电类工程导论(C 类)实验 11 报告

贾萧松 516030910548

一、实验概述

在 Lab11，我们学习并自己完成了一遍 Canny 算法的流程，在每一步中都存在不少选择和值得优化的地方，每种选择各有利弊，也需要我们根据实际要求来做出决策。

二、实验环境

Ubuntu14.04+Pythton2.7+opencv2.4.11+numpy1.13.3+matplotlib2.1.0

三、实验内容

1. 图像平滑滤波（去噪）

Canny 算法在读取完图像的灰度值后，要进行的第一步就是去除噪声，也就是进行模糊处理，滤波器原理就是根据待滤波的像素点及其邻域点的灰度值按照一定的参数规则进行加权平均。这样可以有效滤去理想图像中叠加的高频噪声。

1.1 高斯滤波（Gaussian filter）

a. 原理：

从名字可以看出是按照高斯二维分布进行加权。

离散化公式为

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

经过学习，我了解到：高斯函数具有可分离性，即使用二维矩阵变换得到的效果也可以通过在水平方向进行一维高斯矩阵变换加上竖直方向的一维高斯矩阵变换得到。从计算的角度来看，这是一项有用的特性，因为这样只需要 O(n*M*n)+O(m*M*N)次计算，而二维不可分的矩阵则需要 O(m*n*M*n)次计算，其中，m,n 为高斯矩阵的维数，M,N 为二维图像的维数。

b. 代码实现

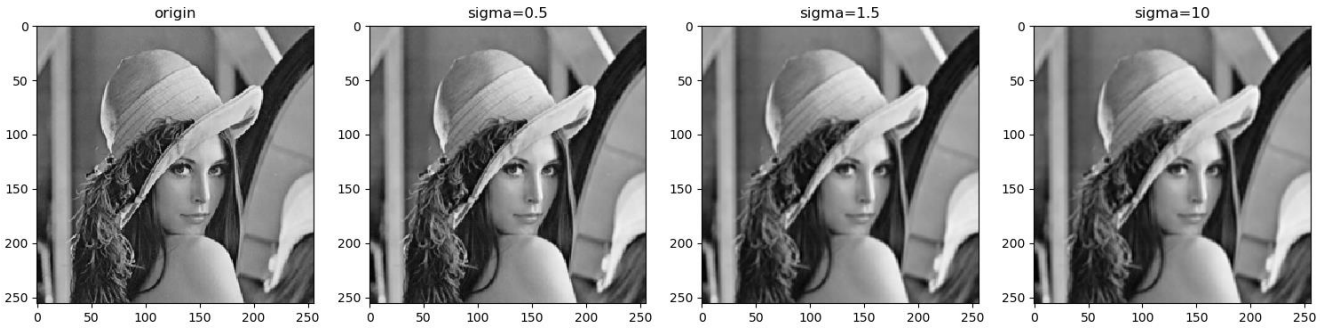
opencv 中提供了这一函数，速度比手动实现的要快：

```
img = cv2.GaussianBLur(img, (3,3), 0)
```

参数中: img 为原图矩阵， (3,3)为高斯核大小，第三个为标准差，值越大，图像越模糊(平滑)。

c. 效果

下面分别是原图， σ =0.5, 1.5, 10 时的图像



可以看出，标准差值越大图像越模糊(平滑)。

1.2 双边滤波（Bilateral filter）

我们已经知道高斯滤波器是求中心点邻近区域像素的高斯加权平均值。这种高斯滤波器只考虑像素之间的空间关系，而不会考虑像素值之间的关系（像素的相似度）。所以这种方法不会考虑一个像素是否位于边界。因此边界也会别模糊掉，而这正不是我们想要。

