Lab2 实验报告

李青雅 523030910004 电院 2301

2024 年 12 月 2 日

目录

1 实验概览

本次实验主要学习了图像边缘提取和 Canny 边缘检测算法。

图像边缘是图像处理中一个非常重要的概念，它通常指的是图像中亮度或颜色发生剧烈变化的区域。图像边缘对于许多计算机视觉任务（如图像分割、物体识别、图像增强等）至关重要，因为它们通常代表了物体的轮廓、边界或者图像中的重要信息。边缘通常是指图像亮度值变化显著的地方，表现为像素值的剧烈变化。

Canny 边缘检测算法是图像处理中一种非常经典和广泛使用的边缘检测算法。由 John F. Canny 在 1986 年提出，Canny 算法的目标是检测图像中的边缘，并且尽量做到检测到真实的边缘，定位准确和抑制噪声。Canny 算法的主要步骤包括高斯滤波、计算梯度、非极大值抑制、高低阈值连接等。

2 实验环境

Visual Studio Code 中的 Python，OpenCV 库用于图像处理，Matplotlib 库用于绘图，numpy 库用于数学处理，os 库用于调整工作目录。

3 解决思路

3.1 灰度化

通常摄像机获取的是彩色图像，而检测的首要步骤是进行灰度化，以 RGB 格式彩图为例，一般的灰度化方法有两种：

方法一：

$$Gray = (R + G + B)/3$$

方法二：

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

（参数考虑人眼的生理特点）

鉴于 OpenCV 中的 imread 函数读取图像时，可以直接传入参数 cv2.IMREAD_GRAYSCALE，因此本实验选择方法二。

3.2 高斯滤波

为了减小图像中的噪声，需要对图像进行高斯滤波。通过卷积操作平滑图像，提高边缘检测的准确性，减少误检。

图像高斯滤波的实现可以用两个一维高斯核分别两次加权实现，也可以通过一个二维高斯核一次卷积实现。离散化的一维高斯函数与二维高斯函数如下：

离散一维高斯函数：

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

离散二维高斯函数：

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

具体而言，大小为 (2k+1)x(2k+1) 的高斯滤波器核的生成方程式由下式给出：

$$H_{ij}=\frac{1}{2\pi\sigma^2}e^{-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}}\quad 1\le i,j\le 2k+1$$

本次实验取 $\sigma=1.4$，$k=1$，生成 3x3 的高斯滤波器核（归一化后）的结果为：

$$H=\begin{bmatrix}0.0924 & 0.1192 & 0.0924\\ 0.1192 & 0.1538 & 0.1192\\ 0.0924 & 0.1192 & 0.0924\end{bmatrix}$$

若图像中一个 3x3 的窗口为 A，要滤波的像素点为 e，则经过高斯滤波之后，像素点 e 的亮度值为：

$$e=H*A=\begin{bmatrix}h_{11} & h_{12} & h_{13}\\ h_{21} & h_{22} & h_{23}\\ h_{31} & h_{32} & h_{33}\end{bmatrix}*\begin{bmatrix}a & b & c\\ d & e & f\\ g & h & i\end{bmatrix}=\begin{bmatrix}h_{11}\times a & h_{12}\times b & h_{13}\times c\\ h_{21}\times d & h_{22}\times e & h_{23}\times f\\ h_{31}\times g & h_{32}\times h & h_{33}\times i\end{bmatrix}$$

其中其中 * 为卷积符号，sum 表示矩阵中所有元素相加求和。

3.3 梯度计算

关于图像灰度值得梯度可使用一阶有限差分来进行近似，这样就可以得图像在 x 和 y 方向上偏导数的两个矩阵。常用的梯度算子有如下几种（本次实验以掌握 Sobel 算子为主）：

3.3.1 Roberts 算子

$$s_x=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\quad s_y=\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}$$

