浙江大学

本科实验报告

Canny Edge Detection

课程名称：计算机视觉

姓　　名：秦雨扬

学　　院：竺可桢学院

专　　业：自动化（控制）

学　　号：3210104591

电　　话：13588789194

邮　　件：qinyuyang2003@zju.edu.cn

指导老师：潘纲

2023 年 11 月 25 日

Contents

List of Figures

浙江大学实验报告

专业：自动化（控制）
姓名：秦雨扬
学号：3210104591
日期：2023 年 11 月 25 日
地点：玉泉曹光彪西-103

课程名称：计算机视觉　　指导老师：潘纲　　助教：周健均
实验名称：Canny Edge Detection　　实验类型：综合　　成绩：

I 功能简述及运行说明

1. 功能简述

在本实验中，主要实现了以下几个功能

- 将给定的图片灰度化
- 调用 *cv2.Canny* 函数，生成边缘作为对比
- 构造了 *myCanny(img,threshold1,threshold2,sigma=3)* 函数，对于输入的黑白图片，实现了 Canny 边缘检测算法 [1]:
    1) 根据 σ 进行高斯模糊
    2) 计算图形梯度
    3) 非最大抑制
    4) 根据双阈值得到强连接图与弱连接图
    5) 基于 BFS 实现边缘连接
- 设计了两种方法，实现彩色边缘提取
- 展示图像

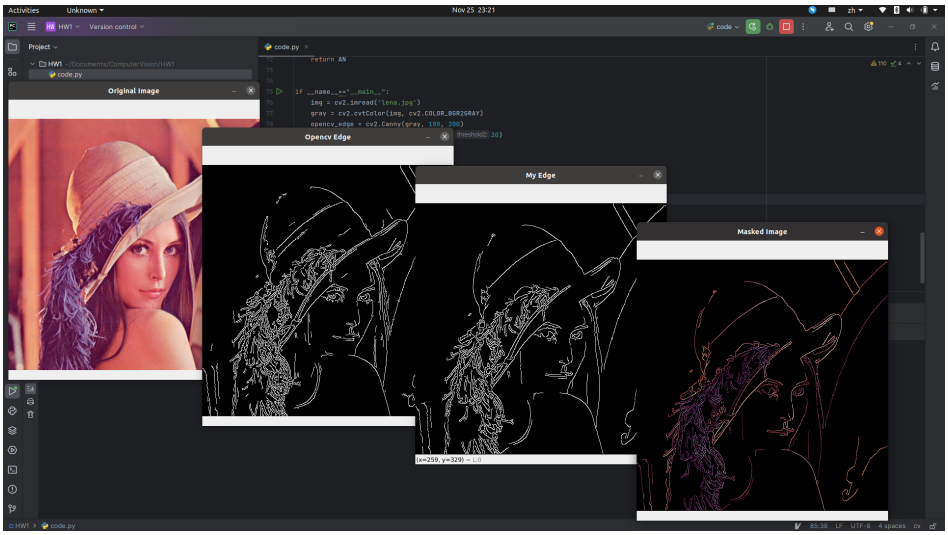


Figure 1: 程序运行截图

2. 运行说明

- 本实验只有一个 python 文件，直接使用 *python code.py* 即可运行，运行截图如 **Figure**1
- 修改 *cv2.imread()* 中的路径可以更换测试图片
- 修改 *mmyCanny(img,threshold1,threshold2,sigma=3)* 中的参数，可分别调整处理图片、低阈值、高阈值与高斯模糊的 σ, 这里的梯度阈值的值代表相对图像中最大梯度的百分比
- 程序将展示四个窗口 *Original Image,Opencv Edge,My Edge,Masked Image*，分别为原始图片、调用 *cv2.Canny* 函数生成的边缘、使用自定义函数 *myCanny* 生成的边缘与自定义函数对应的彩色边缘提取

