实 验 报 告

实验 11：Canny 边缘检测

517030910374
郭嘉宋

目录：

注：在编程过程中我也遇到了很多困难和 Bug，还有很多思考了很久得出的结论与解释，以及一些需要注释的内容，都在下述“2. 实验过程”中，用加粗的 NOTE 加以了说明。

正文:

1. 实验准备

(1) 实验环境

本实验主要采用 Ubuntu 系统下的 python2.7 进行，使用 miniconda 环境，Ubuntu 系统安装在 VMware Workstation 提供的虚拟机中，同时使用 OpenCV、Numpy 等库对图像进行分析与后续处理。

(2) 实验目的

实验 11:

了解 Canny 算法的原理，学习边缘检测技术、进一步深入掌握 OpenCV 和 Numpy 等库对图像进行分析、了解获得更多的图像信息。

(3) 实验原理

实验 11:

1.边缘检测

图像的边缘指图像局部区域亮度变化显著的部分，该区域的灰度剖面一般可以看作是一个阶跃，即灰度值在很小的区域内急剧的变化。

实现图像的边缘检测，主要用离散化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，然后在图像中将这些位置的点连起来就构成了所谓的图像边缘。

彩色图像的每个像素由红色(R)、绿色(G)、蓝色(B)三个颜色分量构成，从而能够呈现多种色彩。一幅宽为 W，高为 H 的彩色图像在计算机中用一个 W×H×3 的矩阵存储，其中最后一个维度表示 RGB 颜色空间。

2.检测步骤

边缘检测是基于图像强度的一阶和二阶导，但导数通常对噪声较为敏感，因此需要去噪处理。

增强算法的目的是将图像灰度点邻域强度值有显著变化的点凸显出来。经过增强的图像，邻域中梯度值较大的很多，检测的目的是去除我们不需要的

点。

Canny算法原理

1. 灰度化

通常摄像机获取的是彩色图像，而检测的首要步骤是进行灰度化，以RGB格式彩图为例，一般的灰度化方法有两种：

方式1： Gray = (R + G + B)/3;

方式2：Gray=0.299R+0.587G+0.114B;（参数考虑到人眼的生理特点）

2. 高斯滤波

图像高斯滤波的实现可以用两个一维高斯核分别两次加权实现，也可以通过一个二维高斯核一次卷积实现。

离散化的一维高斯函数与二维高斯函数如下：确定参数就可以得到一维核向量与二维核向量：

$$K=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{x*x}{2\sigma*\sigma}}$$

离散一维高斯函数

$$K=\frac{1}{2\pi\sigma*\sigma}e^{-\frac{x*x+y*y}{2\sigma*\sigma}}$$

离散二维高斯函数

注：在求得高斯核后，要对整个核进行归一化处理

Canny算法原理

3. 灰一阶偏导的有限差分来计算梯度的幅值和方向

关于图像灰度值得梯度可使用一阶有限差分来进行近似，这样就可以得图像在x和y方向上偏导数的两个矩阵。常用的梯度算子有如下几种：

1) Roberts 算子 $s_x=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, s_y=\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}$

上式为其x和y方向偏导数计算模板，可用数学公式表达其每个点的梯度幅值为：

$$G[i,j]=|f[i,j]-f[i+1,j+1]|+|f[i+1,j]-f[i,j-1]|$$

2）Sobel算子 $s_x=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}, s_y=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$ $K=\begin{bmatrix}a_0 & a_1 & a_2\\ a_7 & [i,j] & a_3\\ a_6 & a_5 & a_4\end{bmatrix}$

上式三个矩阵分别为该算子的x向卷积模板、y向卷积模板以及待处理点的邻域点标记矩阵，据此可用数学公式表达其每个点的梯度幅值为：

$$G[i,j]=\sqrt{s_x^2+s_y^2}$$

$$s_x=(a_2+2a_3+a_4)-(a_0+2a_7+a_6)$$

$$s_y=(a_0+2a_1+a_2)-(a_6+2a_5+a_4)$$

Canny算法原理

3）Prewitt算子

和Sobel算子原理一样，在此仅给出其卷积模板。

$$s_x=\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}, s_y=\begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix}$$

4）Canny算法使用的算子

Canny算法中，所采用的卷积算子较为简单，为 $s_x=\begin{bmatrix}-1&1\\-1&1\end{bmatrix}, s_y=\begin{bmatrix}1&1\\-1&-1\end{bmatrix}$

其x向、y向的一阶偏导数矩阵，梯度幅值以及梯度方向的数学表达式为：

$$P[i,j]=\left(f[i,j+1]-f[i,j]+f[i+1,j+1]-f[i+1,j]\right)/2$$
$$Q[i,j]=\left(f[i,j]-f[i+1,j]+f[i,j+1]-f[i+1,j+1]\right)/2$$
$$M[i,j]=\sqrt{P[i,j]^2+Q[i,j]^2}$$
$$\theta[i,j]=\arctan\left(Q[i,j]/p[i,j]\right)$$