上式为其 x 和 y 方向偏导数计算模板，可用数学公式表达其每个点的梯度幅值为：

$$G[x,y]=|f(x+1,y+1)-f(x,y)|+|f(x+1,y)-f(x,y+1)|$$

3.3.2 Sobel 算子

$$s_x=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}\quad s_y=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}\quad K=\begin{bmatrix}a_0 & a_1 & a_2\\ a_7 & [i,j] & a_3\\ a_6 & a_5 & a_4\end{bmatrix}$$

上式三个矩阵分别为该算子的 x 向卷积模板、y 向卷积模板以及待处理点的邻域点标记矩阵，据此可用数学公式表达其每个点的梯度幅值为：

$$G_{x,i,j}=\sum(s_x*K)$$
$$G_{y,i,j}=\sum(s_y*K)$$
$$G_{i,j}=\sqrt{G_{x,i,j}^2+G_{y,i,j}^2}$$

3.3.3 Prewitt 算子

Prewitt 算子与 Sobel 算子及其类似，只是其卷积模板不同，其 x 和 y 方向的卷积模板如下：

$$s_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad s_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

3.3.4 Canny 算子

其 x 和 y 方向的卷积模板如下：

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

其 x 向、y 向的一阶偏导数矩阵，梯度幅值以及梯度方向的数学表达式为：

$$P[i,j] = (f[i,j+1] - f[i,j]) + (f[i+1,j+1] - f[i+1,j])/2 \tag{1}$$

$$Q[i,j] = (f[i,j] - f[i+1,j] + f[i,j+1] - f[i+1,j+1])/2 \tag{2}$$

$$M[i,j] = \sqrt{P[i,j]^2 + Q[i,j]^2} \tag{3}$$

$$\theta[i,j] = \arctan\left(\frac{Q[i,j]}{P[i,j]}\right) \tag{4}$$

求出这几个矩阵后，就可以进行下一步的检测过程（本实验中我们使用 sobel 算子）。

3.4 非极大值抑制

图像梯度幅值矩阵中的元素值越大，说明图像中该点的梯度值越大。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。

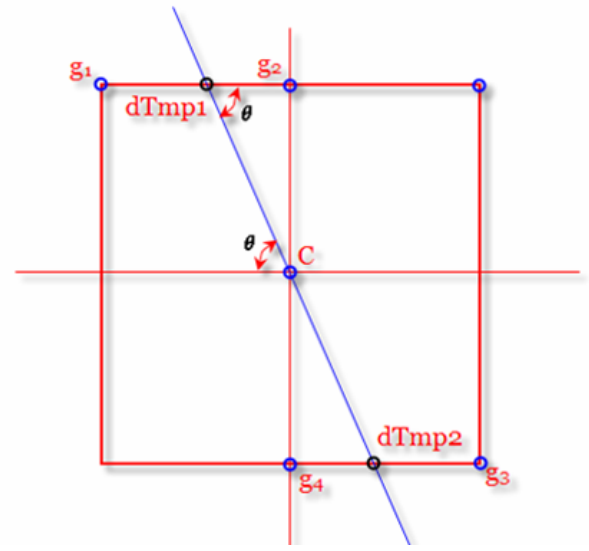


图 1: 非极大值抑制示意图

由 3.3.2 得到的 $G_{x,i,j}$（向右）和 $G_{y,i,j}$（向上）梯度，可以计算得该点的梯度方向。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。

由于像素是离散存储的，梯度方向不一定恰好命中领域的点（如图中的 $dTmp1, dTmp2$）。为了得到对应的像素值，我们可以通过插值的方法计算他们的值。

具体方法为计算角度在 45 度范围内的余数，得到当前像素梯度方向在 45 度内的相对位置，接着将相对位置的值归一化，记为 $weight$, 而 $interpolated_value$ 是通过线性插值计算得到的值。

线性插值公式为：

$$\text{interpolated\_value} = \text{dTmp1} \times (1 - \text{weight}) + \text{dTmp2} \times \text{weight}$$

