《数字图像处理》实验报告

时间: 2021 年 11 月 6 日（第 10 周，星期 六 ）

地点： 测绘工程实验室（B 区）-- B 实验大楼 202

学生姓名： 魏子继 学号： 20194947

一、 实验名称：

图像边缘处理—Canny 算法的实现

二、 实验目的：

本实验的目的是利用 Canny 算法实现图像的边缘提取。

在计算机视觉/机器视觉领域，图像分割的应用十分普遍，它是指将数字图像细分为多个图像子区域的过程。图像分割的目的是简化或改变图像的表示形式，使得图像更容易理解和分析。它的基本形式通常为定位图像中的物体和边界，结果是图像上子区域或轮廓线的集合。图像分割方法主要包括阈值处理（二值化）、聚类法、边缘检测和区域生长等。对于图像分割问题的求解没有统一范式，通常要与领域知识充分结合，才能更为有效地解决。

边缘检测是基于灰度突变来分割图像的常用方法，其实质是提取图像中不连续部分的特征。目前常见边缘检测算子有差分算子、Roberts 算子、Sobel 算子、Prewitt 算子、Log 算子以及 Canny 算子等。其中，Canny 算子是由计算机科学家 John F. Canny

于 1986 年提出的一种边缘检测算子，是目前理论上相对最完善的一种边缘检测算法。在 MATLAB、OpenCV 等常用图像处理工具中已有内置的 Canny 算子 API，本次实验将依据 Canny 算子的算法原理在 Python 环境中进行复现，并与 OpenCV 内置 Canny 算子的效果进行对比。

三、 实验步骤：

根据实验的目标和实现方法，该实验由以下五步完成：

1. 高斯滤波。确定合适的高斯核，与图像进行离散卷积，达到对图像滤波的作用，去除一些噪声。
2. 像素梯度计算。在本实验中，选择使用 Sobel 算子，按照 Sobel 滤波器步骤计算图像像元的幅值与方向，寻找图像的强度梯度。结果得到图像的梯度强度矩阵。
3. 非极大值抑制。非极大值像素梯度抑制的目的在于消除边缘检测带来的杂散响应，起到将边缘“瘦身”的作用。其基本方法是将当前像素梯度强度与沿正负梯度方向上的相邻像素的梯度强度进行比较，若其最大（即为极值），则保留该像素为边缘点，若不是最大，则对其进行抑制，不将其作为边缘点。为了更精确计算，通常在跨越梯度方向的两个相邻像素之间使用线性插值来得到要参与比较的像素梯度。
4. 滞后阈值处理。此步的方法是定义一个高阈值和一个低阈值。梯度强度低于低阈值的像素点被抑制，不作为边缘点；高于

高阈值的像素点被定义为强边缘，保留为边缘点；处于高低阈值之间的定义为弱边缘，留待进一步处理。本实验中，高阈值和低阈值作为参数需要输入。

5. 孤立弱边缘抑制。通常而言，由真实边缘引起的弱边缘像素点将连接到强边缘像素点，而噪声响应则未连接。通过查看弱边缘像素及其 8 个邻域像素，可根据其与强边缘的连接情况来进行判断。一般，可定义只要其中邻域像素其中一个为强边缘像素点，则该弱边缘就可以保留为强边缘，即真实边缘点。

