# 电类工程学导论 C 实验报告 11

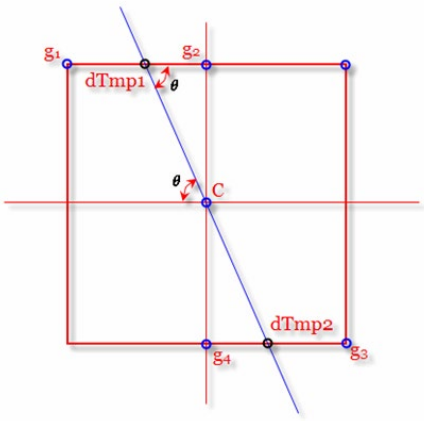
518030910406　郑思榕

一、实验准备

1. 实验环境介绍
   1) 环境：在 windows 系统中进行本次实验
   2) 语言： python 3.7
   3) 工具：本实验主要使用了开源计算机视觉库 openCV、处理大型多维矩阵的 python 库 numpy
2. 实验目的

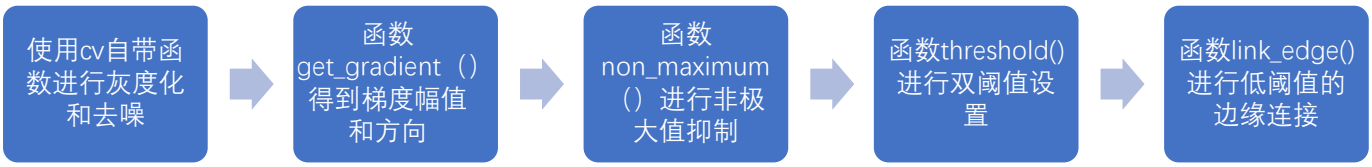
   对 dataset 文件夹中的图片进行 Canny 边缘检测，并与 OpenCV 库中自带 Canny 检测结果进行对比
3. 实验原理

   本实验通过 python 库 openCV 和 numpy 实现 Canny 边缘检测算法。Canny 算法的原理由 5 个部分组成：
   1) 灰度化：检测的首要步骤是将彩色图片灰度化。将 RGB 格式的彩图进行灰度化的方法主要有两种：Gray = (R+B+B)/3 和 Gray = 0.299R+0.587G+0.114B
   2) 高斯滤波：Canny 检测算法需要用到对像素点求导，导数通常对噪声较为敏感，因此需要用高斯滤波去噪处理。
   3) 灰一阶偏导的有限差分来计算梯度的幅值和方向：边缘的像素点是像素值变化最大的点，即梯度幅值最大的点。本实验使用 Sobel 算子进行像素点梯度幅值和梯度方向的计算。
   4) 对梯度幅值进行非极大值抑制：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。如下图所示，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，然后将 c 点梯度幅值和 dTmp1 和 dTmp2 的梯度幅值进行比较。若 c 点的梯度幅值不是最大，则·c 点不是边缘点。
   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。先根据高阈值得到一个边缘图像，这个图像的边缘往往不会闭合，所以当到达边缘的端点时，算法会在端点的邻域中寻找满足低阈值的点，再连接到边缘图像上。

二、实验过程




1. 使用 cv 自带函数进行灰度化和去噪

这部分不是算法的重点，使用 cv2 库自带的函数即可，代码如下。

```
for i in [1,2,3]:
    image = cv2.imread('{}.jpg'.format(i),cv2.IMREAD_GRAYSCALE)
    gause_img = cv2.GaussianBlur(image,(3,3),0)
```

