lab2-Canny 边缘检测　实验报告

电院 2303 徐恺阳 523030910085

November 2024

1　实验概览

本实验内容如下：

1. 边缘检测的概念以及 Canny 算法原理；
2. OpenCV 实现 Canny 边缘检测。

其中，Canny 边缘检测的步骤如下：

- 使用高斯滤波器，以平滑图像，滤除噪声；
- 计算图像中每个像素点的梯度强度和方向；
- 应用非极大值（*Non-Maximum Suppression*）抑制，以消除边缘检测带来的杂散响应；
- 应用双阈值（*Double-Threshold*）检测来确定真实的和潜在的边缘；
- 通过抑制孤立的弱边缘最终完成边缘检测。

2　实验环境

本实验基于 Python 3.10，用到的库有 OpenCV、Numpy 和 Matplotlib。

3　练习一

3.1　题目描述

用 Sobel 算子对 dataset 文件夹中的图片进行 Canny 边缘检测，并与 OpenCV 库中自带 Canny 检测结果进行对比。

3.2　解题思路

下面根据上述步骤对练习一进行分析。

Step0: 读取图像并转为灰度图

这里直接使用 OpenCV 库中的函数进行转换，代码如下：

```python
def read_img(image_path):
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return gray
```

图 1: 读取图像并转为灰度图

Step1: 使用高斯滤波器，以平滑图像，滤除噪声

这里使用 5×5 的高斯滤波器核，并且 σ 取 1.4。

```python
def apply_gaussian_blur(gray, kernel_size=5, sigma=1.4):
    blurred = cv2.GaussianBlur(gray, (kernel_size, kernel_size), sigma)
    return blurred
```

图 2: 高斯模糊

Step2: 计算图像中每个像素点的梯度强度和方向

这里使用 *Sobel* 算子，代码如下：

```python
Gx = cv2.Sobel(blurred, cv2.CV_64F, 1, 0, ksize=3)
Gy = cv2.Sobel(blurred, cv2.CV_64F, 0, 1, ksize=3)
magnitude = np.sqrt(Gx**2 + Gy**2)
direction = np.arctan2(Gy, Gx)
```

图 3: 计算梯度强度和方向

Step3: 应用非极大值抑制（Non-Maximum Suppression）

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，部分代码如下：

```python
# 非极大值抑制
if magnitude[i, j] >= q and magnitude[i, j] >= r:
    nms[i, j] = magnitude[i, j]
else:
    nms[i, j] = 0
```

图 4: 非极大值抑制

注意： 这里利用梯度方向矩阵进行插值操作，准确性更高。

设 g_1 的梯度幅值 $M(g_1)$，g_2 的梯度幅值 $M(g_2)$，则 $dtmp1$ 可以很容易得到：

$$M(dtmp1) = (1 - w) \cdot M(g_2) + w \cdot M(g_1)$$

其中：

$$w = \frac{\text{distance}(dtmp1, g_2)}{\text{distance}(g_1, g_2)}$$

$\text{distance}(g_1, g_2)$ 表示两点之间的距离。

代码如下：

```
if (0 <= angle[i, j] < 22.5) or (157.5 <= angle[i, j] <= 180):
    q = magnitude[i, min(j + 1, cols - 1)] * (abs(angle[i, j] / 22.5)) + \
        magnitude[i, min(j + 2, cols - 1)] * (1 - abs(angle[i, j] / 22.5))
    r = magnitude[i, max(j - 1, 0)] * (abs(angle[i, j] / 22.5)) + \
        magnitude[i, max(j - 2, 0)] * (1 - abs(angle[i, j] / 22.5))
elif 22.5 <= angle[i, j] < 67.5:
    q = magnitude[min(i + 1, rows - 1), max(j - 1, 0)] * (abs((angle[i, j] - 45) / 22.5)) + \
        magnitude[min(i + 2, rows - 1), max(j - 2, 0)] * (1 - abs((angle[i, j] - 45) / 22.5))
    r = magnitude[max(i - 1, 0), min(j + 1, cols - 1)] * (abs((angle[i, j] - 45) / 22.5)) + \
        magnitude[max(i - 2, 0), min(j + 2, cols - 1)] * (1 - abs((angle[i, j] - 45) / 22.5))
elif 67.5 <= angle[i, j] < 112.5:
    q = magnitude[min(i + 1, rows - 1), j] * (abs((angle[i, j] - 90) / 22.5)) + \
        magnitude[min(i + 2, rows - 1), j] * (1 - abs((angle[i, j] - 90) / 22.5))
    r = magnitude[max(i - 1, 0), j] * (abs((angle[i, j] - 90) / 22.5)) + \
        magnitude[max(i - 2, 0), j] * (1 - abs((angle[i, j] - 90) / 22.5))
elif 112.5 <= angle[i, j] < 157.5:
    q = magnitude[max(i - 1, 0), max(j - 1, 0)] * (abs((angle[i, j] - 135) / 22.5)) + \
        magnitude[max(i - 2, 0), max(j - 2, 0)] * (1 - abs((angle[i, j] - 135) / 22.5))
    r = magnitude[min(i + 1, rows - 1), min(j + 1, cols - 1)] * (abs((angle[i, j] - 135) / 22.5)) +
        magnitude[min(i + 2, rows - 1), min(j + 2, cols - 1)] * (1 - abs((angle[i, j] - 135) / 22.5)
```