3.5 双阈值算法检测和边缘连接

Canny 算法中减少假边缘数量的方法是采用双阈值法。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。对非极大值抑制图像作用两个阈值 $th1$ 和 $th2$, 两者关系 $th1 = 0.4th2$ (可尝试不同值)。

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 代码运行结果及分析

4.1 使用 Sobel 算子进行边缘检测与 OpenCV 库函数的对比

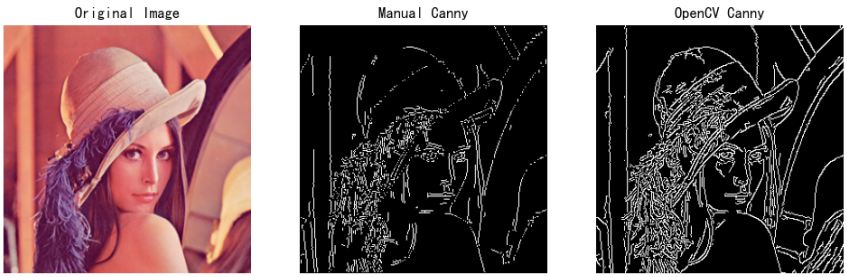


图 2

上图为示例 1 图像的原图，Sobel 算子边缘检测结果和 OpenCV 库函数的边缘检测结果，可以看到由于手动实现 Canny 边缘检测的步骤较为详细，可以更好地理解 Canny 算法的内部原理，但在效率和优化上可能不如 OpenCV 内置的实现。OpenCV 的实现通常会有更好的性能和优化处理，手动实现的版本需要更精细的调优才能接近 OpenCV 的精度。

