计算机视觉
课程实验报告

姓名：	史瑞潇
学号：	58121204
完成时间：	2023/10/29

目录

1 实验目标与基本信息

实验名称	边缘提取
实验时间	2023.10
实验内容提要	实现对彩色图像的灰度处理，并使用高斯一阶差分滤波器计算图像梯度，进而执行非极大值抑制和阈值操作及连接，从而进行 canny 边缘检测。
实验完成人	史瑞潇
实验运行环境	AMD Ryzen 7 7840H 3.80 GHz Windows 11,22H2 Python 3.11

2 实验过程

2.1 Canny 算法流程

Canny 算法是一种用于边缘检测的经典计算机视觉算法。其基本步骤可以分为以下 4 步：

1. 噪声抑制（高斯平滑滤波）：

   对于读入的灰度图像，Canny 算法会通过应用高斯滤波器来降低图像中的噪声。5×5大小的高斯平滑算子的近似值如下：

$$\frac{1}{273} \times \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

2. 梯度强度和方向计算：

   对平滑后的图像，使用 Sobel 算子进行梯度强度和方向的计算。常用的3×3大小的 Sobel 算子为：

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

3. 非极大值抑制（Non-Maximum Suppression, NMS）

   Canny 算法会对梯度图像进行非极大值抑制，以细化边缘。这一步骤通过检查每个像素点的梯度强度，只保留局部梯度最大的像素，以获得更细的边缘。

4. 双阈值边缘跟踪与边缘连接

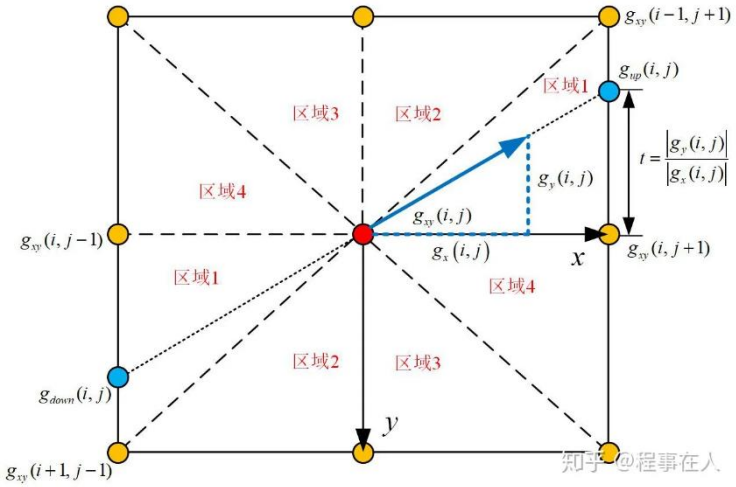
在这个步骤中，用户通过给出阈值的上下界对像素进行区分。像素点的梯度强度如果高于高阈值，被标记为强边缘；如果在高阈值和低阈值之间，被标记为弱边缘；如果低于低阈值，被标记为非边缘。

弱边缘像素可能是真正边缘的一部分。Canny 算法通过连接弱边缘像素到强边缘像素来形成完整的边缘。

2.2 Canny 算法实现与代码说明

在个人实现代码中，Canny 算法被封装为一个类 myCanny，该类具有以下几种类方法：

1. myCanny.GaussianKernel 生成高斯平滑矩阵
   通过给定高斯核大小、高斯 sigma 参数生成指定参数下的高斯矩阵。
2. myCanny. Smooth 将高斯平滑核作用在图像上
   通过 1 中生成的高斯平滑矩阵对图像进行平滑
3. myCanny. SobelFliter Sobel 滤波器求导数
   通过三阶 Sobel 算子，求图像的各方向导数、梯度强度及方向
4. myCanny.NMS 非极大值抑制
   在 NMS 步骤中，实现细节使用了插值的方式计算临近值。



如上图所示，可将像素的邻接情况划分为 4 个区域，其中每个区域包含上下两部分。若中心像素点 x 方向梯度强度为 $g_x(i,j)$ ， y 方向梯度强度为 $g_y(i,j)$ ，梯度强度为 $g_{xy}(i,j)$ ，则根据 $g_x(i,j)$ 和 $g_y(i,j)$ 的正负和大小可判断出其梯度方向所属区域，进而根据其像素梯度方向以及相邻点的像素梯度线性插值得到正负梯度方向的两个参与比较的梯度强度 $g_{up}(i,j)$ 和 $g_{down}(i,j)$。公式如下：

$$g_{up}(i,j) = (1-t)\cdot g_{xy}(i,j+1) + t\cdot g_{xy}(i-1,j+1)$$
$$g_{down}(i,j) = (1-t)\cdot g_{xy}(i,j-1) + t\cdot g_{xy}(i+1,j-1)$$