图 5: 插值操作

Step4: 应用双阈值（Double-Threshold）检测

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。

代码如下：

```
def double_threshold(nms, low_threshold, high_threshold):
    strong_edge = 255
    weak_edge = 50
    result = np.zeros_like(nms)

    strong_i, strong_j = np.where(nms >= high_threshold)
    weak_i, weak_j = np.where((nms >= low_threshold) & (nms < high_threshold))

    result[strong_i, strong_j] = strong_edge
    result[weak_i, weak_j] = weak_edge

    return result, strong_edge, weak_edge
```

图 6: 双阈值检测

Step5: 通过抑制孤立的弱边缘完成边缘检测

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，代码如下：

```
def hysteresis_tracking(result, strong_edge, weak_edge):
    rows, cols = result.shape
    for i in range(1, rows-1):
        for j in range(1, cols-1):
            if result[i, j] == weak_edge:
                if ((result[i+1, j-1:j+2] == strong_edge).any() or
                    (result[i-1, j-1:j+2] == strong_edge).any() or
                    (result[i, [j-1, j+1]] == strong_edge).any()):
                    result[i, j] = strong_edge
                else:
                    result[i, j] = 0
    return result
```

图 7: 边缘连接

3.3 代码运行结果

经过双阈值的调整，最终得出手搓的 Canny 检测和 OpenCV 自带的 Canny 检测得到的边缘检测结果，如图 8所示。

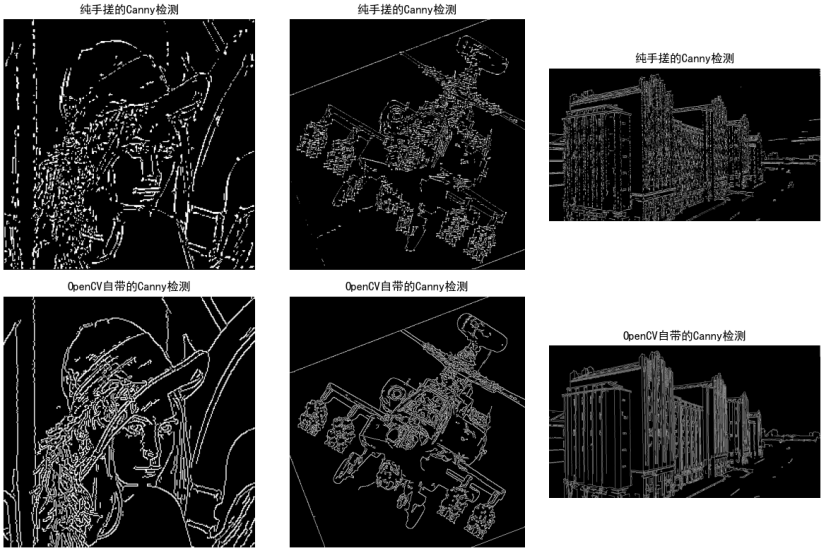

图 8: Canny 检测

3.4 分析与思考

对比手搓的 Canny 检测和 OpenCV 自带的 Canny 检测，发现 OpenCV 自带的 Canny 检测在细节处理方面优于手搓的，可能是因为梯度计算不够精确或者双阈值范围设置不够精确。

4 练习二

4.1 题目描述

选取不同的双阈值，比较检测性能。

4.2 解题思路

针对 1.jpg 分别选取三组不同的双阈值，得到三幅边缘检测结果，高低阈值分别为 [4, 10], [10, 25], [30, 75]。

4.3 代码运行结果

图 9: 不同的双阈值

4.4 分析与思考

可以看出，

- 双阈值较低时，检测结果中易出现多余的轮廓，显得杂乱；
- 双阈值较高时，检测结果中轮廓稀少，甚至无法闭合；
- 双阈值适中时，检测结果能够很好地框出的主体的轮廓。

5 练习三

5.1 题目描述

选取除 Sobel 外不同梯度幅值算子比较检测性能。

5.2 解题思路

针对 1.jpg 分别选取三组不同的梯度算子，得到三幅边缘检测结果，高低阈值取 [10, 25]。

5.3 代码运行结果

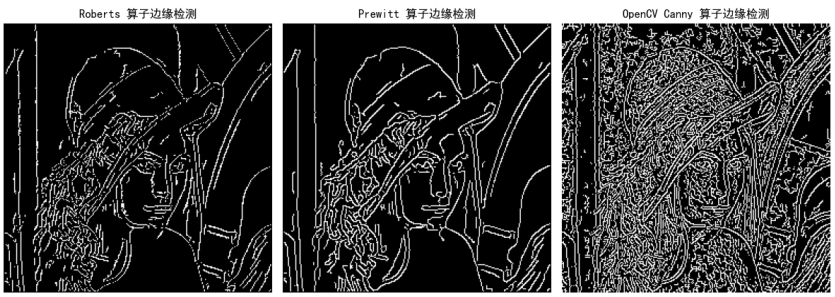

图 10: 不同的梯度算子

5.4 分析与思考

可以看出，相同的双阈值下，*Roberts* 算子边缘线条最稀疏，而 *Canny* 算子边缘线条最密集，说明对于最合适的双阈值，*Roberts* 算子最低，*Prewitt* 算子次之，*Canny* 算子最高。

同时，通过 *Prewitt* 算子计算梯度，得到的边缘检测结果中，边缘连接更加丝滑，可能 *Prewitt* 算子对梯度的计算更为精细。

6 实验感想

通过 lab2-Canny 边缘检测实验中，我了解了边缘检测的概念以及 *Canny* 算法原理，并尝试用 OpenCV 实现 Canny 边缘检测。同时，我对比了双阈值和梯度算子等参数对检测结果的影响，能够更好地进行参数的调整。

7 拓展思考

7.1 PPT 中对“线性插值法”的描述是否有误？

答：有误。应该将 w 和 $1-w$ 调换位置。