下图为示例 2，示例 3 的结果：

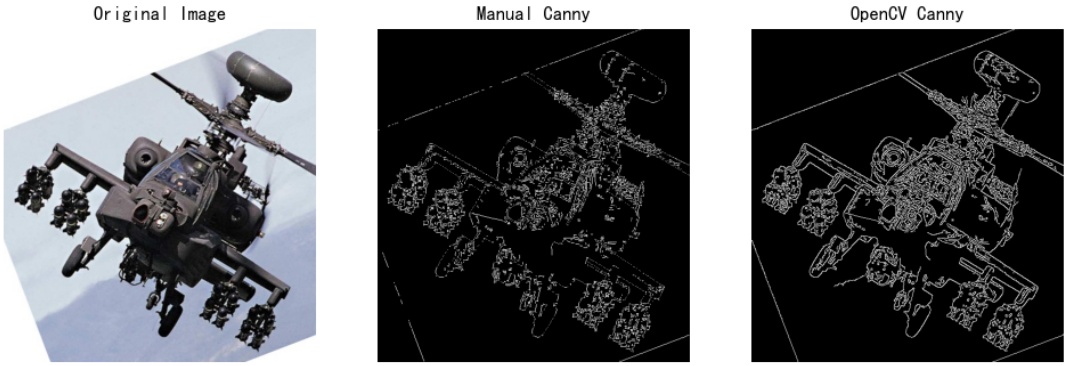


图 3

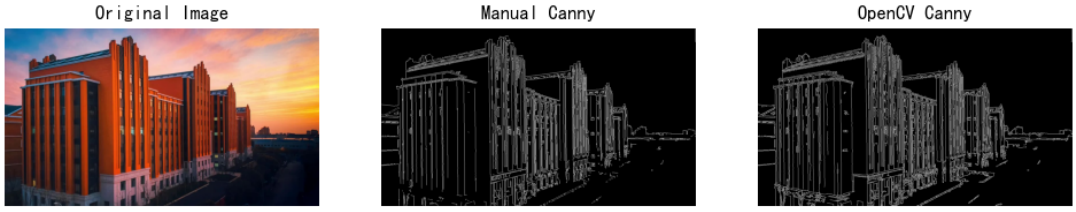
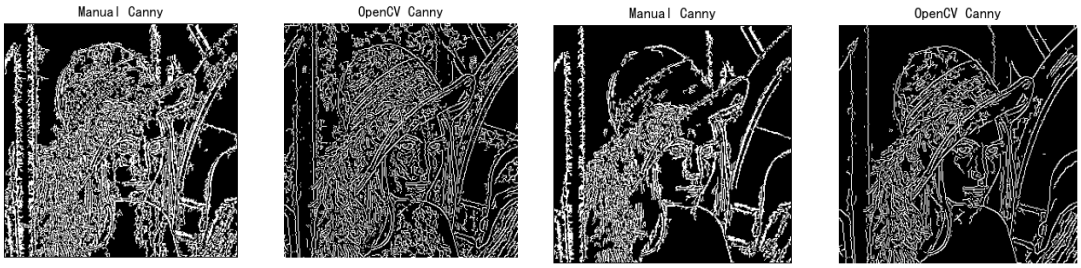


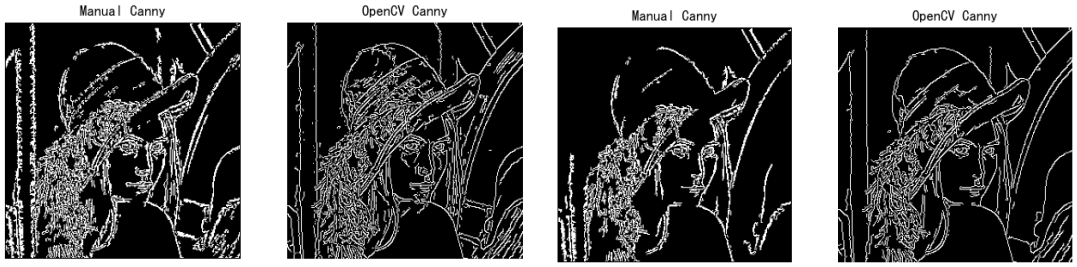
图 4

4.2 选取不同的双阈值获比较检测性能

从左至右，从上至下依次为（10，50），（30，100），（50，100）和（70，200）的高低阈值组合。



(a) 10-50 (b) 30-100



(a) 50-100 (b) 70-200

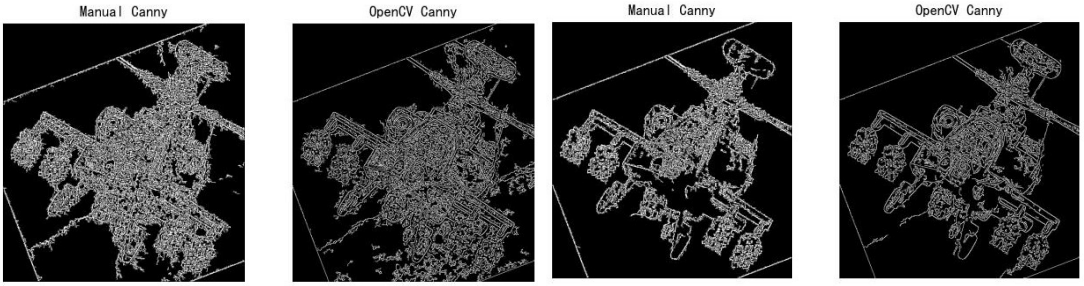
低阈值的作用是确定哪些弱边缘可以成为最终边缘。如果一个像素的梯度幅值大于低阈值但小于高阈值，这个像素被视为弱边缘。弱边缘的连接通常依赖于其邻域中是否存在强边缘。如果邻域

有强边缘，这个弱边缘就会被保留，成为最终的边缘。可以看出，较低的低阈值增加了更多的弱边缘，可能会导致过多的噪声被误认为是边缘。结果是图像中的边缘变得更加模糊且有噪声。较高的低阈值减少了弱边缘的数量，使得边缘更加锐利且清晰，但有可能会丢失一些微弱的边缘。

