

计算机视觉项目 EX2 实验报告

——16340220 王培钰

项目内容：

Ex2：用 CImg 重写、封装给定的 Canny 代码，并测试：

1. 附件有三个 Canny 相关的 Code 以及测试数据若干(测试数据自己转化成 BMP 图像);
2. 同学按照各自学号最末尾的数字除 3 取余数，余数为 0 的改写 Code0，余数为 1 的改写 Code1，余数为 2 的改写 Code2;
3. 封装要求：(1)所有的图像读写、数据处理只能用 CImg 库(整个工程文件不允许使用 Opencv 之类的第三方库); (2)代码封装要求函数接口简洁清晰，可参考 Code2 的方式封装。
4. 在原来的代码基础上，增加一个函数：首先把相邻的边缘连成长的线条，并删除长度小于 20 的 Edge。
5. 对算法的若干组参数，对所有测试图像进行测试，并分析各参数对结果的影响。

我的学号尾号为 0，所以改写的代码是 Code0。

实现过程：

原理：

实现 canny 边缘检测的原理通俗来说就是用离散化的梯度逼近函数根据二维灰度矩阵梯度向量来寻找图像灰度矩阵的灰度跃变位置，然后再图像中将这此点连起来就形成了图像的边缘。

一般对图像进行 canny 边缘检测要经过以下几步：

1. 对图像进行灰度化处理： $G_{img} = 0.299R + 0.587G + 0.114B$

```
void RGBtoGray();
```

2. 对图像进行高斯滤波(出去图片中噪声对边缘检测的影响)

```
void gaussian_smooth(float sigma);
```

用到辅助函数：

```
void make_gaussian_kernel(float sigma, float **kernel, int *windowsize);
```

3. 用一阶偏导的有限差分来计算梯度的幅值和方向

```
//计算x,y方向的一阶导数
```

```
void derrivative_x_y();
```

用到辅助函数：

```
double angle_radians(double x, double y);
```

```
//计算梯度向上的方向，以正x轴为逆时针方向指定的弧度
```

```
void radian_direction(int xdirtag, int ydirtag);
```

```
//计算梯度的幅值
```

```
void magnitude_x_y();
```

4. 对梯度幅值进行非极大值抑制

```
void non_max_supp();
```

5. 双阈值检测和连接边缘

```
void apply_hysteresis(float tlow, float thigh);
```

用到辅助函数：

```
void follow_edges(int *edgemaptr, int *edgemagptr, int lowval, int cols);
```

6. 之后再对将相邻边缘连成线条：

```
CImg<int> canny_line(CImg<int> picture, int distance);
```

用到辅助函数：

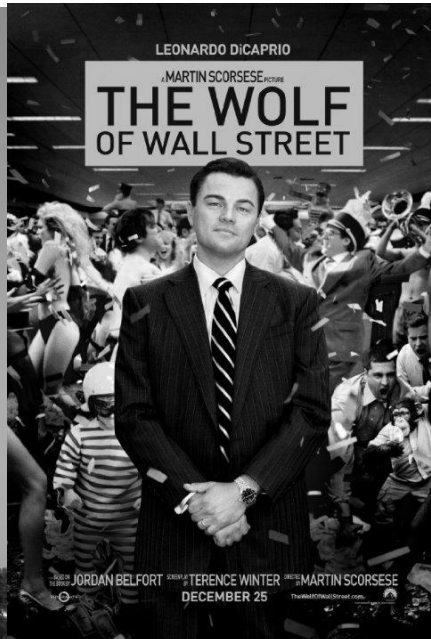
```
CImg<int> Draw_line(CImg<int> tmp, int x, int y, int xl, int yl);
```