求出这几个矩阵后，就可以进行下一步的检测过程。

Canny算法原理

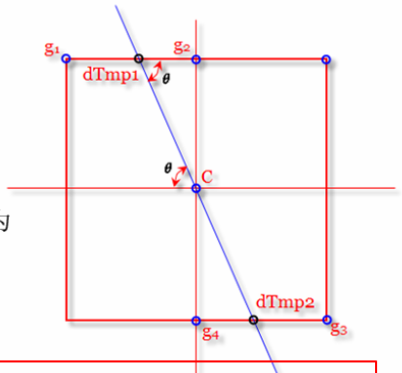
4. 对梯度幅值进行非极大值抑制

图像梯度幅值矩阵中的元素值越大，说明图像中该点的梯度值越大。在Canny算法中，非极大值抑制是进行边缘检测的重要步骤，**是指寻找像素点局部最大值，将非极大值点所对应的灰度值置为0**，从而可以剔除掉一大部分非边缘点。

右图为判断像素点C的灰度值在邻域内是否为最大的原理。蓝色线条方向为C点的梯度方向，C点局部的最大值则分布在这条线上。

即除C点外，还需要判定dTmp1和dTmp2两点灰度值。若C点灰度值小于两点中任一个，则C点不是局部极大值，因而可以排除C点为边缘点。

需要判别两点是因为之前求梯度忽略了梯度正负，取了平方，否则只需要寻找梯度方向。



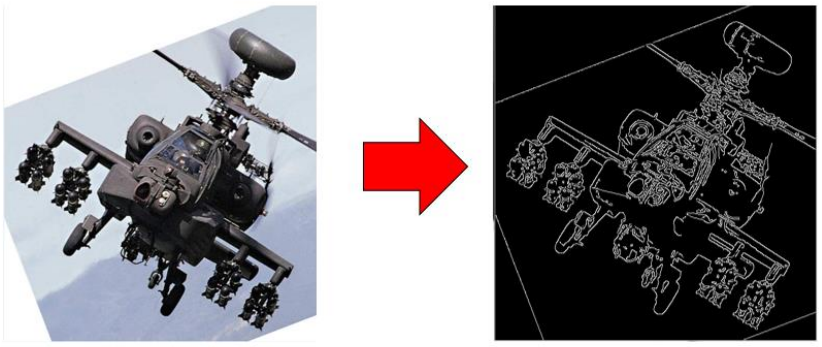
注意：我们只能得到C点邻域的8个点，而dTmp1和dTmp2并不在其中，因而需要根据g1和g2对dTmp1进行插值，根据g3和g4对dTmp2进行插值，这要用到其梯度方向，即上文Canny算法中要求解梯度方向矩阵的原因。

Canny算法原理

5. 双阈值算法检测和连接边缘

Canny算法中减少假边缘数量的方法是采用双阈值法。选择两个阈值，根据高阈值得到一个边缘图像，这样一个图像含有很少的假边缘，但是由于阈值较高，产生的图像边缘可能不闭合，为解决此问题采用了另外一个低阈值。

在高阈值图像中把边缘链接成轮廓，当到达轮廓的端点时，该算法会在断点邻域点中寻找满足低阈值的点，再根据此点收集新的边缘，直到整个图像边缘闭合。



Canny 算法效果示意图

2. 实验过程

1.1 实验步骤

首先用 cv2 读入图片，存入 myimg 变量进行操作，第一次实验我先使用了 opencv 自带的高斯函数以（3,3）的范围对图像进行卷积模糊，方便之后进行边缘检测，这样可以消除高斯噪声，如下图所示：

```
import cv2
import numpy

myimg=cv2.imread("dataset/1.jpg",cv2.IMREAD_GRAYSCALE)
myimg=cv2.GaussianBlur(myimg,(3,3),0)# after Gauss.
```

之后对每一个像素使用 Canny 算子求出其梯度，我将原先的灰度图转换成了梯度图，如下图所示：

```
num_row=len(myimg)
num_column=len(myimg[0])
angle_list=[]

for k1 in range(0,num_row-1):
    angle_per_row=[]

    for k2 in range(0,num_column-1):
        gradient_row=float(int(myimg[k1][k2+1])-int(myimg[k1][k2])
                           +int(myimg[k1+1][k2+1])-int(myimg[k1+1][k2]))/2
        gradient_column=float(int(myimg[k1][k2])-int(myimg[k1+1][k2])
                              +int(myimg[k1][k2+1])-int(myimg[k1+1][k2+1]))/2

        gradient=int(numpy.sqrt(numpy.square(gradient_row)+numpy.square(gradient_column)))

        myimg[k1][k2] = gradient
```

现在的 myimg 中的每一个元素都是原先对应点的梯度值。

NOTE：查阅资料后得知，一定要将每个点的灰度值先转换成 int 类型再进行操作，因为灰度值为 byte 类型，如果不进行类型转换就会出现如下错误。

```
/home/guo/PycharmProjects/lab11/lab11.py:24: RuntimeWarning: overflow encountered in ubyte_scalars
  gradient_row=float(myimg[k1][k2+1]-myimg[k1][k2]+myimg[k1+1][k2+1]-myimg[k1+1][k2])/2
/home/guo/PycharmProjects/lab11/lab11.py:25: RuntimeWarning: overflow encountered in ubyte_scalars
  gradient_column=float(myimg[k1][k2]-myimg[k1+1][k2]+myimg[k1][k2+1]-myimg[k1+1][k2+1])/2
```