通过学习，我了解到双边滤波在同时使用空间高斯权重和灰度值相似性高斯权重。空间高斯函数确保只有邻近区域的像素对中心点有影响，灰度值相似性高斯函数确保只与中心像素灰度值相近的才会被用来做模糊运算。所以这种方法会确保边界不会被模糊掉，因为边界处的灰度值变化比较大。

a. 原理

设原图为 f(x,y),(x,y)为像素的坐标，双边滤波后(x,y)点的像素值变为

$$\hat{f}(x,y)=\frac{\sum_{(i,j)\in Sx,y} w(i,j)g(i,j)}{\sum_{(i,j\in Sx,y)} w(i,j)}$$

公式中 Sx,y 表示中心点（x,y）的（2N+1）*（2N+1）大小的领域。实际上，公式右边就是中心像素点邻域内像素亮度值的加权平均。

权值 W 由两部分组成：

$$w_s(i,j)=e^{\frac{|i-x|^2+|j-y|^2}{2\delta_s^2}} \qquad w_r(i,j)=e^{\frac{|g(i,j)-g(x,y)|^2}{2\delta_r^2}}$$

其中 W(i,j)=Ws(i,j)*Wr(i,j)

双边滤波器受 3 个参数的控制：滤波器半宽 N、参数 δs 和 δr。N 越大，平滑作用越强； δs 和 δr 分别控制着空间邻近度因子 Ws 和亮度像似度因子 Wr 的衰减程度。

b. 代码实现

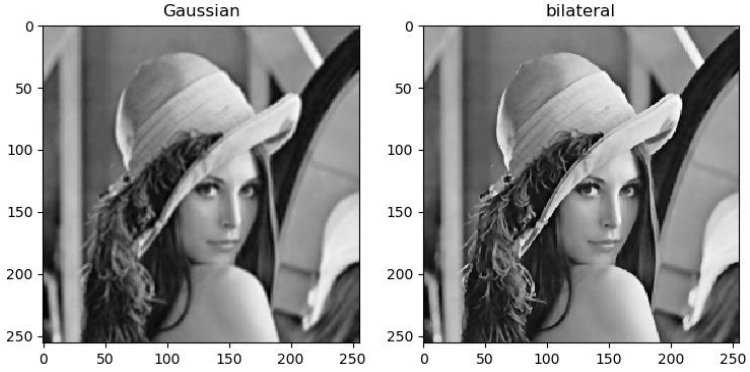
同样的，opencv 中提供了这一函数：

```
img = cv2.adaptiveBilateralFilter(img, (3,3), 1.5)
```

该函数为自适应双边滤波器，只需要第三个参数设置距离权值公式中的方差即可。

c. 结果

下面为高斯滤波器和自适应双边滤波器的对比



可以看出，在内核大小和距离 sigma 相同的情况下，双边滤波器虽然模糊了整体（去除噪声），但边界仍然清晰，是我们想要的。

2. 求灰度梯度

在这一步中，我们要求图像灰度值的梯度，这里可使用一阶有限差分来进行近似。

我使用了三种算子进行测试：

2.1 Canny 算子

a. 原理

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

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

b. 代码实现

```
def four_point(img):
    mx = np.array([-1, 1], [-1, 1])
    my = np.array([1, 1], [-1, -1])
    dx = cv2.filter2D(img, cv2.CV_32F, mx, anchor=(0, 0))/2
    dy = cv2.filter2D(img, cv2.CV_32F, my, anchor=(0, 0))/2
    res = np.sqrt(dx*dx+dy*dy)
    return res
```

实现直接使用 opencv 中的 filter2D 卷积函数，其中参数：
第一个为原灰度矩阵；第二个为数据类型这里设置 32 位浮点，防止计算中出现负数而溢出；第三个为算子；第四个由于算子并没有中心，需要手动指定锚点。最后，按照公式求出梯度矩阵。

c. 结果

2.2 Sobel 算子

Sobel 算子是一种常用的边缘检测算子，是一阶的梯度算法。对噪声具有平滑作用，提供较为精确的边缘方向信息，边缘定位精度不够高。

a. 原理

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

当 ksize=3 时，Sobel 内核可能产生比较明显的误差，此时，可以使用 Scharr 函数，该函数仅作用于大小为 3 的内核。具有比 Sobel 快的速度，而且结果更精确（具体优化的数学原理，参考 https://en.wikipedia.org/wiki/Sobel_operator），其内核为：