2. 函数 get_gradient（）得到梯度幅值和方向

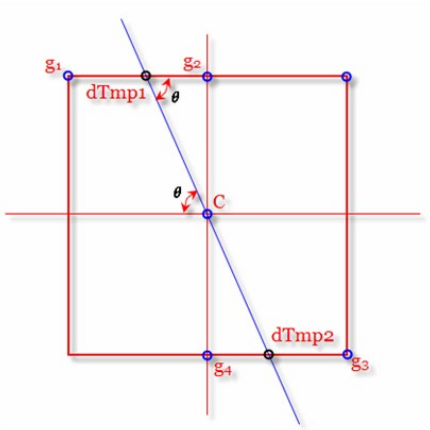
```
def get_gradient(img):

    img_sobelx = cv2.Sobel(img,cv2.CV_16S,1,0,3)
    gradient_x = cv2.convertScaleAbs(img_sobelx)

    img_sobely = cv2.Sobel(img,cv2.CV_16S,0,1,3)
    gradient_y = cv2.convertScaleAbs(img_sobely)

    gradient=np.hypot(gradient_x,gradient_y)
    gradient=np.asarray(gradient,dtype=np.uint8)

    np.seterr(divide='ignore', invalid='ignore')
    direction = np.divide(img_sobely,img_sobelx)
    direction[np.isnan(direction)]=0.0
    direction[np.isinf(direction)]=0.0

    return (gradient,direction)
```

使用 cv2 库自带的函数 cv2.Sobel()函数进行梯度值的计算。sobel 函数的第一个参数是需要进行卷积的矩阵；第二个参数是输出图像的深度，这里用 CV_16 防止数据出现符号和数值溢出；第三四个参数分别是是否沿 xy 轴进行 Sobel 算子求导，1 为进行求导，0 为不进行求导；最后一个参数是 Soble 算子的大小，由于本实验使用的 Soble 算子是 3*3 矩阵，所以是 3。注意 Sobel 函数求导后会有负值和>255 的值，因而需先使用 convertScaleAbs()函数转换成 16 位有符号的数据类型，即 cv2.CV_16S，再转为 unit8 类型。

然后使用 np.hypot()函数进行梯度幅值的计算。hypot 函数本意为给定直角三角形的两条直角边长度，返回斜边的长度，与梯度幅值的定义相同。然后用 np.divide()函数求梯度方向角的正切值。为了防止报警告，用 np.seterr()去掉除数等于 0 的警告，函数中的参数为 divide='ignore', invalid='ignore'。并用 isnan 和 isinf 函数将 direction 矩阵里的不是数字和过小的值设为 0.0。

3. 函数 non_maximum（）进行非极大值抑制

这部分是 canny 检测算法的重点之一。根据下图的所示，使用插值法计算 c 点




梯度方向所在直线与 8 像素矩形的两交点 dtmp1 和 dtmp2。由于先前已经计算了每个点的梯度方向 tanθ=y/x，得到了梯度方向矩阵 direction。由上图可知，有四种形况：tanθ>=1，1>tanθ>=0,0>tanθ>=-1 和 tanθ<=-1，四种情况中 dtemp1 和 dtemp2 的计算方式各不相同。所以对梯度幅值矩阵 gradient 里的每一个数进行下列四个判断，得到 dtemp1 和 dtemp2，若 c 点幅值比其中任何一个小，就置为 0。

```
if (direction[i][j]>1):
    dtemp1 = (1-1/direction[i][j])*gradient[i-1][j]+1/direction[i][j]*gradient[i-1][j+1]
    dtemp2 = (1-1/direction[i][j])*gradient[i+1][j]+1/direction[i][j]*gradient[i+1][j-1]
elif(0<=direction[i][j]<=1):
    dtemp1 = (1-direction[i][j])*gradient[i][j+1]+direction[i][j]*gradient[i-1][j+1]
    dtemp2 = (1-direction[i][j])*gradient[i][j-1]+direction[i][j]*gradient[i+1][j-1]
elif(0>direction[i][j]>-1):
    dtemp1 = (1+direction[i][j])*gradient[i][j+1]-direction[i][j]*gradient[i+1][j+1]
    dtemp2 = (1+direction[i][j])*gradient[i][j-1]-direction[i][j]*gradient[i-1][j-1]
elif(direction[i][j]<=-1):
    dtemp1 = (1+1/direction[i][j])*gradient[i+1][j]-1/direction[i][j]*gradient[i+1][j+1]
    dtemp2 = (1+1/direction[i][j])*gradient[i-1][j]-1/direction[i][j]*gradient[i-1][j-1]

if not (dtemp1<=gradient[i][j] and dtemp2<=gradient[i][j]):
    gradient[i][j] = 0
```