接下来我将每一个元素梯度的方向的正切值，即为 tan 值存入了 angle_list 中，方便之后极大值处理时使用，如下图所示。

```
        if gradient_row:
            tan_angle=gradient_column/gradient_row
        else:
            tan_angle='0'

        angle_per_row.append(tan_angle)
    angle_list.append(angle_per_row)

# turn to gradient picture
```

NOTE：字符类型的‘0’为一个标记，标记正切值为正无穷的点，即梯度角为 90 度的点方便之后读取，防止分母为 0。

之后对所有元素进行非极大值抑制，使用梯度角对应比例计算两点的梯度值，与之比较即可，若为最大值则保留，否则删除。删除信息记录在 transfer_list 中，如下图所示：

```
old_img=myimg.copy()
transfer_list=[]

transfer_per_row=[]
for k2 in range(0,num_column-1):
    transfer_per_row.append(0)
transfer_list.append(transfer_per_row)



for k1 in range(1,num_row-1):
    transfer_per_row = []
    transfer_per_row.append(0)
    for k2 in range(1,num_column-1):
```

```
        if angle_list[k1][k2]=='0':
            if myimg[k1][k2]>myimg[k1-1][k2] and myimg[k1][k2]>myimg[k1+1][k2]:
                transfer_per_row.append(1)
            else:
                transfer_per_row.append(0)

        else:
            if angle_list[k1][k2]<1 and angle_list[k1][k2]>0:
                tmp1=angle_list[k1][k2]*myimg[k1][k2-1]+\
                     (1-angle_list[k1][k2])*myimg[k1-1][k2-1]
                tmp2=angle_list[k1][k2]*myimg[k1][k2+1]+\
                     (1-angle_list[k1][k2])*myimg[k1+1][k2+1]

                if myimg[k1][k2] >tmp1 and myimg[k1][k2] > tmp2:
                    transfer_per_row.append(1)
                else:
```

```
                    transfer_per_row.append(0)

            if angle_list[k1][k2] >= 1:
                tmp_angle_tan=(1/angle_list[k1][k2])
                tmp1 = tmp_angle_tan * myimg[k1-1][k2] + \
                       (1 - tmp_angle_tan) * myimg[k1 - 1][k2 - 1]
                tmp2 = tmp_angle_tan * myimg[k1+1][k2] + \
                       (1 - tmp_angle_tan) * myimg[k1 + 1][k2 + 1]
                if myimg[k1][k2] >tmp1 and myimg[k1][k2] > tmp2:
                    transfer_per_row.append(1)
                else:
                    transfer_per_row.append(0)
```

```
            if angle_list[k1][k2] <= -1 :
                tmp_angle_tan = abs(1 / angle_list[k1][k2])
                tmp1 = tmp_angle_tan * myimg[k1 - 1][k2] +\
                       (1 - tmp_angle_tan) * myimg[k1 - 1][k2 + 1]
                tmp2 = tmp_angle_tan * myimg[k1 + 1][k2] + \
                       (1 - tmp_angle_tan) * myimg[k1 + 1][k2 - 1]
                if myimg[k1][k2] > tmp1 and myimg[k1][k2] > tmp2:
                    transfer_per_row.append(1)
                else:
                    transfer_per_row.append(0)



            if angle_list[k1][k2] > -1 and angle_list[k1][k2] <= 0:
                tmp_angle_tan = abs(angle_list[k1][k2])
                tmp1 = tmp_angle_tan * myimg[k1][k2+1] + \
                       (1 - tmp_angle_tan) * myimg[k1 - 1][k2]
                tmp2 = tmp_angle_tan * myimg[k1][k2-1] + \
```

```
                    (1 - tmp_angle_tan) * myimg[k1 + 1][k2]
                if myimg[k1][k2] > tmp1 and myimg[k1][k2] > tmp2:
                    transfer_per_row.append(1)
                else:
                    transfer_per_row.append(0)

    transfer_list.append(transfer_per_row)


for k1 in range(0,num_column):

    myimg[num_row-1][k1]=0

for k2 in range(0,num_row):
    myimg[k2][num_column-1]=0
```

NOTE：需要对第一列和最后一列、第一行和最后一行补全，一行并不影响图片整体效果，因此补为 0，如上图所示。

```
myimg_high=myimg.copy()
myimg_low=myimg.copy()
for k1 in range(0,num_row-1):
    for k2 in range(0,num_column-1):

        if transfer_list[k1][k2] and myimg_high[k1][k2]>=10:
            myimg_high[k1][k2]=255
        else:
            myimg_high[k1][k2]=0

for k1 in range(0,num_row-1):
    for k2 in range(0,num_column-1):

        if transfer_list[k1][k2] and myimg_low[k1][k2]>=2:
            myimg_low[k1][k2]=255
        else:
            myimg_low[k1][k2]=0
```