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

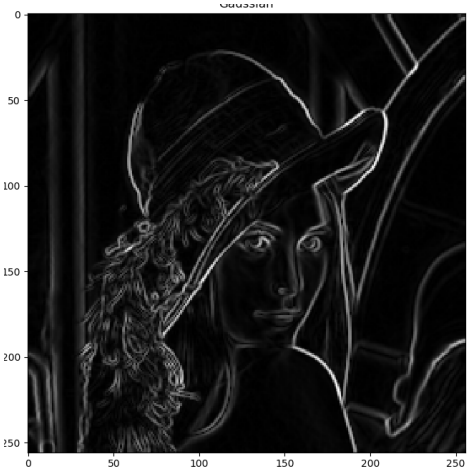
b. 代码实现

opencv 同样有 Scharr 函数：

```
dx = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=-1)/16
dy = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=-1)/16
```

使用 Sobel 函数，第三四个参数分别为 x,y 梯度比例，最后一个参数内核大小为-1 的话，就调用了 Scharr 函数。

c. 结果

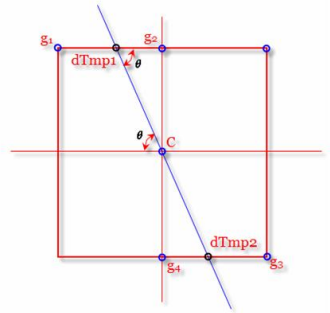


综合考虑两算子的利弊后，最终，我选用了 Scharr 算子作为梯度算子，使用它求了梯度幅值和梯度方向。

3. 非极大值抑制

a. 原理：

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梯度最大值，将非极大值点所对应的灰度梯度值置为 0，从而可以剔除掉一大部分非边缘点，可以认为使得边缘变得更 “细”。



上图为判断像素点 C 的灰度梯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 两点灰度梯度值。若 C 点灰度梯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。

b. 代码实现

根据原理图，我把要求的相邻点分成了 2X2 种情况：梯度角的绝对值是否大于 pi/4(决定了是 X 还是 Y 方向插值)，梯度角是否大于零（决定了插值点 X 和 Y 的选取），在确定情况后，只需根据几何关系求出两相邻点的值即可。

下面是梯度角在 0~pi/4 计算过程：

```
if(abs(angle) > pi/4):
    if(angle == 0):
        weight = 1
    else:
        weight = 1/abs(tan(angle))
    if(angle > 0):
        dtmp1 = (1-weight) * grad[y+1][x] + weight * grad[y+1][x+1]
        dtmp2 = (1-weight) * grad[y-1][x] + weight * grad[y-1][x-1]
```

其他的三种情况类似。

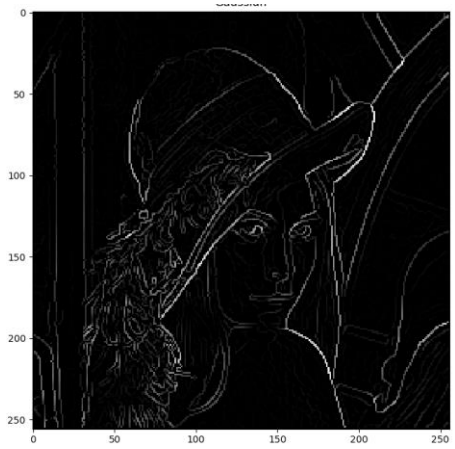
最后，将每一点和两相邻点梯度幅值比较：

```
if(grad[y][x] > dtmp1 and grad[y][x] > dtmp2):
    res[y][x] = grad[y][x]
```