四、 实验中的关键点分析（包括关键算法与代码实现）：

边缘检测之 Canny 算法的实现（EdgeDetection.py）

```
"""
Created by Chloe on 6/11/2021
"""
import cv2.cv2 as cv2
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']  # 图像字体汉化


# Canny 算法边缘提取
def Canny(img, minT, maxT):
    """
    Canny 算法边缘提取
    :param img: 原图像，不为灰度
    :param minT: 低阈值
    :param maxT: 高阈值
    :return: Canny 算法结果图像
    """
    # 获取灰度图像
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
22.
23.     # 定义空白图像，用于存放高斯滤波处理后的图像
24.     gauss_img = np.zeros((height, width), dtype=gray.dtype)
25.
26.     # 定义高斯核卷积模板
27.     a = np.array([[1, 4, 7, 4, 1],
28.                   [4, 16, 26, 16, 4],
29.                   [7, 26, 41, 26, 7],
30.                   [4, 16, 26, 16, 4],
31.                   [1, 4, 7, 4, 1]])
32.     kernel = a * (1 / 273)
33.
34.     '''''
35.     使用高斯平滑去除噪声
36.     '''
37.     # 对图像进行高斯滤波处理
38.     for i in range(2, height - 2):
39.         for j in range(2, width - 2):
40.             sum_num = np.sum(gray[i - 2:i + 2 + 1, j - 2:j + 2 + 1] * ker
    nel)
41.             gauss_img[i, j] = sum_num
42.
43.     '''''
44.     按照 Sobel 滤波器步骤计算梯度幅值与方向，寻找图像强度梯度
45.     '''
46.     sobel_img = np.zeros((height, width), dtype=gray.dtype)
47.     # 对阈值化图像遍历，进行 Sobel 求解梯度值
48.     for i in range(height - 1):
49.         for j in range(width - 1):
50.             dx = (int(gauss_img[i - 1, j - 1]) + 2 * int(gauss_img[i - 1,
     j]) + int(gauss_img[i - 1, j + 1])) - (
51.                     int(gauss_img[i + 1, j - 1]) + 2 * int(gauss_img[i +
    1, j]) + int(gauss_img[i + 1, j + 1]))
52.             dy = (int(gauss_img[i - 1, j + 1]) + 2 * int(gauss_img[i, j +
     1]) + int(gauss_img[i + 1, j + 1])) - (
53.                     int(gauss_img[i - 1, j - 1]) + 2 * int(gauss_img[i, j
     - 1]) + int(gauss_img[i + 1, j - 1]))
54.             sobel_img[i, j] = np.sqrt(dx ** 2 + dy ** 2)
55.
56.     '''''
57.     通过 Non-maximum Suppression 过滤非边缘元素
58.     '''
59.     suppression_img = np.zeros((height, width), dtype=gray.dtype)
60.     # 对阈值化图像遍历，进行 non-maximum suppression
```

```
    for i in range(height - 1):
        for j in range(width - 1):
            # 首先仍是 Sobel 算子的计算结果，因 sobel_image 后还要用做判断，此处不做修改，而是再生成一系列。
            dx = (int(gauss_img[i - 1, j - 1]) + 2 * int(gauss_img[i - 1, j]) + int(gauss_img[i - 1, j + 1])) - (
                    int(gauss_img[i + 1, j - 1]) + 2 * int(gauss_img[i + 1, j]) + int(gauss_img[i + 1, j + 1]))
            dy = (int(gauss_img[i - 1, j + 1]) + 2 * int(gauss_img[i, j + 1]) + int(gauss_img[i + 1, j + 1])) - (
                    int(gauss_img[i - 1, j - 1]) + 2 * int(gauss_img[i, j - 1]) + int(gauss_img[i + 1, j - 1]))

            # 确保分母不为 0
            dx = np.maximum(dx, 1e-10)
            theta = np.arctan(dy / dx)

            '''''
            确定梯度角度
            '''
            if -0.4142 < theta < 0.4142:
                angle = 0
            elif 0.4142 < theta < 2.4142:
                angle = 45
            elif abs(theta) > 2.4142:
                angle = 90
            elif -2.4142 < theta < -0.4142:
                angle = 135

            '''''
            根据梯度角度方向，求对应的非极大值抑制
            '''
            if angle == 0:
                if max(sobel_img[i, j], sobel_img[i, j - 1], sobel_img[i, j + 1]) == sobel_img[i, j]:
                    # 比较 x 方向梯度三个值中的最大值，如果 Sobel_img[i,j]最大则保留，否则设置为 0
                    suppression_img[i, j] = sobel_img[i, j]
                else:
                    suppression_img[i, j] = 0
            elif angle == 45:
                if max(sobel_img[i, j], sobel_img[i - 1, j + 1], sobel_img[i + 1, j - 1]) == sobel_img[i, j]:
```

```
                    # 比较正对角线方向梯度三个值中的最大值，如果sobel_img[i,j]最大则保留，否则设置为0
                    suppression_img[i, j] = sobel_img[i, j]
                else:
                    suppression_img[i, j] = 0
            elif angle == 90:
                if max(sobel_img[i, j], sobel_img[i - 1, j], sobel_img[i + 1.j]) == sobel_img[i, j]:
                    # 比较y方向梯度三个值中的最大值，如果sobel_img[i,j]最大则保留，否则设置为0
                    suppression_img[i, j] = sobel_img[i, j]
                else:
                    suppression_img[i, j] = 0
            elif angle == 135:
                if max(sobel_img[i, j], sobel_img[i - 1, j - 1], sobel_img[i + 1, j - 1]) == sobel_img[i, j]:
                    # 比较反对角线方向梯度三个值中的最大值，如果sobel_img[i,j]最大则保留，否则设置为0
                    suppression_img[i, j] = sobel_img[i, j]
                else:
                    suppression_img[i, j] = 0

        '''''
        利用双阈值方法确定潜在的边界
        '''
        canny_img = np.zeros((height, width), dtype=gray.dtype)
        # 对阈值化图像遍历，进行双阈值处理
        for i in range(height):
            for j in range(width):
                if suppression_img[i, j] >= maxT:  # 大于高阈值，设置为255
                    canny_img[i, j] = 255
                elif suppression_img[i, j] <= minT:  # 小于低阈值，设置为0
                    canny_img[i, j] = 0
                else:
                    '''''
                    利用滞后技术跟踪边界，若某一像素位置和strong edge相连的weak edge认定是边界，其余的若边界删除(设为0)
                    '''
                    # 周围8邻域内有比该像素值更大的像素，则设置为255，否则设置为0
                    if max(suppression_img[i - 1, j - 1], suppression_img[i - 1, j], suppression_img[i - 1, j + 1],
```

```
                            suppression_img[i, j - 1], suppression_img[i, j + 1], suppression_img[i + 1, j - 1],
                            suppression_img[i + 1, j], suppression_img[i + 1, j + 1]) >= suppression_img[i, j]:
                        canny_img[i, j] = 255
                    else:
                        canny_img[i, j] = 0

        # 转为可输出格式
        canny_img = cv2.cvtColor(canny_img, cv2.COLOR_BGR2RGB)
        return canny_img


    # 主程序
    if __name__ == '__main__':
        # 读取图片
        image = cv2.imread("D:\\pyfiles\\DigitalImageProcessing\\image\\Celtics.jpg")

        # 获取图片尺寸
        height, width = image.shape[0:2]

        # 使用 opencv 进行 Canny 算子边缘提取
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # blur = cv2.GaussianBlur(gray, (3, 3), 0)  # 高斯滤波
        Canny_opencv = cv2.Canny(gray, 50, 200)  # 80 为低阈值，255 为高阈值

        # 使用 Canny 算法进行边缘提取
        Canny_img = Canny(image, 50, 200)
        # cv2.imshow('Canny_img', Canny_img)
        # cv2.waitKey()

        # 将图像转为 plt 输出格式
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        Canny_opencv = cv2.cvtColor(Canny_opencv, cv2.COLOR_BGR2RGB)

        # 图像显示
        titles = ['原图', 'opencv 中 canny 算法', 'canny 算法']
        images = [image, Canny_opencv, Canny_img]
        for i in range(3):
            plt.subplot(1, 3, i + 1)
            plt.imshow(images[i])
            plt.title(titles[i])
```

```
            plt.axis('off')
        plt.savefig("D:\\pyfiles\\DigitalImageProcessing\\result\\CannyImage1111.png", bbox_inches='tight')
        plt.show()
```