之后采用高低阈值对边缘进行过滤，对于 img1 如上图，我使用 10 和 2 作为两个阈值数值，使用两个阈值各做一张图，如上图所示。

```
for k1 in range(0,num_row-1):
    for k2 in range(0,num_column-1):
        connect(myimg_new,k1,k2,myimg_low)
```

```
def connect(myimg_new,k1,k2,myimg_low):
# while k1<num_row-1 and k2<num_column-1:
    if k1>=num_row-1 or k2>=num_column-1 or k1<0 or k2<0:
        return

    if myimg_new[k1][k2]:
        if myimg_low[k1+1][k2]:
            myimg_new[k1+1][k2]=255
            # found.append([k1,k2])
            # to_find.append([k1+1,k2])
            # return k1+1,k2
            # connect(myimg_new,k1+1,k2,myimg_low)
        if myimg_low[k1-1][k2]:
            myimg_new[k1-1][k2]=255
            # found.append([k1, k2])
            # to_find.append([k1 - 1, k2])
```

```
            # connect(myimg_new, k1 - 1, k2,myimg_low)
            # return k1 -1, k2
        if myimg_low[k1][k2+1]:
            myimg_new[k1][k2+1]=255
            # found.append([k1, k2])
            # to_find.append([k1 , k2+1])
            # connect(myimg_new, k1, k2+1,myimg_low)
            # return k1 , k2+1
        if myimg_low[k1][k2-1]:
            myimg_new[k1][k2-1]=255
            # found.append([k1, k2])
            # to_find.append([k1 , k2-1])
            # connect(myimg_new, k1, k2 - 1, myimg_low)
            # return k1,k2-1
```

```
        if myimg_low[k1+1][k2+1]:
            myimg_new[k1+1][k2+1]=255
            # found.append([k1, k2])
            # to_find.append([k1+1, k2 - 1])
            # connect(myimg_new, k1 + 1, k2+1,myimg_low)
            # return k1 + 1, k2+1
        if myimg_low[k1+1][k2-1]:
            myimg_new[k1+1][k2-1]=255
            # found.append([k1, k2])
            # to_find.append([k1+1, k2 - 1])
            # connect(myimg_new, k1+1, k2 - 1, myimg_low)
            # return k1 + 1, k2-1
        if myimg_low[k1-1][k2+1]:
            myimg_new[k1 - 1][k2+1] = 255
            # found.append([k1, k2])
            # to_find.append([k1-1, k2 + 1])
```

```
            # return k1 - 1, k2+1
        if myimg_low[k1-1][k2-1]:
            myimg_new[k1-1][k2-1]=255
            # found.append([k1, k2])
            # to_find.append([k1-1, k2 - 1])
            # connect(myimg_new, k1-1, k2 - 1, myimg_low)
            # return k1-1,k2-1
```

如上图所示，最后即将高阈值图按照低阈值连接起来即可。我使用的算法是在高阈值中寻找周围 8 个方向的点，以梯度方向连线，若在低阈值中找到此点即

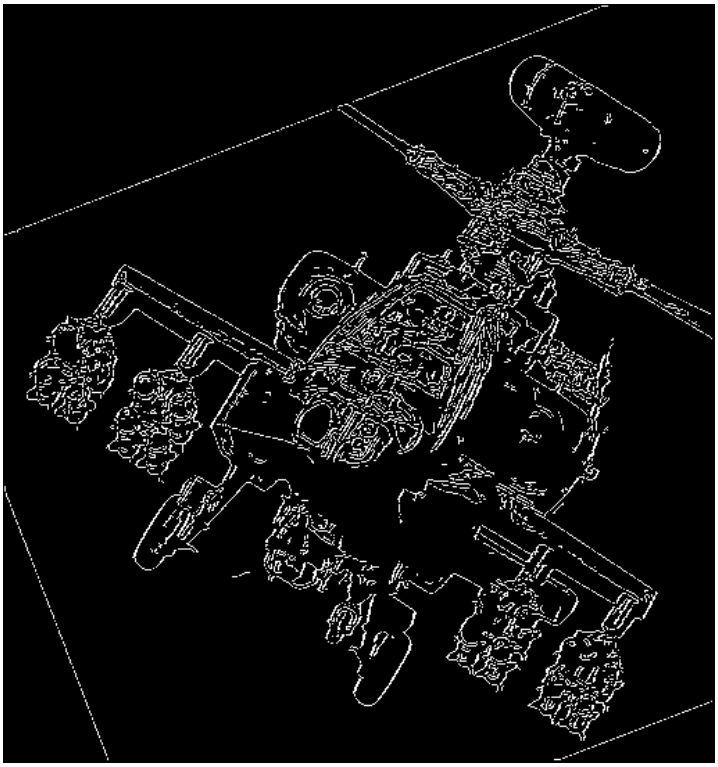
为边缘线，没有找到则不连。

NOTE：梯度方向保证了连线方向的正确，同时如源码中注释内容，我最开始使用了深度优先的递归算法连接边缘，发现复杂度实在太高，运算太慢，现在的算法简单很多最终效果也相差无几。