除此之外，算法有一个优化地方就是若该点梯度为 0，可直接跳过，免去了后续的计算。

c. 结果

同样的用 matplotlib 绘图来看一下大概的结果。



可以看出，该图的边缘比原先细了不少。

4. 双阈值算法和连接边缘

Canny 算法中为了减少假边缘数量：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。

4.1 双阈值算法

首先我们需要指定阈值大小，由于手动指定需要调试，所以比较常见的方式是利用比例来确定阈值大小，即自适应确定阈值算法，一般来说高阈值设定为 0.7~0.8，而低阈值为高阈值的 1/3 ~ 1/2。

a. 原理

求出灰度梯度直方图，然后直方图中设置比例（0.7~0.8）对应的梯度大小为高阈值，再把高阈值乘另一个比例（1/3 ~ 1/2）得出低阈值。

b. 代码实现

下面是取得高阈值的过程

```
def get_high_threshold(grad, weight):
    maxm = int(np.max(grad))
    hist,bins = np.histogram(grad.ravel(), maxm+1, [0,maxm+1])
    cnt = 0
    threshold = float(grad.size-hist[0]) * weight
    for i in range(1, hist.size):
        cnt += hist[i]
        if(cnt > threshold):
            return i
```

上面算法实现要注意的是，由于非极大抑制之后，绝大多数点梯度为 0，也就是说，把梯度为 0 的点去掉之后，比例才显得比较有意义。

4.2 边缘连接

a. 原理：

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。

b. 代码实现

我使用了非递归实现（DFS），即先根据高阈值，将强边界灰度设为 255 并且设置栈分别存强边界的 x,y 坐标：

```
for i in range(1, grad.shape[0]-1):
    for j in range(1, grad.shape[1]-1):
        if(grad[i][j] > high):
            res[i][j] = 255
            y_stack.append(i)
            x_stack.append(j)
```