7. 删除长度小于20的线条：

```
CImg<int> delete_line(CImg<int> picture);
```

实现过程（对每一步函数进行的测试）：

测试灰度生成图：



测试高斯模糊函数，用3*3的高斯核以及3.0的标准差进行测试，效果如下：

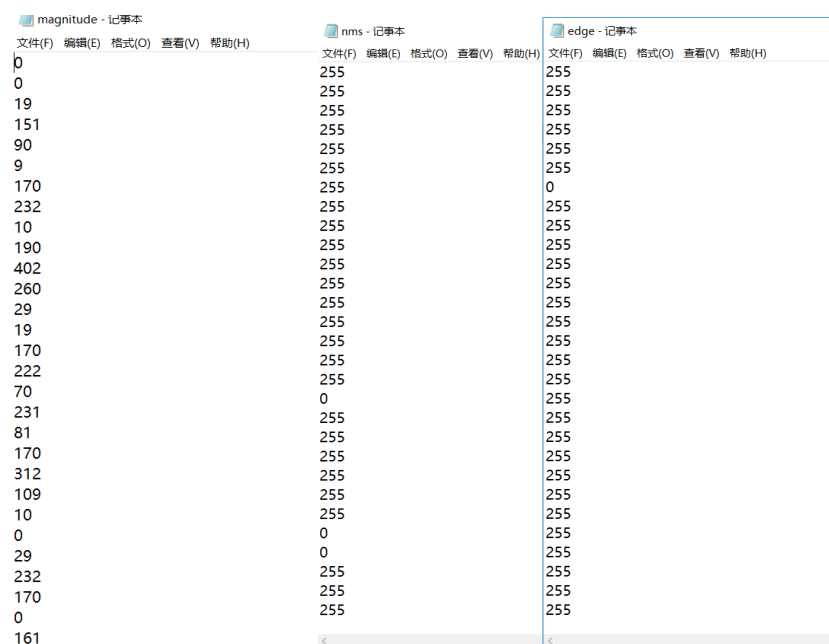


接下来是求梯度的幅值方向，非极大值抑制，以及双阈值检测求边缘像素点：

（采用的方法是对于同一幅图导出canny_source.c中生成的数据到一个txt中，然后将这个txt读入自己写的函数测试下一步对应的效果）

例如下面从左到右分别是梯度幅值的数据，非极大值抑制的数据，边缘点数据：

edge	2018/10/13 16:20	文本文档	2,620 KB
magnitude	2018/10/13 15:51	文本文档	1,286 KB
nms	2018/10/13 16:01	文本文档	1,192 KB



最后再测试一下实现的函数是否有真正达到连线删线的效果。

将每一步过后的黑点数目打印出来，发现经过连线后像素点数目比初步处理要多，经过删线后像素点数目减少。

```
CImg<int> delete_line(CImg<int> picture);
```

```
CImg<int> picture = img.canny_image();
int count = 0;
cimg_forXY(picture, x, y) {
    if (picture(x, y) == 0) {
        count++;
    }
}

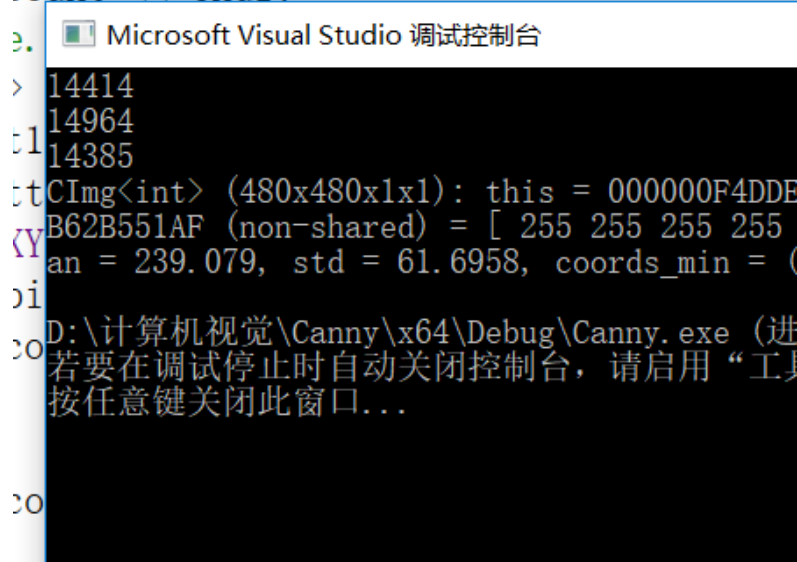
cout << count << endl;

CImg<int> picture_a = img.canny_line(picture, 10);
int count1 = 0;
int countt = 0;
cimg_forXY(picture_a, x, y) {
    if (picture_a(x, y) == 0) {
        count1++;
    }
    else
        countt++;
}

cout << count1 << endl;

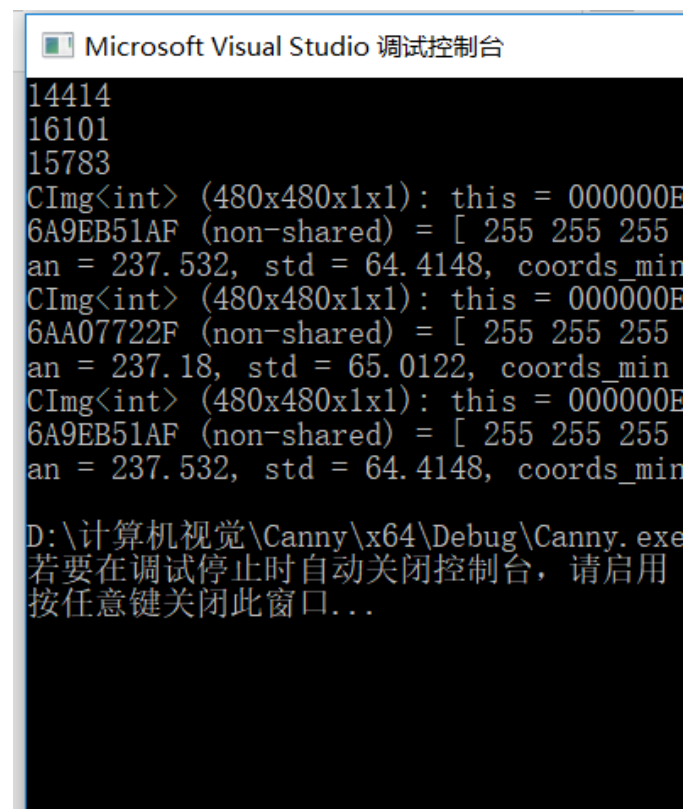
CImg<int> picture_b = img.delete_line(picture_a);
int count2 = 0;
```

```
cimg_forXY(picture_b, x, y) {  
    if (picture_b(x, y) == 0) {  
        count2++;  
    }  
}  
cout << count2 << endl;  
count << endl;
```