II 开发与运行环境

实验使用 python 语言, 测试系统为 *Ubuntu 20.04.6 LTS*, 测试环境为 *Anaconda 23.7.3* 下的 *python 3.8.18* ，其它使用的包分别为

1) opencv-python 4.8.1.78
2) numpy 1.24.4
3) python 自带的 copy,Queue 与 math 包

III 算法原理与具体实现

1. 灰度化图片

灰度化的公式为:

$$Gray = 0.299 * R + 0.587 * G + 0.114 * B$$

在实际代码中，采用 openCV 的自带函数 *cv2.cvtColor*

```
77  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

2. 用高斯滤波器平滑图像

$$S[i,j] = G[i,j;\sigma] * I[i,j]$$

在实际代码中，采用 OpenCV 的自带函数 *cv2.GaussianBlur*

```
11      img = cv2.GaussianBlur(img, (sigma, sigma), cv2.BORDER_DEFAULT)
```

3. 用一阶偏导有限差分计算梯度幅值和方向

使用差商近似梯度，使用以下公式计算梯度幅值和方向:

$$\begin{aligned} G_x[i,j] &= (S[i,j+1] - S[i,j] + S[i+1,j+1] - S[i+1,j])/2 \\ G_y[i,j] &= (S[i,j] - S[i+1,j] + S[i,j+1] - S[i+1,j+1])/2 \\ M[i,j]i,j &= \sqrt{G_x[i,j]^2 + G_y[i,j]^2} \\ \theta[i,j] &= arctan(G_y[i,j]/G_x[i,j]) \end{aligned} \tag{1}$$

```
    m,n=img.shape
    Gx=np.zeros([m-1,n-1])
    Gy=np.zeros([m-1,n-1])
    M=np.zeros([m-1,n-1])
    thetas=np.zeros([m-1,n-1])
    for i in range(m-1):
        for j in range(n-1):
            Gx[i][j]=(img[i,j+1]-img[i][j]+img[i+1][j+1]-img[i+1][j])/2
            Gy[i][j]=(img[i][j]-img[i+1][j]+img[i][j+1]-img[i+1][j+1])/2
            M[i][j]=(Gx[i][j]**2+Gy[i][j]**2)**0.5
            thetas[i][j]=math.atan2(Gy[i][j],Gx[i][j])
    M=M/np.max(M)*100
```

4. 对梯度幅值进行非极大值抑制（NMS）

非极大值抑制考虑的是对于一条边缘，其上每个点的梯度与线的方向垂直，所以在梯度方向上只保留最大值则可以使边缘只剩下一条。

这里只考虑以当前点为中心的 3*3 的窗口，因此我们可以如 **Figure** 2所示将梯度方向离散化为四个方向，若 M[i,j] 不比沿梯度线方向上的两个相邻点幅值大，则 N[i,j]=0

而在实际编程中，直接将 M 复制给 N(注意一定要 deepcopy)，然后通过将 N[i][j] 与其两个梯度方向上的判断结果（若梯度方向上更大则为 0，否则为 1）相乘实现对不满足条件的 N 自动赋 0。此外，通过 check 函数来判断越界，减小了主体程序的判断压力，简化了程序。

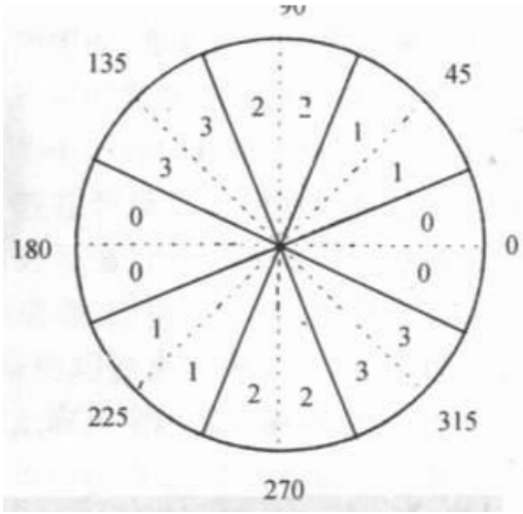


Figure 2: 非极大值抑制梯度方向离散化示意图