4. 函数 threshold()进行双阈值设置

```
def threshold(gradient,lowradio=0.2,highradio=0.4):
    high = gradient.max()*highradio
    low = gradient.max()*lowradio

    weak = np.int32(70)
    strong = np.int32(255)

    strong_i,strong_j = np.where(gradient >= high)
    weak_i,weak_j = np.where((gradient>=low) & (gradient<high))
    zero_i,zero_j = np.where((gradient<low))

    gradient[strong_i,strong_j] = strong
    gradient[weak_i,weak_j] = weak
    gradient[zero_i,zero_j] = 0

    return gradient,weak,strong
```

上面非极大值抑制的方法已经排除掉一部分非边缘的像素点了，但还有很大部分非边缘像素点存在。使用双阈值法进行分类，设定高低阈值分别为 high 和 low，对灰度大于 high 的像素点的灰度设为 255，介于 low 和 high 之间的像素点灰度设为 70，低于 low 的像素点设为灰度 0，代码如上图所示。

5. 函数 link_edge()进行低阈值的边缘连接

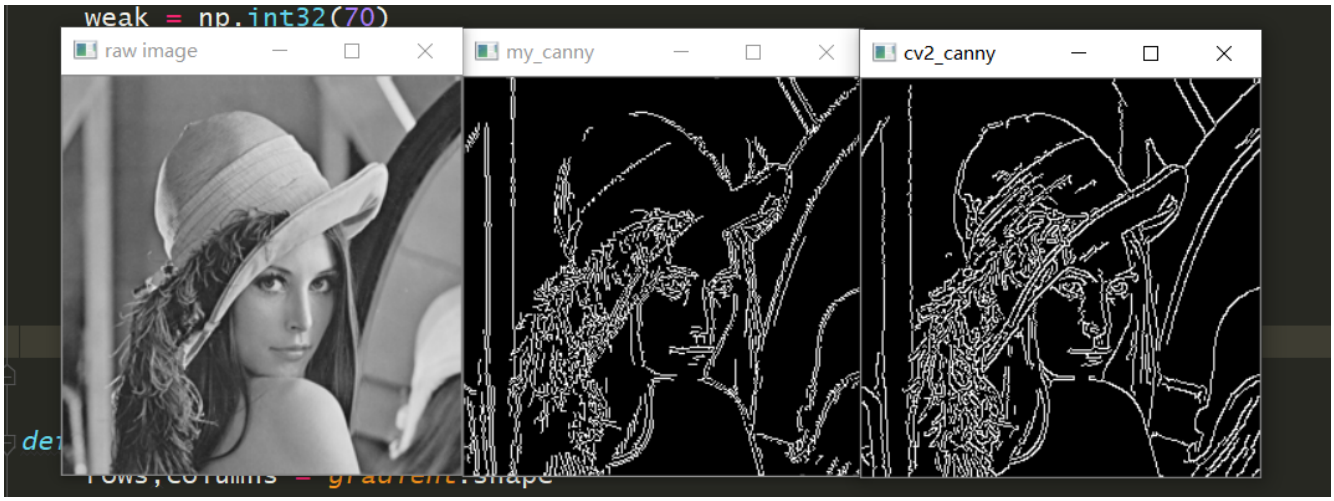
双阈值法得到了含两种灰度边缘的图像，遍历整张图像，在灰度为 weak 的像素点的周围 8 个点进行判断，若这 8 个点中存在灰度为 strong 的像素点，则此处就是端点，所以应将该 weak 点的灰度设置为 strong。反之，若周围八个点不存在灰度为 strong 的像素点，则该点不是端点，将该 weak 点的灰度设置为 0。代码如下：