The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio 调试控制台". The output text is as follows:
> 14414
14964
t1 14385
t1 CImg<int> (480x480x1x1): this = 000000F4DDE
B62B551AF (non-shared) = [255 255 255 255
(Y) an = 239.079, std = 61.6958, coords_min = (
oi D:\计算机视觉\Canny\x64\Debug\Canny.exe (进
20 若要在调试停止时自动关闭控制台, 请启用“工
按任意键关闭此窗口...
20

如果将画直线的距离distance改为20:



The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio 调试控制台". The output text is as follows:
14414
16101
15783
CImg<int> (480x480x1x1): this = 000000E
6A9EB51AF (non-shared) = [255 255 255
an = 237.532, std = 64.4148, coords_min
CImg<int> (480x480x1x1): this = 000000E
6AA07722F (non-shared) = [255 255 255
an = 237.18, std = 65.0122, coords_min
CImg<int> (480x480x1x1): this = 000000E
6A9EB51AF (non-shared) = [255 255 255
an = 237.532, std = 64.4148, coords_min
D:\计算机视觉\Canny\x64\Debug\Canny.exe
若要在调试停止时自动关闭控制台, 请启用
按任意键关闭此窗口...

图像调参测试：

(默认参数值设置为 $\sigma = 2.0$, $t_{low} = 2.5$, $t_{high} = 7.5$, $distance = 10$)