```
    def check(i,j):
        if(i<0 or i>=m-1):
            return -1
        if(j<0 or j>=n-1):
            return -1
        return M[i][j]
    thetas=thetas/math.pi*180
```

```
    N = copy.deepcopy(M)
    for i in range(m-1):
        for j in range(n-1):
            theta=thetas[i][j]
            if -157.5<theta<=-112.5 or 22.5<theta<=67.5:
                N[i][j] = (check(i - 1, j + 1) < M[i][j]) * N[i][j]
                N[i][j] = (check(i + 1, j - 1) < M[i][j]) * N[i][j]
            if -22.5<theta<=22.5 or theta<=-157.5 or theta >157.5:
                N[i][j] = (check(i , j -1) < M[i][j]) * N[i][j]
                N[i][j] = (check(i , j +1) < M[i][j]) * N[i][j]
            if -67.5<theta<=-22.5 or 112.5<theta<=157.5:
                N[i][j] = (check(i + 1, j +1) < M[i][j]) * N[i][j]
                N[i][j] = (check(i - 1, j -1) < M[i][j]) * N[i][j]
            if -112.5<theta<=-67.5 or 67.5<theta<=112.5:
                N[i][j] = (check(i+1 , j ) < M[i][j]) * N[i][j]
                N[i][j] = (check(i-1 , j ) < M[i][j]) * N[i][j]
```

5. 用双阈值算法检测和连接边缘

通过 threshold1,threshold2 生成了强连接图（N>threshold2）与弱连接图 (threshold1<threshold2), 算法需要把弱连接图中周围有强连接图上点的点不断地加入强连接图中，从而实现将强连接图中的一些断开的线相连。

在具体实现过程中，采用了广度优先搜索 (Breadth First Search，BFS)，通过维护队列 q 与强连接图 AN，实现了上述算法。

```
    def checkN(i, j):
        if (i < 0 or i >= m - 1):
            return -1
        if (j < 0 or j >= n - 1):
            return -1
        return N[i][j]
    AN=np.zeros([m,n],np.uint8)
    q=Queue()
    for i in range(m-1):
        for j in range(n-1):
            if(N[i][j]>threshold2):
                q.put([i,j])
                AN[i][j]=1
    # return AN*255
    while(q.empty() is False):
        x,y=q.get()
        for i in [x-1,x,x+1]:
            for j in [y-1,y,y+1]:
                if(checkN(i,j)>threshold1):
                    if(AN[i][j]==0):
                        q.put([i,j])
                        AN[i][j]=1
```

6. 提取彩色边缘

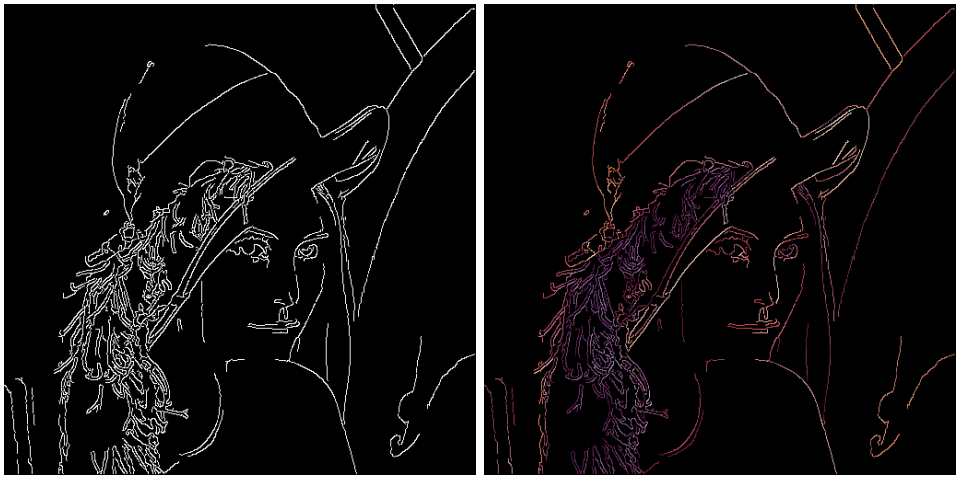
这里实现了两种方法，一种是使用 numpy 的乘法，分别对 RGB 三层相乘，从而实现提取彩色的效果。另一个是使用 OpenCV 的自带函数 *cv2.bitwise_and*, 将 img 自己与自己位和则得到它自己，但是通过 Mask 参数实现了对彩色的提取。这里应当注意 AN 在生成过程中应该指定 *up.uint8*, 因为该函数的 Mask 需要 8 位灰度图。