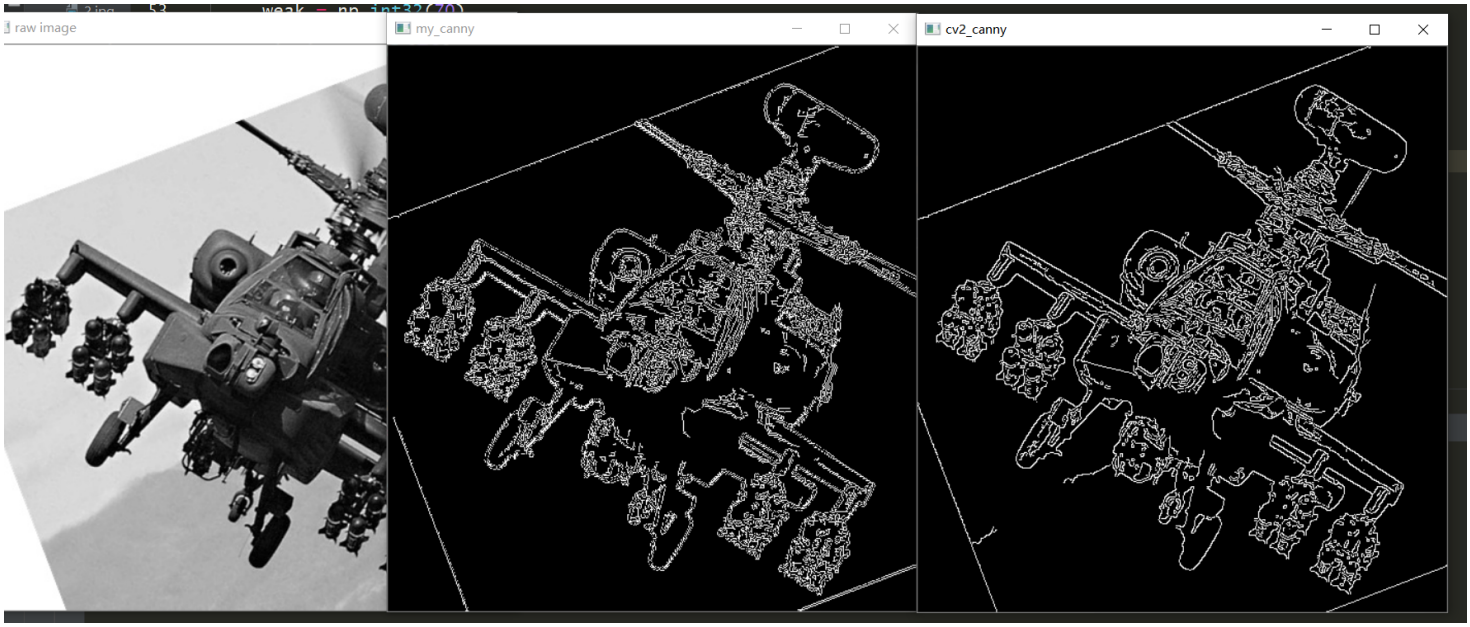
```
def link_edge(gradient,weak,strong):
    rows,columns = gradient.shape

    for i in range(1,rows-1):
        for j in range(1,columns-1):
            if (gradient[i][j]==weak):
                try:
                    if (gradient[i-1][j-1]==strong or gradient[i-1][j]==strong or gradient[i-1][j+1]
                    ==strong
                    or gradient[i][j-1]==strong or gradient[i][j+1]==strong or gradient[i+1][j-1]
                    ==strong
                    or gradient[i+1][j]==strong or gradient[i+1][j+1]==strong):
                        gradient[i][j]=strong
                    else:
                        gradient[i][j] = 0
                except IndexError:
                    pass
    return gradient
```