当栈不空时，取栈中的点，灰度设置为 255，并对该点周围 8 个点判断，若灰度梯度大于低阈值且未被处理过，则入栈，灰度设置为 255。

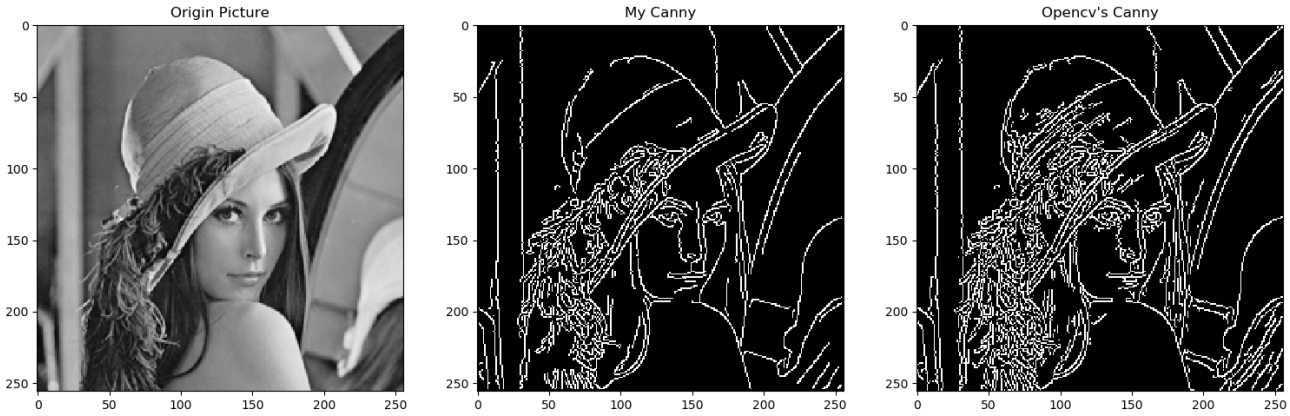
下面是每一步初始化及其中一个点的处理：

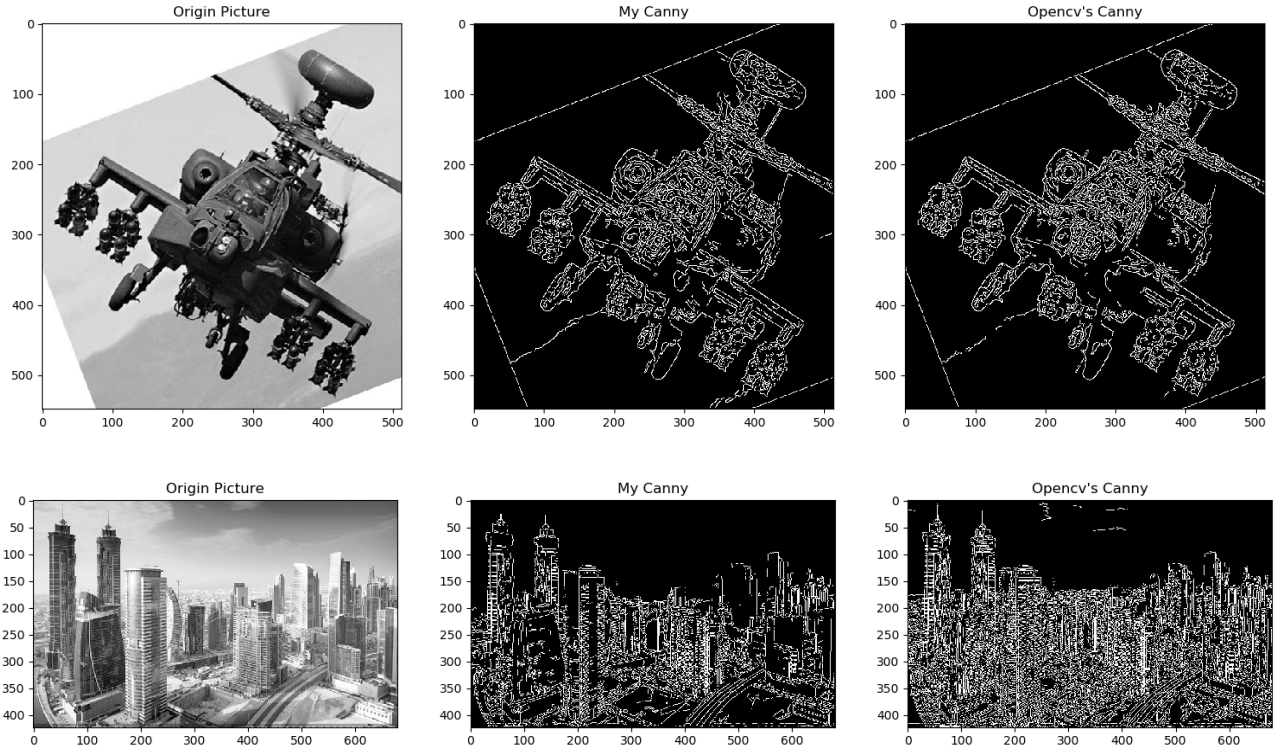
```
# weak edge
while(len(y_stack) != 0):
    y = y_stack.pop()
    x = x_stack.pop()
    res[y][x] = 255
    # right
    if(grad[y][x+1] > low and res[y][x+1] < 255):
        y_stack.append(y)
        x_stack.append(x+1)
        res[y][x+1] = 255
```

最终，栈空后，也就得到了连通的边缘

5. 结果

由左到右分别为原图，我的 Canny 结果，opencv 中 Canny 的结果：





分析：

可以看出，我的结果比 opencv 中自带的 Canny 要稀疏一些，猜测的原因有两点：一是在非极大值抑制过程中，opencv 中的实现是离散化近似没有使用插值，而且 opencv 中非极大值抑制和双阈值滞后算法是在一起实现的，可能导致条件变宽松，曲线更健壮一些；二是在调用 opencv 的 Canny 时，参数都直接使用了 PPT 上推荐的，可能没有针对图片选择参数，导致无效的边界多一些。