五、 实验原始数据与实验结果：

源图像：（世界上最伟大的球队凯尔特人队）

结果图像：

六、 问题分析与心得体会：

在实验过程中，主要遇到了以下问题并做出总结：

1. Canny 算法的速度比我预想的速度要快很多，甚至比大津算法还快。但是在最后编写完成代码后发现了一个不太理想的状态型问题，即算法模块设计的问题。在本算例中，将 Canny 算法的整个实现过程打包在了一个 def 中，完成之后发现能

够将高斯滤波的 Sobel 算子过程单独抽象出来成为两个 def，这样的话使得本算法的结构更加整洁，抽象程度更高。

2. 选择本图的原因是这个图像背景有明显的线形条纹，易于 Canny 算法的线边缘提取。在 EdgeDetection 中，设计的目标是自行书写的 Canny 算法和 opencv 中的 Canny 算法图像处理结果相同，但在完成比较之后发现差强人意。主要体现在我的图像的部分边缘提取后是断断续续地。与老师探讨原因并查阅相关文献后，认为问题可能出现在高斯滤波的过程中，因为没有编写专门的滤波方法，而只是简单得进行高斯滤波，因此和 opencv 的 Canny 算法得出图像有些偏差。同时，因为在 opencv 中使用 Canny 算法的过程实际没有进行滤波，因此有双边缘的嫌疑（滤波后有明显的边缘丢失）。下图即为进行了使用 opencv 自带算法进行高斯滤波后再进行 Canny 算法的图示：