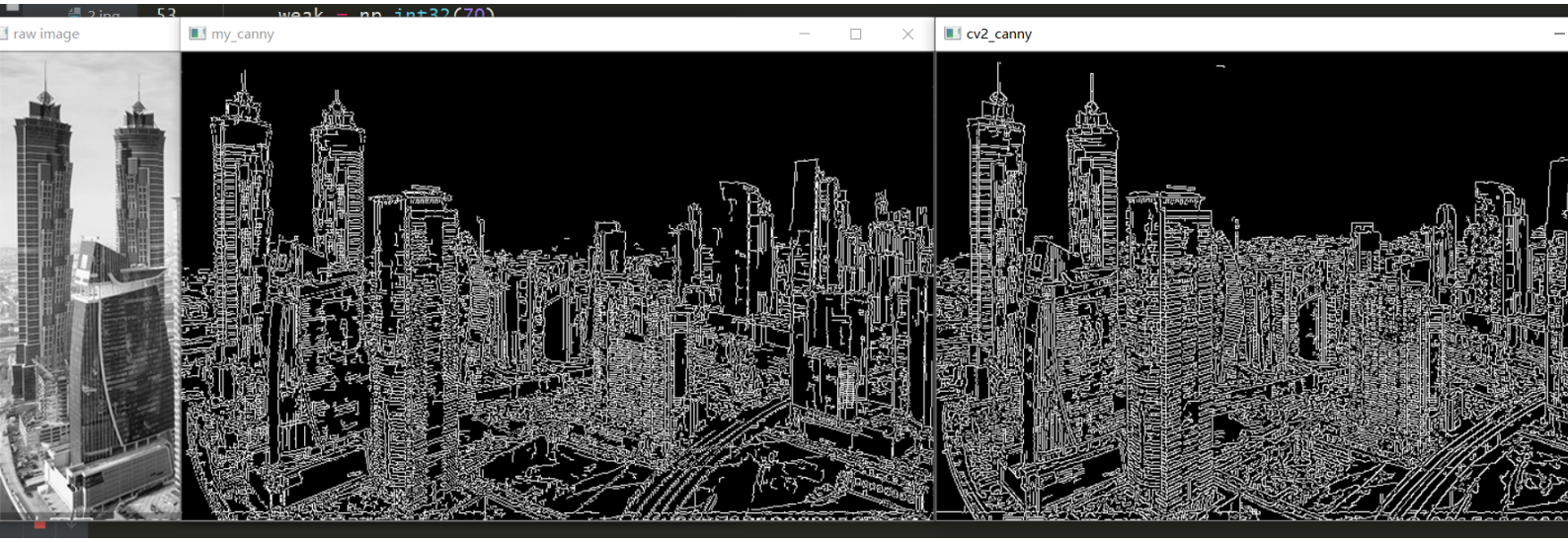
但这样进行边缘连接有点问题，就是边缘的会加宽会“发毛”，而且遍历图像是从左上到右下进行遍历的，如果一开始某个点周围没有 strong 点，但经过后来其他点的灰度值判定修改，该 weak 点周围可能又有 strong 点了，而对该 weak 点的判断已经结束，就会出现没有把边缘线连接起来的情况。我想的优化方法是加入对该点的梯度方向的判定，但实际代码中有些 bug 没有能够完成，这里就不放出来了。

6. 结果展示

从左到右分别是 raw_image，my_canny 和 cv2_canny，意即灰度图、自己实现的 canny 算法得到的边缘和 cv2.Canny()函数得到的边缘






三、实验总结

1. 实验概述

本实验根据 canny 边缘检测算法的定义，使用 opecv 和 numpy 库函数实现了像素点梯度幅值方向矩阵的计算、非极大值抑制、双阈值法和边缘连接等 canny 算法的关键步骤，实现了基于 Sobel 算子的 canny 边缘检测算法。

2. 实验心得与算法讨论

本实验我收获良多，主要有以下几点：

1) 学会了 canny 边缘检测算法的定义，学会了如何自己实现 canny 算法
2) 学会了 cv2 库中的一些函数的使用，如函数 cv2.Sobel()，cv2. convertScaleAbs()等等
3) 学会了 numpy 库中的一些函数的使用，如 np.hypot()，np.asarray()，np.where()等等

但我自己实现的 canny 边缘检测算法与 cv2 库中自带的 canny 算法的出

来的边缘图像有差异，我想主要是在边缘连接时，判断 weak 点周围 8 个点有没有灰度为 strong 的算法不太好。因为如果一开始某个点周围没有 strong 点，但经过后来其他点的灰度值判定修改，该 weak 点周围可能又有 strong 点了，而对该 weak 点的判断已经结束，就会出现没有把边缘线连接起来的情况。但我尝试优化了以下，没有写出令人满意的代码。另外计算灰度梯度幅值和方向时使用的是 Sobel 算子而不是 Canny 算子可能是另一个原因。

3. 实验创新点
   1) 对梯度方向矩阵出现 inf 和 NaN 的情况进行了优化，使用下列两行代码将该位置的方向正切值设为 0。

```
direction[np.isnan(direction)]=0.0
direction[np.isinf(direction)]=0.0
```

   2) 使用 np.where()函数来找梯度矩阵中介于量阈值间、大于高阈值、低于低阈值的点，避免了使用两层 for 循环，代码看起来更简洁。

```
strong_i,strong_j = np.where(gradient >= high)
weak_i,weak_j = np.where((gradient>=low) & (gradient<high))
zero_i,zero_j = np.where((gradient<low))
```

最后，衷心感谢实验中老师和各位助教的帮助!