```
masked=copy.deepcopy(img)
for i in range(3):
    masked[:,:,i]*=myedge
# masked = cv2.bitwise_and(img, img,mask=myedge)
```

IV 实验结果与分析

首先测试了 lena 图, 随意选了 Opencv 的阈值并调整自定义函数阈值使两者效果接近, 结果如 **Figure** 5

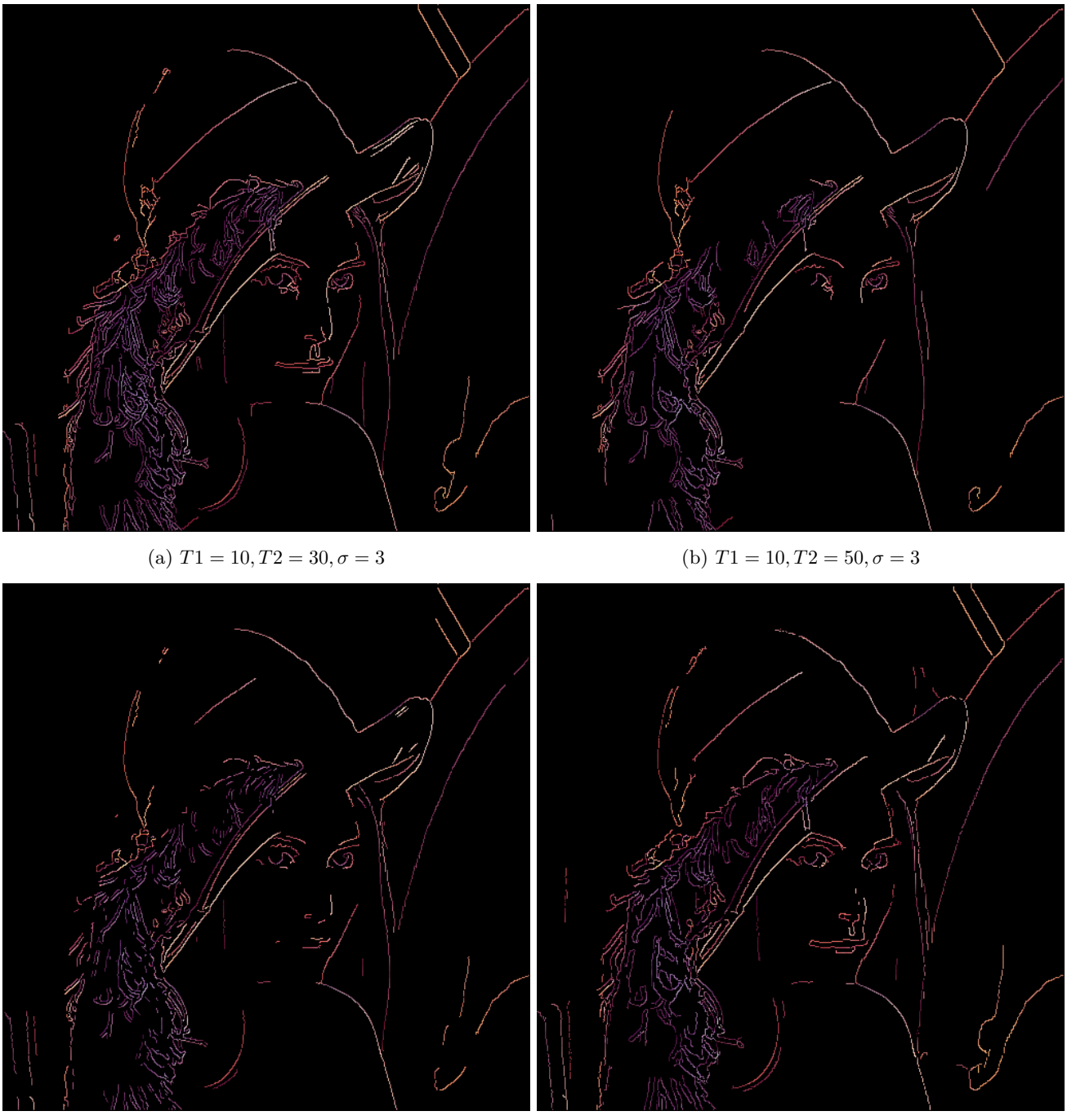
(a) 原始图像 (b) OpenCV 生成边缘，T1=100,T2=200



(c) 自定义函数生成边缘，T1=10,T2=30 (d) 自定义函数对应彩色边缘

Figure 3: lena 图

然后改变 $threshold1, threshold2, \sigma$ 观察效果。可以观察到增大 threshold2 会减少主干边缘，增大 threshold1 会减少一些边缘的细节且线的连贯性会下降，而增大 σ 则会减少细节。