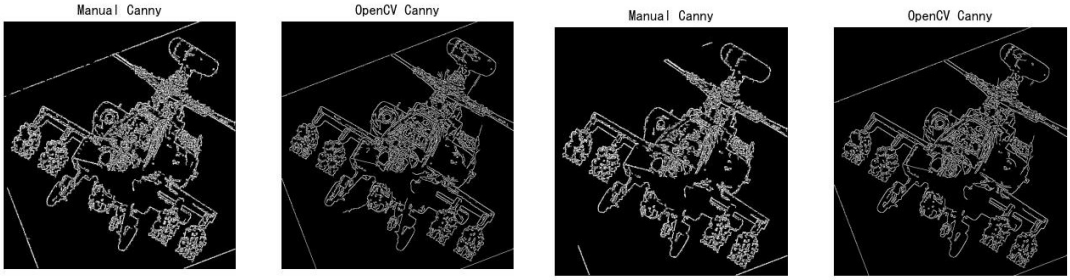
同理，较低的高阈值允许更多的像素被认为是强边缘，从而可能导致图像中出现更多的边缘细节，但也可能增加噪声。而较高的高阈值减少了强边缘的数量，使得最终图像中的边缘更为粗糙，可能会忽略细节部分，导致一些边缘被丢失。

可以看出，(70-200）的检测效果最好，边缘清晰，噪声较少。而（10-50）的噪声较多，边缘模糊，出现假边缘保留情况。

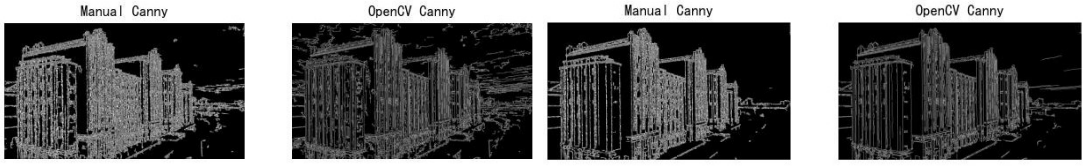
下图为示例 2，示例 3 的结果：



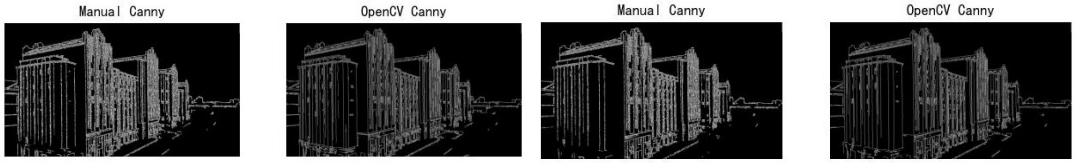
(a) 10-50　　(b) 30-100



(a) 50-150　　(b) 70-200



(a) 10-50　　(b) 30-100



(a) 50-150　　(b) 70-200

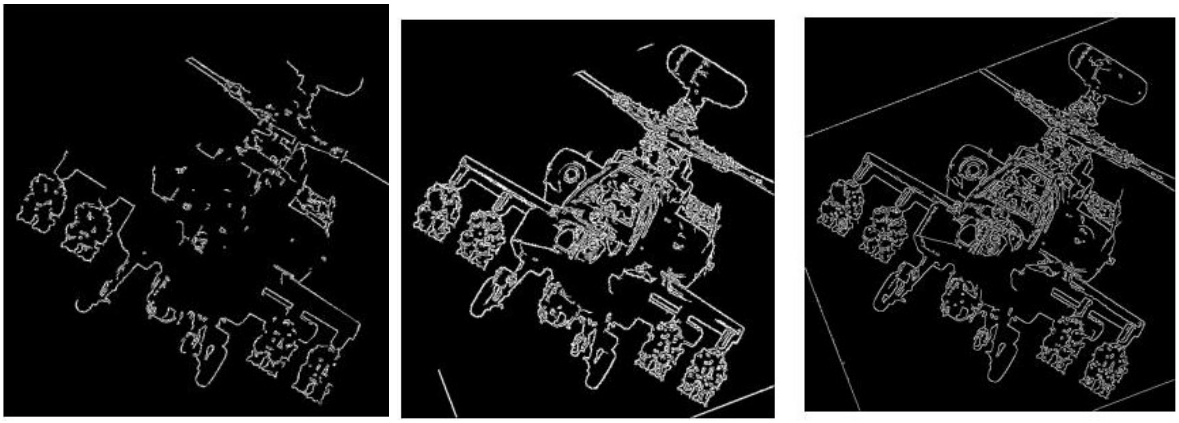
4.3 选取不同梯度幅值算子比较检测性能

这里采用了阈值参数为（70，200）的高低阈值组合，分别使用了 Roberts 算子、Prewitt 算子和 Canny 算子进行边缘检测（下图从左至右），算子定义见 3.3。

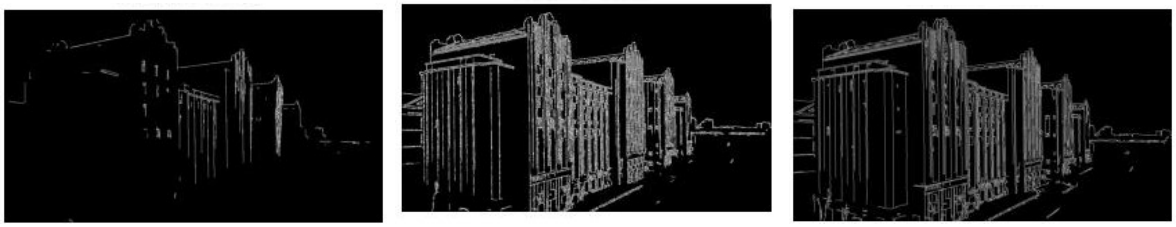
(a) Roberts 算子　　(b) Prewitt 算子　　(c) Canny 算子

可以看出，Roberts 算子的检测精度较低，仅能检测大致边缘，并且噪声抑制差，容易受噪声影响；Prewitt 算子的检测精度较高，能检测较大边缘，但仍有噪声影响；Canny 算子的检测精度最高，能检测到较小的边缘，且噪声抑制较好。

下图为示例 2，示例 3 的结果：



(a) Roberts 算子　　(b) Prewitt 算子　　(c) Canny 算子



(a) Roberts 算子　　(b) Prewitt 算子　　(c) Canny 算子

5 实验感想

图像边缘检测是计算机视觉中的一个基础任务，对许多后续任务（如物体检测、图像分割、目标跟踪等）至关重要。通过提取图像中的边缘，可以有效地提取出物体的轮廓和重要信息。