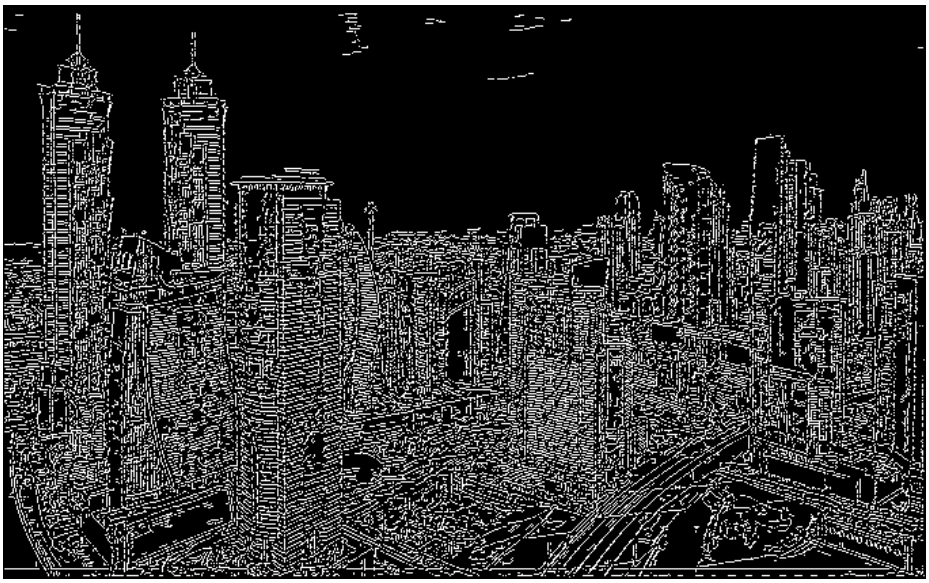
1.2 实验结果：

Img1：

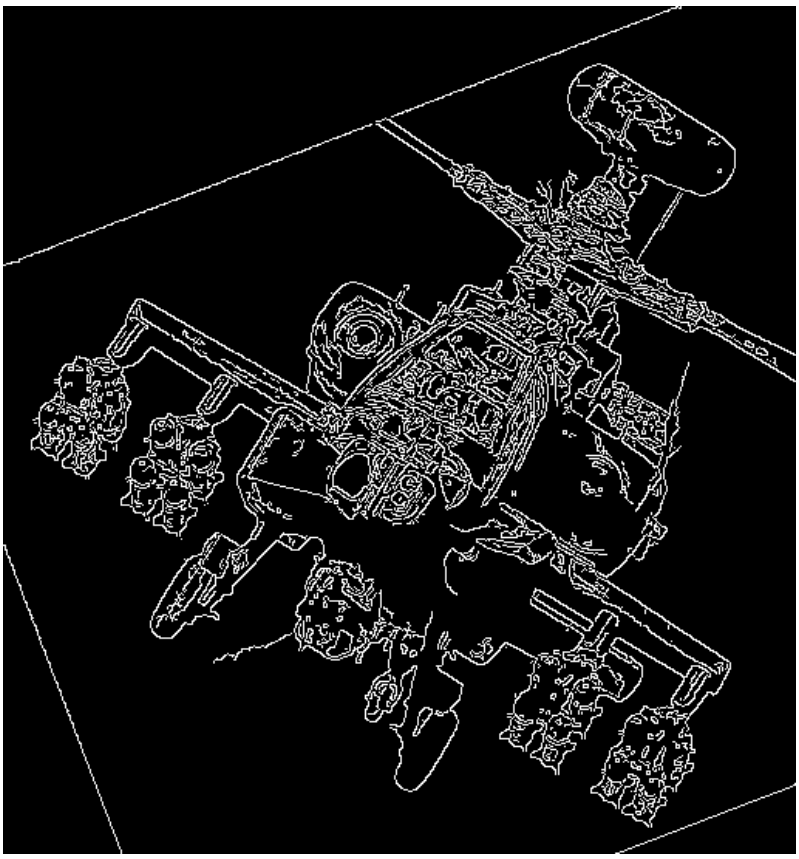
Img2：