(a) $T1 = 10, T2 = 30, \sigma = 3$

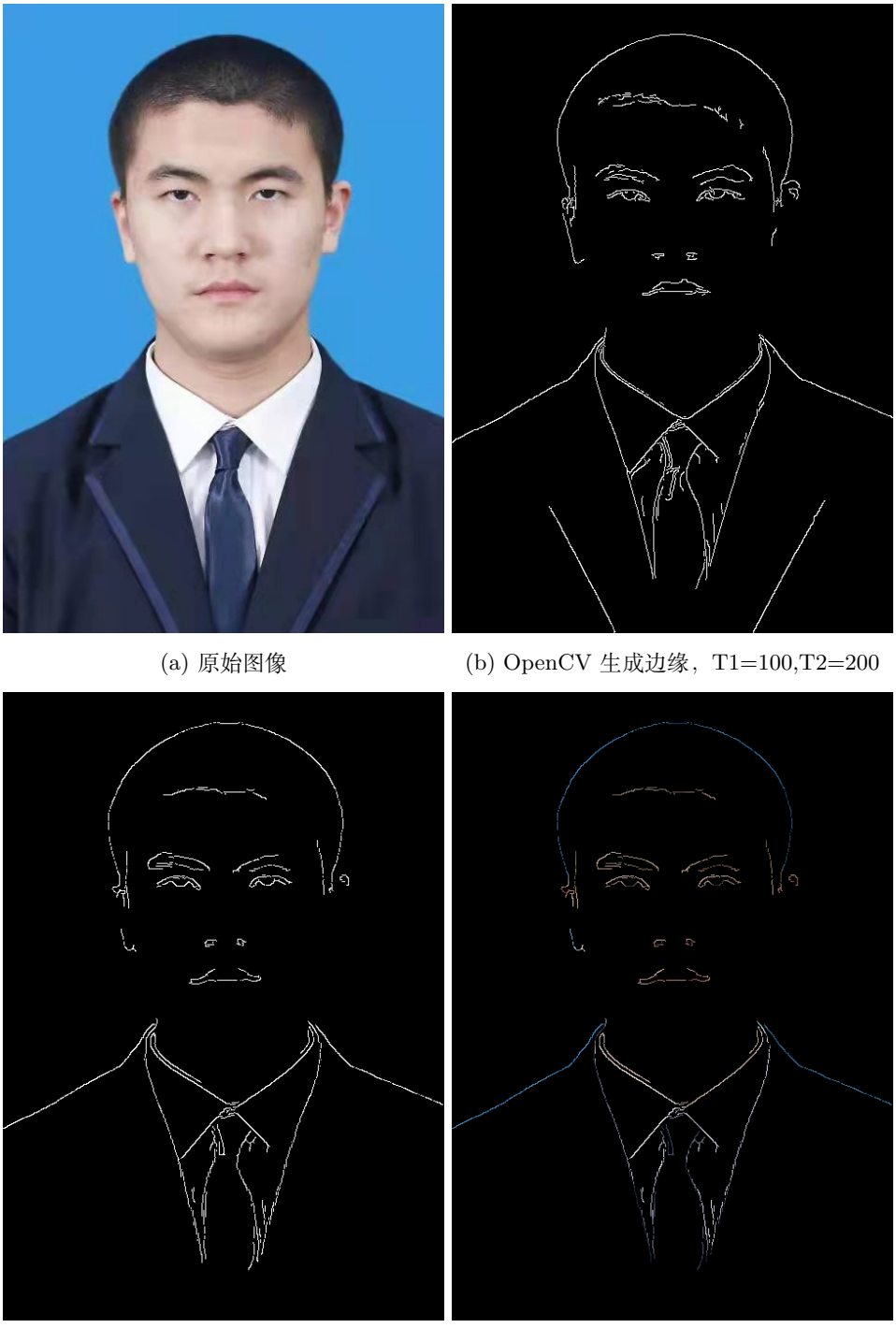
(b) $T1 = 10, T2 = 50, \sigma = 3$

(c) $T1 = 25, T2 = 30, \sigma = 3$

(d) $T1 = 10, T2 = 30, \sigma = 5$

Figure 4: lena 图参数测试

接着，放入个人照片进行测试



(a) 原始图像 (b) OpenCV 生成边缘，T1=100,T2=200

(c) 自定义函数生成边缘，T1=10,T2=25 (d) 自定义函数对应彩色边缘

Figure 5: 个人照片

V 结论与心得体会

1. 结论

Canny 边缘检测算子虽然看起来简单，但是其具有很强的实用价值，而在真正的部署过程中才发现它其实并没有那么简单。而且随着对它的实践才明白它的一个个环节是紧紧相扣的。为了只保留一条边缘，考虑到边缘方向与梯度方向垂直，所以可以在梯度方向上进行抑制。但是如果没有高斯滤波，其求出的梯度方向则具有很大的不确定性，难以满足假定的与边缘方向垂直，所以高斯滤波非常关键。而如果没有 Non-max Suppression，双阈值则会带来一块面积的填充，与预期的边缘检测相去深远。而正是这几个简单步骤的环环相扣，成就了 Canny 简约而不简单的传奇。

2. 心得体会

- 在实现 NMS 的过程中，越界的判断尤为头疼，后来想到了使用乘法，并使用 check 函数，大大减小的程序设计的难度。且一开始忘记 deepcopy 造成了一定的错误。
- 而也正是 check 函数让我 debug 了好久。原来是我在 BFS 的过程中用了原来的 check，直接返回了 M[i][j] 而不是 N[i][j]，相当于对于弱连接图没有做 NMS，造成甚至的边缘非常粗，让我百思不得其解。
- 在最后的彩色提取中，才明白数据类型的重要性，Mask 需要 8 位灰度图，RGB 在乘的过程中也表示无法从 float64 转化为 uint8，这是我日常在使用 python 的过程中常常忽略的一点。

VI Appendix