Canny 算法是经典的边缘检测算法，其主要目标是检测真实的边缘并抑制噪声。通过几个关键步骤（高斯滤波、梯度计算、非极大值抑制、高低阈值连接等），Canny 算法能够有效地找到图像中的边缘。

总的来说，这次实验不仅加深了我对图像边缘检测原理的理解，还让我掌握了如何使用梯度算子、滤波器等技术来处理图像，提升了我在图像处理和计算机视觉领域的实践能力。

6 源代码

6.1 使用 Sobel 算子进行边缘检测与 OpenCV 库函数的对比

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import os

# 更改工作目录到脚本所在目录
os.chdir(os.path.dirname(os.path.abspath(__file__)))

# 让直方图中有中文!
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['axes.unicode_minus'] = False  # 使用 ASCII 负号

def canny_edge_detection(image, low_threshold, high_threshold):
    """
    手动实现 Canny 边缘检测
    1. 转灰度
    2. 使用 Sobel 算子计算梯度
    3. 计算梯度幅值和方向
    4. 非极大值抑制
    5. 双阈值检测和边缘连接
    """
    # 1. 转为灰度图
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 2. 使用 Sobel 算子计算梯度
    sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
    magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    direction = np.arctan2(sobel_y, sobel_x) * (180 / np.pi)  # 梯度方向（角度）
    direction = (direction + 180) % 180  # 转为正角度