四、实验拓展

1. 关于 Otsu（大津法）求双阈值

在求双阈值的过程中，我用到了所谓的自适应求取双阈值的方法，但在实际使用中，还是需要调节高低阈值的比例，也是两个参数，只不过可能比直接调节阈值大小在直观上更容易理解。

经过学习，我了解到一种叫 Otsu 算法（大津法）可以用来根据图像来求双阈值，其基本思想是把图像像素分类，通过搜索计算类间方差最大值，得到最优阈值。选取的最佳阈值应当是用该阈值分割得到的类间具有最好的分离性，类间分离性最好的判据是数理统计意义上的类间方差最大化或者类内特性方差最小。

在 Canny 算法的应用中，可以把像素点分为三类，找高低阈值，使得评价函数最大。具体数学方法，参考：

https://wenku.baidu.com/view/607820c16137ee06eff9187b.html

但最终我没有使用该算法，主要原因有两点：

一是该算法寻找高低阈值的过程其实需要一对值一对值的逐个尝试，每一对

值的计算开销都不小，而需要尝试的对数又特别多，会极大的降低效率，速度极慢。

二是在那么大开销的情况下，Ostu 算法算出来的结果没有效果好太多，反而感觉不是很理想。

可见，想要设计一个不针对图像特点而自动确定理想高低阈值的算法是极其困难的，在实际应用中，还是应该针对图像的提取一些特征来具体分析。

五、总结

在本实验中，我实现了一遍 Canny 算法基本流程，在此过程中，学习了许多种滤波器，算子的特点，初步感受了图像处理一部分领域要做的工作。

最后，衷心感谢老师和助教们的精心准备和辛勤付出，你们整理完善的 ppt 和到位的答疑大大提高了我学习图像处理知识的效率，感谢~

贾萧松 516030910548
2017.12.2

源代码：

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-
import sys
reload(sys)
sys.setdefaultencoding('utf-8')

import cv2
import numpy as np
from math import *
from matplotlib import pyplot as plt

# origin canny gradient algorithm
def canny_operator(img):
    mx = np.array([[-1, 1], [-1, 1]])
    my = np.array([[1, 1], [-1, -1]])
    dx = cv2.filter2D(img, cv2.CV_32F, mx, anchor=(0, 0))/2
    dy = cv2.filter2D(img, cv2.CV_32F, my, anchor=(0, 0))/2
    res = np.sqrt(dx*dx+dy*dy)
    return res

# my final gradient algorithm
def scharr(img):
    dx = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=-1)/16
    dy = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=-1)/16
```

```
    amp = np.sqrt(dx*dx+dy*dy)
    # calculate the angle of the gradient
    angles = np.zeros(img.shape)
    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1):
            if(dx[i][j] == 0):
                angles[i][j] = np.sign(dy[i][j])*pi/2.0
            else:
                angles[i][j] = atan(dy[i][j]/dx[i][j])
    return amp, angles


def non_maxm_suppression(grad, angles):
    res = np.zeros(grad.shape)
    for y in range(1, grad.shape[0]-1):
        for x in range(1, grad.shape[1]-1):
            if(grad[y][x] == 0):
                continue
            angle = angles[y][x]
            if(abs(angle) > pi/4):
                if(angle == 0):
                    weight = 1
                else:
                    weight = 1/abs(tan(angle))
                if(angle > 0):
                    dtmp1 = (1-weight) * grad[y+1][x] + weight *
grad[y+1][x+1]
                    dtmp2 = (1-weight) * grad[y-1][x] + weight *
grad[y-1][x-1]
                else:
                    dtmp1 = (1-weight) * grad[y-1][x] + weight *
grad[y-1][x+1]
                    dtmp2 = (1-weight) * grad[y+1][x] + weight *
grad[y+1][x-1]
            else:
                weight = abs(tan(angle))
                if(angle > 0):
                    dtmp1 = (1-weight) * grad[y][x+1] + weight *
grad[y+1][x+1]
                    dtmp2 = (1-weight) * grad[y][x-1] + weight *
grad[y-1][x-1]
                else:
                    dtmp1 = (1-weight) * grad[y][x+1] + weight *
grad[y-1][x+1]
```