Lena. jpg

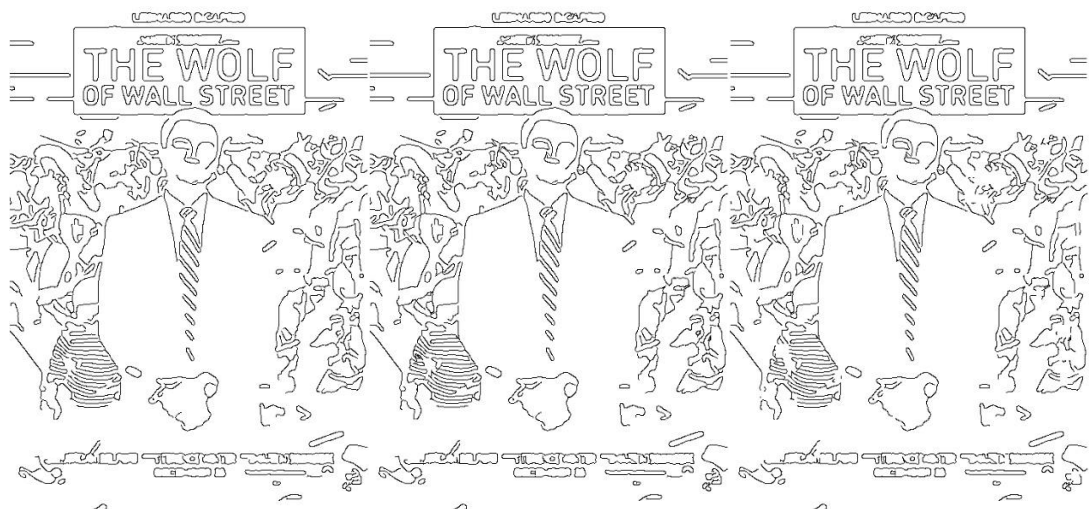


Distance设置为20生成的图像：

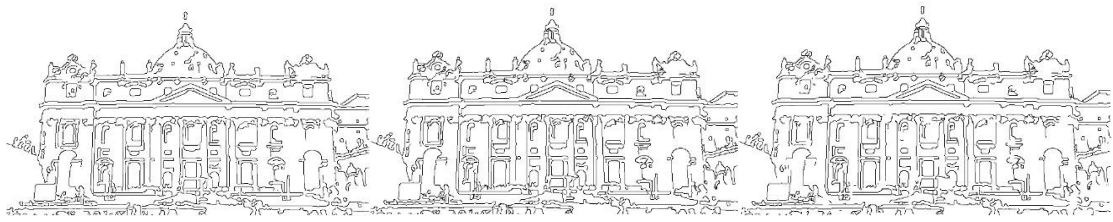


分析：可以看出将连直线的边缘点距离调大，连线的效果会越来越明显，但图像也会随之而来多了很多不必要的噪声点。

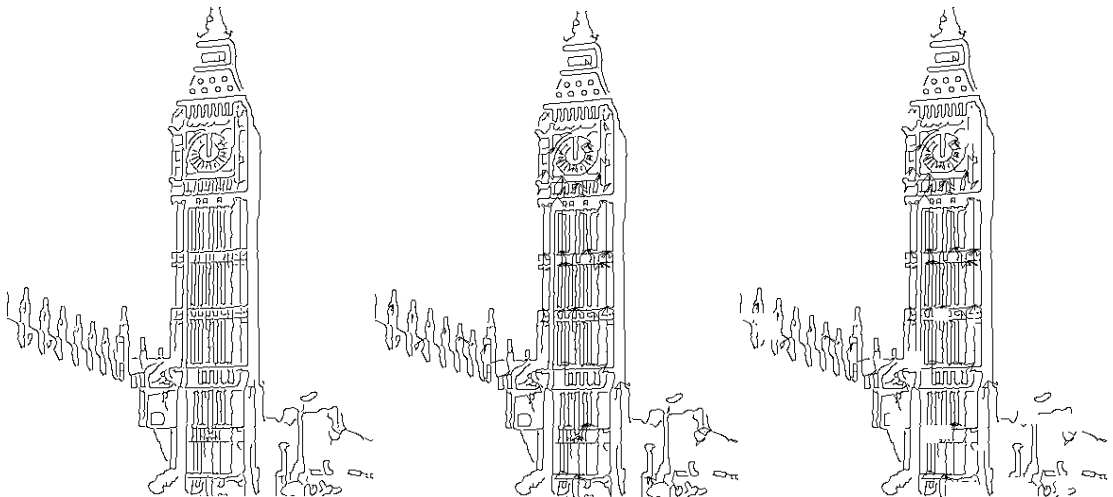
twows. jpg



stpietro. jpg



Bigben. jpg



接下来对sigma, tlow, thigh三个参数的调参只针对Lena图片

设置高斯标准差为1:



设置标准差为2:



设置标准差为3:



标准差为4:



分析：通过调节高斯标准差大小可以看出来参数越小，对图像噪声点滤波的效果就越差，因而在最终边缘图中噪声点越多；而如果标准差过大，模糊效果太强，边缘之间的很多细节又会给模糊掉，导致最后分割图的边缘细节又不是那么明显。

低阈值 t_{low} 为0.01



低阈值 t_{low} 为0.1



低阈值 t_{low} 为0.4



低阈值为0.65



分析：修改lowthreshold低阈值后，低阈值减小，将会增加很多噪声；而低阈值增大，又会丢失很多强边缘像素。这是因为对于弱边缘像素（在低阈值和高阈值之间），这些像素可以从真实边缘提取也可以是因噪声或颜色变化引起的。为了获得准确的结果，应该抑制由后者引起的弱边缘像素。因此需要调整低阈值参数。

高阈值为0.6



高阈值0.9



分析：高阈值减小，此时将会增加很多强边缘像素，因为大于选定这个阈值的像素点都将被确定为边缘；而高阈值增大，原来的强边缘像素大部分会转化为弱边缘像素，丢失一部分边缘像素点。这是因为被划分为强边缘的像素点已经被确定为边缘，因为它们是从图像中的真实边缘中提取出来的