其他三个区域的计算方法类似。[1]

[1] 本段原理及图像引用自 Canny 边缘检测算法 - 知乎 (zhihu.com)

5. myCanny.DoubleThresold 双阈值边缘跟踪与边缘连接
具体实现中，采用了深度优先搜索（DFS）的方式进行连接。这与 OpenCV API[2]中的实现一致。

3 实验结果与分析

3.1 图像测试与 OpenCV 交互实验

按照要求，我们对 1.jpg 进行测试：

采用(45,85)阈值得到的图像(左图)

该图像与 OpenCV API 采用(50, 100)阈值得到的图像近似(右图)

但是，相比于 OpenCV API，我们的 Canny 算法还有一些问题与差距，分析如 3.2 节所示

[2] opencv/modules/imgproc/src/canny.cpp at 617d7ff575200c0a647cc615b86003f10b64587b · opencv/opencv (github.com)

3.2 对比实验与分析

3.2.1 双阈值与连接步骤 DFS 与邻域检查对比

在实现双阈值与连接步骤时，确定弱边缘是否为真实边缘可以使用深度优先搜索或者检查邻域是否为强边缘的方法。这两种方法中，第一种方法效率较慢，但效果较好，能提高边缘的连接性；第二种则反之，在提高效率的同时降低了边缘的连接性。


对比同一个部位两种方法（在同样阈值下处理），DFS 在衣摆处的细节处理强于邻域检查。

3.2.2 与 OpenCV API 对比

在相同阈值下，OpenCV API 实现的图像提取的细节更多。

经过实验，通过 OpenCV 的 Sobel 算子进行 NMS 前置步骤，效果和我们自己实现的函数基本无差别；因此，我们推断，这有可能是 OpenCV 的 Canny 函数在实现上采用计算强度梯度方式与我们使用的略有不同所导致的。