```
                dtmp2 = (1-weight) * grad[y][x-1] + weight *
grad[y+1][x-1]

            if(grad[y][x] > dtmp1 and grad[y][x] > dtmp2):
                res[y][x] = grad[y][x]
    return res


def get_high_threshold(grad, weight):
    maxm = int(np.max(grad))
    hist, bins = np.histogram(grad.ravel(), maxm+1, [0, maxm+1])
    cnt = 0
    threshold = float(grad.size-hist[0]) * weight
    for i in range(1, hist.size):
        cnt += hist[i]
        if(cnt > threshold):
            return i


def edge_linking(grad, high, low):
    res = np.zeros(grad.shape)
    y_stack = []
    x_stack = []
    # strong edge
    for i in range(1, grad.shape[0]-1):
        for j in range(1, grad.shape[1]-1):
            if(grad[i][j] > high):
                res[i][j] = 255
                y_stack.append(i)
                x_stack.append(j)

    # weak edge
    while(len(y_stack) != 0):
        y = y_stack.pop()
        x = x_stack.pop()
        res[y][x] = 255
        # right
        if(grad[y][x+1] > low and res[y][x+1] < 255):
            y_stack.append(y)
            x_stack.append(x+1)
            res[y][x+1] = 255
        # up right
        if(grad[y+1][x+1] > low and res[y+1][x+1] < 255):
            y_stack.append(y+1)
```

```
                x_stack.append(x+1)
                res[y+1][x+1] = 255
        # up
        if(grad[y+1][x] > low and res[y+1][x] < 255):
                y_stack.append(y+1)
                x_stack.append(x)
                res[y+1][x] = 255
        # up left
        if(grad[y+1][x-1] > low and res[y+1][x-1] < 255):
                y_stack.append(y+1)
                x_stack.append(x-1)
                res[y+1][x-1] = 255
        # left
        if(grad[y][x-1] > low and res[y][x-1] < 255):
                y_stack.append(y)
                x_stack.append(x-1)
                res[y][x-1] = 255
        # down left
        if(grad[y-1][x-1] > low and res[y-1][x-1] < 255):
                y_stack.append(y-1)
                x_stack.append(x-1)
                res[y-1][x-1] = 255
        # down
        if(grad[y-1][x] > low and res[y-1][x] < 255):
                y_stack.append(y-1)
                x_stack.append(x)
                res[y-1][x] = 255
        # down right
        if(grad[y-1][x+1] > low and res[y-1][x+1] < 255):
                y_stack.append(y-1)
                x_stack.append(x+1)
                res[y-1][x+1] = 255
    return res


def canny(filename):
    # 1 get the gray scale
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

    # 2 smooth filter
    #img = cv2.GaussianBlur(img, (3,3), 1.5)
    img_smoothed = cv2.adaptiveBilateralFilter(img, (3, 3), 75)

    # 3 get the gradient
```

```
    #grad = canny_operator(img)
    grad, angles = scharr(img_smoothed)

    # 4 Non-Maximum Suppression
    suppress = non_maxm_suppression(grad, angles)

    # 5 fuzzy thresholds algrithm and edge linking
    high = get_high_threshold(suppress, 0.7)
    low = round(high*0.4)
    img_edge = edge_linking(suppress, high, low)

    opencv_canny = cv2.Canny(img, 50, 150)
    plt.subplot(131), plt.imshow(
        img, cmap=plt.cm.gray), plt.title("Origin Picture")
    plt.subplot(132), plt.imshow(
        img_edge, cmap=plt.cm.gray), plt.title("My Canny")
    plt.subplot(133), plt.imshow(
        opencv_canny, cmap=plt.cm.gray), plt.title("Opencv's Canny")
    plt.show()

canny("1.jpg")
```