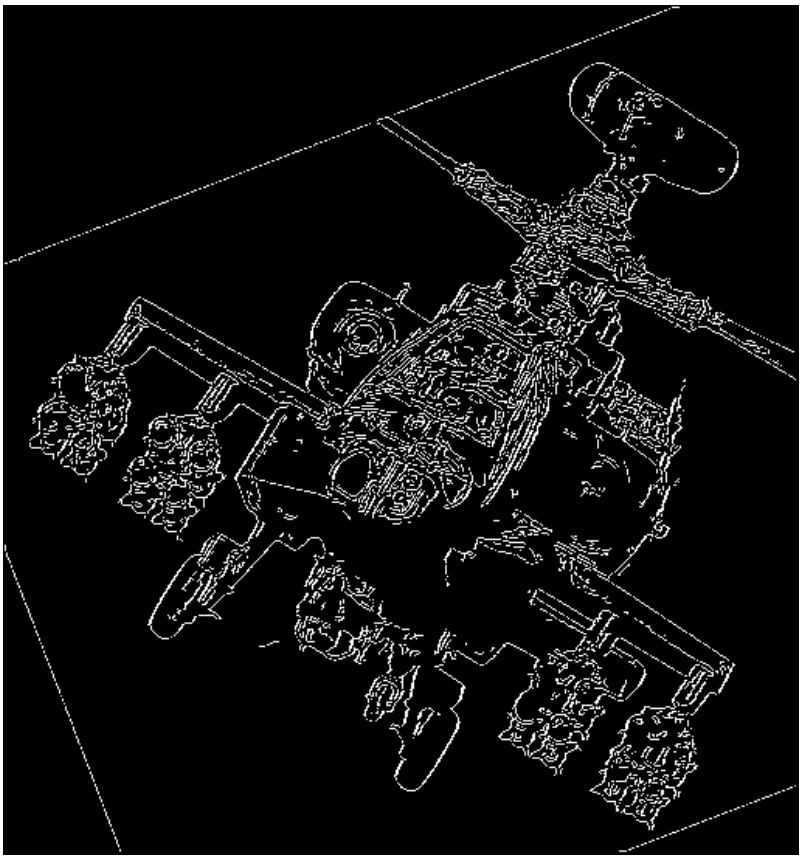


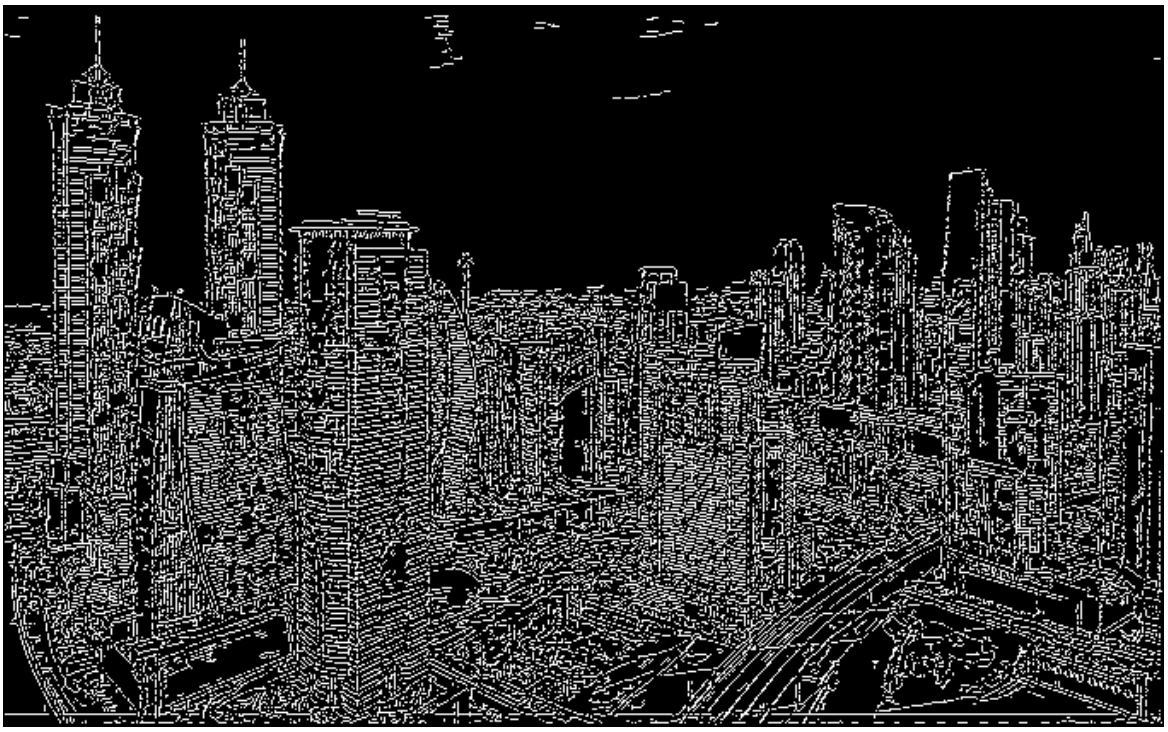
Img3:



最后将最终结果与自带 Canny 库中函数进行比较，如下图所示:





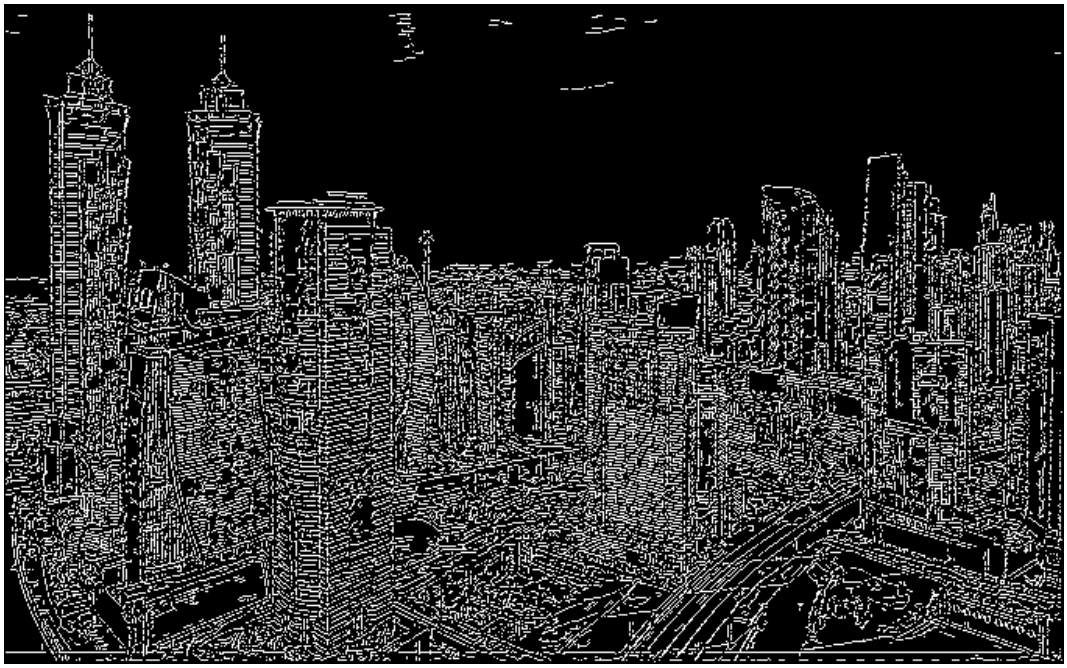


NOTE： 由上图可见，比较下来结果十分相似，但仍有改进的地方，比如连接算法有时会漏掉孤立的分界线孤立点，需要进一步优化，总体实验成功。

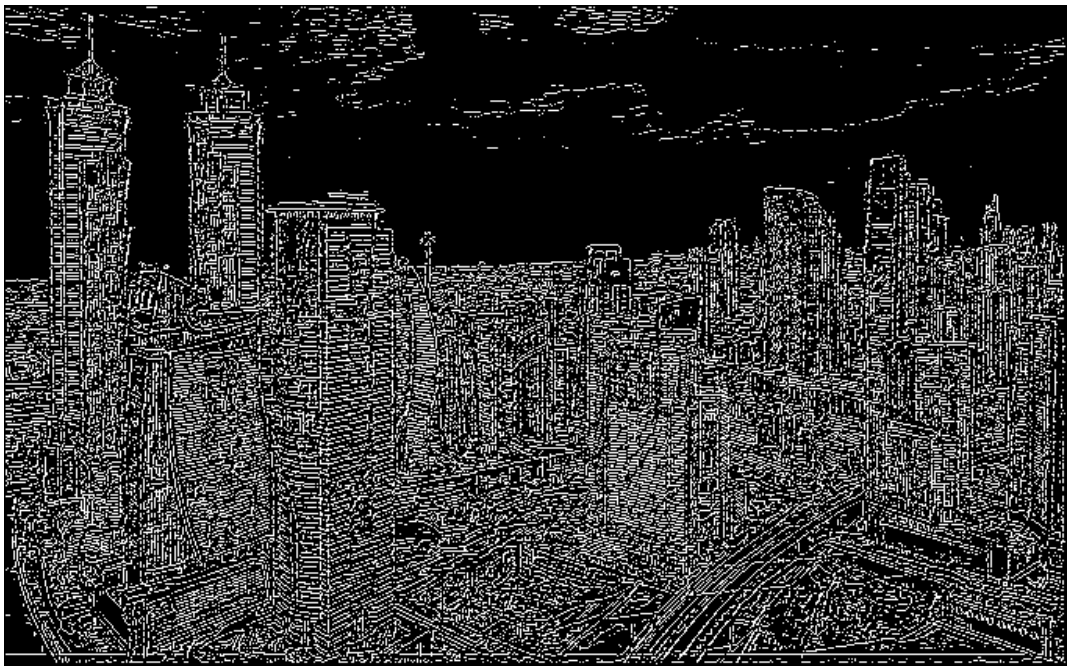
3. 实验总结

在此次试验中我也注意到其实梯度图的信息是最完全的，而处理梯度图的后序工作则看出不同程序的性能好坏，如 IMG3 的梯度图，其实保留了完整的信息，IMG3 完整的 5 幅图如下图所示。

