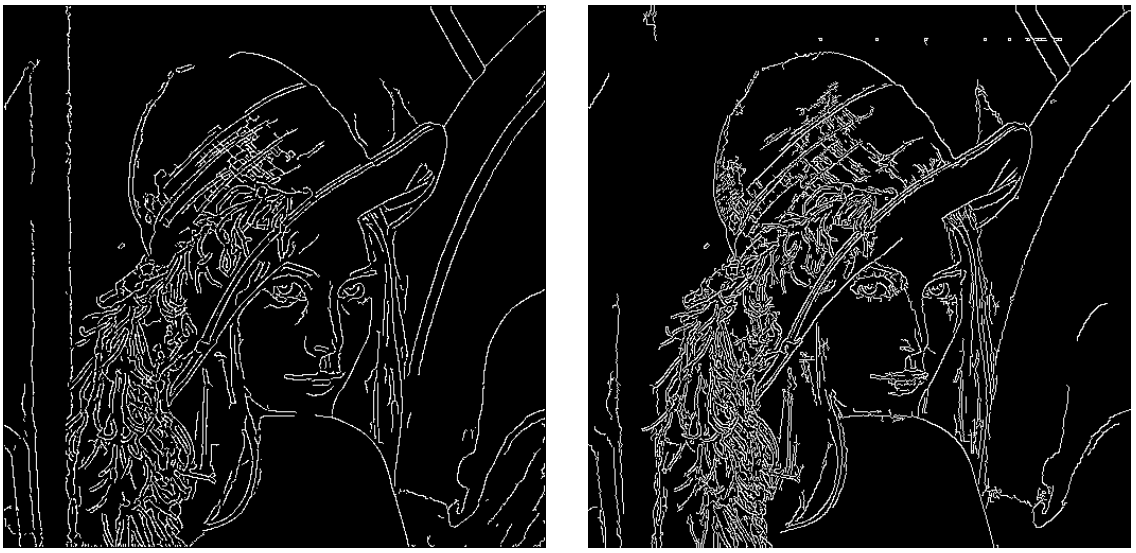
对于非极大值抑制后的结果，本文中有一个基本的假设，即有些本文中不想要的边界的值，和另外一些本文中想要的边界的值是相同的。因此如果只设定一个简单的阈值，阈值太低则会引入很多不想要的细节，阈值太高则会忽略掉本文中需要的边界。因此设定两个阈值 $0 \leq thresh_{low} \leq thresh_{high} \leq 255$，如果某处梯度的幅值大于 $thresh_{high}$，则直接认定为边界，记为强边界；如果某处梯度的幅值小于 $thresh_{low}$，则直接认定为非边界；如果在这两者之间，则记为弱边界，如果弱边界与某个强边界相连，则该点同样也被认定为强边界，否则认定为非边界。

在实际编程中，如果直接判断某个弱边界八邻域内是否有强边界，如果有就认定为强边界，否则认定为非边界，这种算法是有问题的。本文中可以构造一个特殊的图形，使得其只有一条边界，通过选取边界值，使得从左到右所有像素点都是弱边界，只有最右端一个像素点是强边界。则如果从左向右遍历这种朴素算法，则只有最右端的像素点被认定为边界，但如果从右向左遍历图形，则所有弱边界均被认定为强边界。

因此，本文中首先找到所有强边界，之后从每个强边界点出发，用 BFS 算法遍历图像，同时为了防止重复遍历，用一个二维数组来标记访问过的节点。在实际编程中，必须用队列来储存 BSF 中待遍历的点，而不能用递归算法，因为每个像素点要创建一个函数调用栈，很可能造成栈溢出。

在调试最合适的阈值的过程中，发现可以实际调整的范围很小，分析后发现，是因为幅值图中，大部分都是 0 值或是很小的值。因此，在确定 $thresh_low$ 后，把幅值图中，低于 $thresh_low$ 的像素值置为 0，之后把大于 0 的部分的灰度，做一次直方图均衡化，即可增大动态范围调整的区间，方便找到合适的阈值。

本文提出的边缘检测结果和 OpenCV 官方提出的检测结果的对比如图5所示。



(a) 本文中的结果　　**(b)** OpenCV 提供的结果

图 5: 本文中的边缘检测结果和 OpenCV 的边缘检测结果

2.7 GUI 界面

因为检测算法有两个阈值需要动态调整，因此本文中还设计了一个用户图形界面，如图6所示。如图6a所示，用户图形界面初始情况下会展示待检测的图像，以及默认的两个阈值。设定好阈值后，点击下方的“更新阈值并检测边界”按钮，即可看到如图6b所示的结果，此后可以继续修改边界并查看结果。

3 C++ 版本

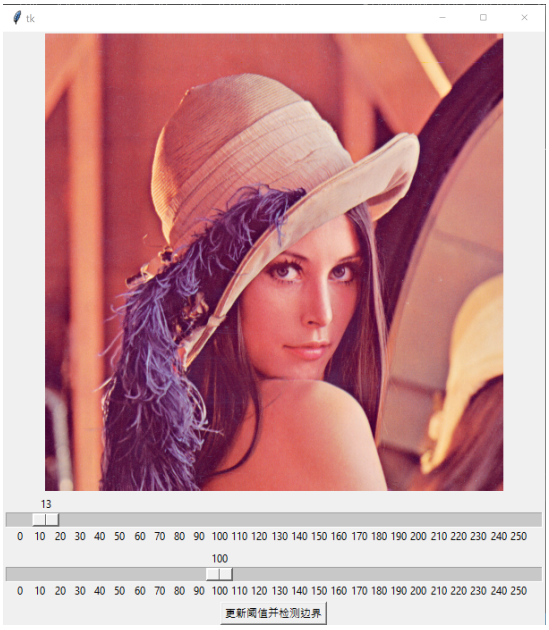
因为 Python 版本运行速度较慢，所以本文中又重新实现了 C++ 版本的 Canny 边缘检测，在 C++ 版本中，我们用到了 OpenCV 库中函数读取以及`Mat`类来管理数据。因为 OpenCV 不适合用作设计 GUI 界面，而 C++ 代码运行速度较快，因此在 C++ 版本中，把交互式界面替换为调用系统摄像头，并实时显示摄像头捕捉到画面的，运行界面如图7所示。

测试 Python 代码中边缘检测和 C++ 代码中边缘检测 lenna 图片，大小为 $512 * 512$，记录所用时间，其中不包括图片读取和写入的时间。结果如表1所示。

表 1: Canny 边缘检测各种实现的时间对比

语言	实现者	所用时间 (s)
Python	本文	5.52
C++	本文	0.052
C++	OpenCV	0.056

可以看到，本文实现的 Canny 边缘检测的速度略快于 OpenCV 官方提供的 Canny 边缘检测的速度，远远好于 Python 版本的运行速度。



(a) 初始界面

(b) 尝试阈值界面

图 6: 用户图形界面

图 7: 调用摄像头实现的实时边缘检测

参考文献

[1] Canny J. A computational approach to edge detection[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986, PAMI-8(6):679-698.

[2] caoqi95. Python 实现 Canny 边缘检测算法[EB/OL]. 2019. https://www.jianshu.com/p/effb2371ea12.

[3] Sofiane Sahir. Canny edge detection step by step in python – computer vision[EB/OL]. 2019. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.