4 附录：实验代码

代码 1 myCanny.py

```
import numpy as np
import cv2
from scipy.signal import convolve2d
import matplotlib.pyplot as plt

class myCanny:
    def __init__(self):
        pass

    def GaussianKernel(self, size, sigma):
```

```
        '''
        myCanny.GaussianKernel 生成高斯平滑矩阵
        :param size: 高斯核大小
        :param sigma: 高斯 sigma 参数
        :return: Gaussian 指定参数下的高斯矩阵
        '''
        kernel = np.fromfunction(
            lambda x, y: (1 / (2 * np.pi * sigma ** 2)) * np.exp(
                -((x - (size - 1) / 2) ** 2 + (y - (size - 1) / 2) ** 2)
/ (2 * sigma ** 2)),
            (size, size)
        )
        gaussian = kernel / np.sum(kernel)
        return gaussian

    def Smooth(self, image, size=5, sigma=1.4):
        '''
        myCanny.Smooth 将高斯平滑核作用在图像上
        :param image: 待处理图像
        :param size: 高斯核大小
        :param sigma: 高斯 sigma 参数
        :return: result 平滑后的图像
        '''
        width, height = image.shape
        kernel = self.GaussianKernel(size=size, sigma=sigma)
        result     =     convolve2d(image,     kernel,     mode='same',
boundary='wrap')

        return result

    def SobelFliter(self, image):
        '''
        myCanny.SobelFliter Sobel 滤波器
        :param image: 待处理图像
        :return:     gradient_x,     gradient_y,     gradient_magnitude,
gradient_direction
        图像的各方向导数、梯度强度及方向
        '''
        Gx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
        Gy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

        width, height = image.shape

        gradient_x = cv2.filter2D(image, -1, Gx)
```

```
        gradient_y = cv2.filter2D(image, -1, Gy)

        gradient_magnitude = np.sqrt(gradient_x ** 2 + gradient_y ** 2)
        gradient_direction = np.arctan2(gradient_y, gradient_x)

        return     gradient_x,     gradient_y,     gradient_magnitude,
gradient_direction

    def NMS(self, grad, dx, dy):
        '''
        myCanny.NMS 非极大值抑制
        :param grad: 梯度强度
        :param dx: x 方向梯度
        :param dy: y 方向梯度
        :return: nms 处理后的梯度强度
        '''
        W2, H2 = grad.shape
        nms = np.copy(grad)
        nms[0, :] = nms[W2 - 1, :] = nms[:, 0] = nms[:, H2 - 1] = 0
        for i in range(1, W2 - 1):
            for j in range(1, H2 - 1):

                if grad[i, j] == 0:
                    nms[i, j] = 0
                else:
                    gradX = dx[i, j]
                    gradY = dy[i, j]
                    gradTemp = grad[i, j]

                    if np.abs(gradY) > np.abs(gradX):
                        weight = np.abs(gradX) / np.abs(gradY)
                        g2 = grad[i - 1, j]
                        g4 = grad[i + 1, j]
                        if gradX * gradY > 0:
                            g1 = grad[i - 1, j - 1]
                            g3 = grad[i + 1, j + 1]
                        else:
                            g1 = grad[i - 1, j + 1]
                            g3 = grad[i + 1, j - 1]

                    else:
                        weight = np.abs(gradY) / np.abs(gradX)
                        g2 = grad[i, j - 1]
                        g4 = grad[i, j + 1]
```

```
                    if gradX * gradY > 0:
                        g1 = grad[i + 1, j - 1]
                        g3 = grad[i - 1, j + 1]
                    else:
                        g1 = grad[i - 1, j - 1]
                        g3 = grad[i + 1, j + 1]

                gradTemp1 = weight * g1 + (1 - weight) * g2
                gradTemp2 = weight * g3 + (1 - weight) * g4
                if gradTemp >= gradTemp1 and gradTemp >= gradTemp2:
                    nms[i, j] = gradTemp
                else:
                    nms[i, j] = 0

        return nms

    def DoubleThresold(self, image, low, high):
        '''
        myCanny.DoubleThresold 双阈值边缘跟踪与边缘连接
        :param image: 待处理梯度强度
        :param low: 低阈值
        :param high: 高阈值
        :return: doublethresold 二值图像
        '''
        width, height = image.shape
        doublethresold = np.zeros(image.shape)

        stack = []

        for i in range(1, width - 1):
            for j in range(1, height - 1):
                if image[i, j] < low:
                    doublethresold[i, j] = 0
                elif image[i, j] > high:
                    doublethresold[i, j] = 255

                    t = (i, j)
                    stack.append(t)
                # elif (image[i, j - 1] > high) or (image[i - 1, j - 1] >
high) or (
                #         image[i + 1, j - 1] > high) or (image[i - 1,
j] > high) or (image[i + 1, j] > high) or (
                #         image[i - 1, j + 1] > high) or (image[i, j +
1] > high) or (image[i + 1, j + 1] > high):
```

```
            #     doublethresold[i, j] = 255
            else:
                doublethresold[i, j] = 100

    # 基于 DFS 实现的方式
    while len(stack):
        x, y = stack[-1]
        stack.pop()
        print(x, y)

        if doublethresold[x - 1, y] == 100:
            doublethresold[x - 1, y] = 255
            t = (x-1,y)
            stack.append(t)
        if doublethresold[x - 1, y - 1] == 100:
            doublethresold[x - 1, y - 1] = 255
            t = (x-1,y-1)
            stack.append(t)
        if doublethresold[x - 1, y + 1] == 100:
            doublethresold[x - 1, y + 1] = 255
            t = (x-1,y+1)
            stack.append(t)
        if doublethresold[x, y - 1] == 100:
            doublethresold[x, y - 1] = 255
            t = (x,y-1)
            stack.append(t)
        if doublethresold[x, y + 1] == 100:
            doublethresold[x, y + 1] = 255
            t = (x,y+1)
            stack.append(t)
        if doublethresold[x + 1, y] == 100:
            doublethresold[x + 1, y] = 255
            t = (x+1,y)
            stack.append(t)
        if doublethresold[x + 1, y - 1] == 100:
            doublethresold[x + 1, y - 1] = 255
            t = (x+1,y-1)
            stack.append(t)
        if doublethresold[x + 1, y + 1] == 100:
            doublethresold[x + 1, y + 1] = 255
            t = (x + 1, y + 1)
            stack.append(t)

    for i in range(1, width - 1):
```

```
            for j in range(1, height - 1):
                if doublethresold[i, j] == 100:
                    doublethresold[i, j] = 0

        return doublethresold


def RGB2gray(rgb):
    '''
    RGB2gray 转灰度图像
    :param rgb: RGB 图像
    :return: Mat 灰度图像
    '''
    return np.dot(rgb[..., :3], [0.299, 0.587, 0.114])




if __name__ == '__main__':
    mycanny = myCanny()

    img = plt.imread('1.jpg')
    grayimg = RGB2gray(img)
    smoothimg = mycanny.Smooth(grayimg)
    sobel_x,        sobel_y,        sobel_mag,        sobel_dir        =
mycanny.SobelFliter(smoothimg)
    nms = mycanny.NMS(sobel_mag, sobel_x, sobel_y)

    print(np.max(nms))
    ans = mycanny.DoubleThresold(nms, 45, 65)

    cv2.imshow('test1', ans)
    cv2.waitKey(0)

    img2 = cv2.imread('1.jpg', cv2.IMREAD_GRAYSCALE)
    blur = cv2.GaussianBlur(img2, (5, 5), 0)
    ans2 = cv2.Canny(blur, 50, 100)
    cv2.imshow('test2', ans2)
    cv2.waitKey(0)
```