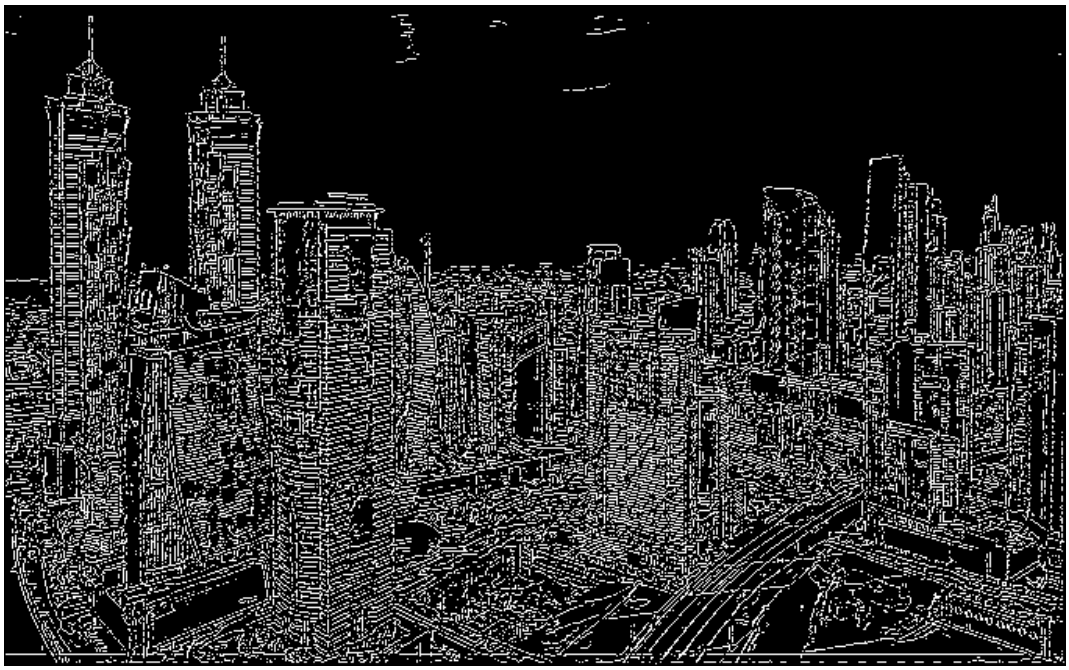
高阈值图:



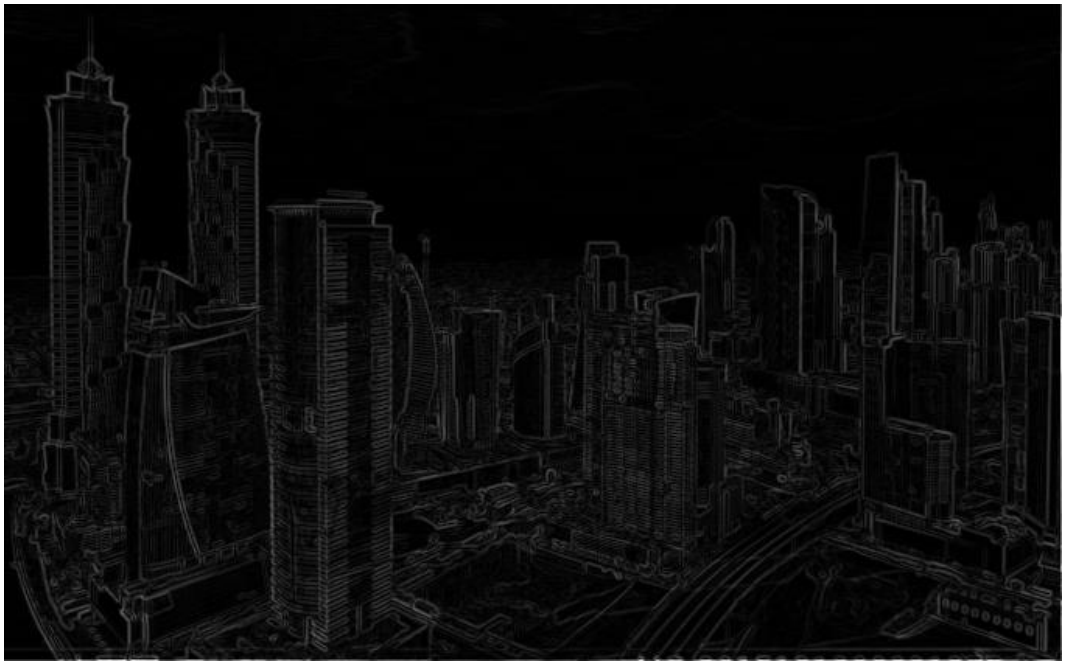
低阈值图:



最终图:



梯度图:



Canny 检测图:

因此本次实验我的程序和 cv2 库十分接近，实验较为成功，但还有改进的地方，主要体现在连接算法，同样的梯度图，cv2 库使得噪点几乎没有，而且连接线十分平滑，我还需要进一步优化连接算法达到更好的效果。

此次试验让我对 OpenCV 的了解加深了，上学期计算导论使用过 OpenCV 制作过人脸识别，对 OpenCV 的有一定的认识，这次试验让我学习到了图片边缘检测技术，这也是一切图像识别的基础，希望在以后的学习过程中继续了解图像处理技术。

注：在编程过程中我也遇到了很多困难和 Bug，还有很多思考了很久得出的结论与解释，以及一些需要注释的内容，都在下述 “2. 实验过程” 中，用加粗的 NOTE 加以了说明。源码见附带 lab11.py 文件即可。

继续努力！

o(*≧▽≦)ツ！

517030910374

郭嘉宋