    # 3. 非极大值抑制
    nms = np.zeros_like(magnitude)
    rows, cols = gray.shape
    for r in range(1, rows - 1):
        for c in range(1, cols - 1):
            angle = direction[r, c]
            angle = angle % 180  # 将角度限制在0到180度之间

            if (0 <= angle < 22.5) or (157.5 <= angle <= 180):
```

```
                before = magnitude[r, c - 1]
                after = magnitude[r, c + 1]
            elif 22.5 <= angle < 67.5:
                before = magnitude[r - 1, c + 1]
                after = magnitude[r + 1, c - 1]
            elif 67.5 <= angle < 112.5:
                before = magnitude[r - 1, c]
                after = magnitude[r + 1, c]
            else:
                before = magnitude[r - 1, c - 1]
                after = magnitude[r + 1, c + 1]

            # 线性插值
            weight = (angle % 45) / 45.0
            interpolated_value = before * (1 - weight) + after * weight

            if magnitude[r, c] >= interpolated_value:
                nms[r, c] = magnitude[r, c]

    # 4. 双阈值检测
    edges = np.zeros_like(nms, dtype=np.uint8)
    strong = np.int32(nms > high_threshold)
    weak = np.int32((nms >= low_threshold) & (nms <= high_threshold))
    edges[strong == 1] = 255  # 强边缘
    edges[weak == 1] = 50  # 弱边缘（暂时标记）

    # 5. 弱边缘连接
    for r in range(1, rows - 1):
        for c in range(1, cols - 1):
            if edges[r, c] == 50:  # 弱边缘
                if 255 in edges[r - 1:r + 2, c - 1:c + 2]:  # 如果邻域中有强边缘
                    edges[r, c] = 255
                else:
                    edges[r, c] = 0  # 否则丢弃

    return edges

# 读取图像
image = cv2.imread("../../dataset/3.jpg")  # 替换为您的图片路径

# 设置低阈值和高阈值
low_threshold = 50
high_threshold = 150

# 手动实现 Canny 边缘检测
manual_edges = canny_edge_detection(image, low_threshold, high_threshold)

# 使用 OpenCV 自带的 Canny 方法
opencv_edges = cv2.Canny(image, low_threshold, high_threshold)

# 显示结果
plt.figure(figsize=(12, 6))
plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 3, 2)
plt.title("Manual Canny")
```

```
plt.imshow(manual_edges, cmap="gray")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.title("OpenCV Canny")
plt.imshow(opencv_edges, cmap="gray")
plt.axis("off")

plt.show()
```

6.2 选取不同梯度幅值算子比较检测性能

6.2.1 Roberts 算子

```
kernel_x = np.array([[1, 0], [0, -1]], dtype=int)
kernel_y = np.array([[0, 1], [-1, 0]], dtype=int)
grad_x = cv2.filter2D(gray, cv2.CV_64F, kernel_x)
grad_y = cv2.filter2D(gray, cv2.CV_64F, kernel_y)
edges = cv2.magnitude(grad_x, grad_y)
edges = np.uint8(edges)
```

6.2.2 Prewitt 算子

```
grad_x = cv2.filter2D(gray, cv2.CV_64F, np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]]))
grad_y = cv2.filter2D(gray, cv2.CV_64F, np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]]))
edges = cv2.magnitude(grad_x, grad_y)
edges = np.uint8(edges)
```

6.2.3 Canny 算子

```
edges = cv2.Canny(gray, low_threshold, high_threshold)  # 这里的阈值可以根据需要调整
magnitude = edges
direction = None  # Canny 算子不直接提供方向信息
```