```python
import copy
from queue import Queue
import cv2
import numpy as np
import math


def myCanny(img,threshold1,threshold2,sigma=3):
    assert threshold1>=0 and threshold2<=255
    #Gaussian Filter
    img = cv2.GaussianBlur(img, (sigma, sigma), cv2.BORDER_DEFAULT)
    img=np.array(img,dtype=int)
    #Calculate Gradient's Amplitude and Direction
    m,n=img.shape
    Gx=np.zeros([m-1,n-1])
    Gy=np.zeros([m-1,n-1])
    M=np.zeros([m-1,n-1])
    thetas=np.zeros([m-1,n-1])
    for i in range(m-1):
        for j in range(n-1):
            Gx[i][j]=(img[i,j+1]-img[i][j]+img[i+1][j+1]-img[i+1][j])/2
            Gy[i][j]=(img[i][j]-img[i+1][j]+img[i][j+1]-img[i+1][j+1])/2
            M[i][j]=(Gx[i][j]**2+Gy[i][j]**2)**0.5
            thetas[i][j]=math.atan2(Gy[i][j],Gx[i][j])
    M=M/np.max(M)*100
    def check(i,j):
        if(i<0 or i>=m-1):
```

```
            return -1
        if(j<0 or j>=n-1):
            return -1
        return M[i][j]
    thetas=thetas/math.pi*180
    N = copy.deepcopy(M)
    for i in range(m-1):
        for j in range(n-1):
            theta=thetas[i][j]
            if -157.5<theta<=-112.5 or 22.5<theta<=67.5:
                N[i][j] = (check(i - 1, j + 1) < M[i][j]) * N[i][j]
                N[i][j] = (check(i + 1, j - 1) < M[i][j]) * N[i][j]
            if -22.5<theta<=22.5 or theta<=-157.5 or theta >157.5:
                N[i][j] = (check(i , j -1) < M[i][j]) * N[i][j]
                N[i][j] = (check(i , j +1) < M[i][j]) * N[i][j]
            if -67.5<theta<=-22.5 or 112.5<theta<=157.5:
                N[i][j] = (check(i + 1, j +1) < M[i][j]) * N[i][j]
                N[i][j] = (check(i - 1, j -1) < M[i][j]) * N[i][j]
            if -112.5<theta<=-67.5 or 67.5<theta<=112.5:
                N[i][j] = (check(i+1 , j ) < M[i][j]) * N[i][j]
                N[i][j] = (check(i-1 , j ) < M[i][j]) * N[i][j]

    def checkN(i, j):
        if (i < 0 or i >= m - 1):
            return -1
        if (j < 0 or j >= n - 1):
            return -1
        return N[i][j]
    AN=np.zeros([m,n],np.uint8)
    q=Queue()
    for i in range(m-1):
        for j in range(n-1):
            if(N[i][j]>threshold2):
                q.put([i,j])
                AN[i][j]=1
    # return AN*255
    while(q.empty() is False):
        x,y=q.get()
        for i in [x-1,x,x+1]:
            for j in [y-1,y,y+1]:
                if(checkN(i,j)>threshold1):
                    if(AN[i][j]==0):
                        q.put([i,j])
                        AN[i][j]=1
    return AN


if __name__=="__main__":
    img = cv2.imread('lena.jpg')
```

```
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    opencv_edge = cv2.Canny(gray, 100, 200)
    myedge = myCanny(gray, 10, 30)
    # myedge = myedge*img
    masked=copy.deepcopy(img)
    for i in range(3):
        masked[:,:,i]*=myedge
    # masked = cv2.bitwise_and(img, img,mask=myedge)
    cv2.imshow('Original Image', img)
    cv2.imshow('Opencv Edge', opencv_edge)
    cv2.imshow('My Edge', myedge*255)
    cv2.imshow('Masked Image', masked)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

参考文献

[1] J. Canny, "A computational approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679–698, 1986.