《生物医学图像处理实验》实验报告

(2022-2023 学年第 2 学期)

大作业：图像综合处理

课程号：310025010　　**课序号**：01　　**任课教师**：刘奇　林江莉　庄艳　　**成绩**：

姓名：徐广玄

实验报告提交日期：2023 年 6 月 26 日 （实验报告的报告日期预计：2023 年 6 月 27 日）

一、实验目的

充分利用所学各种图像处理技术，实现对图像的综合处理，加深对基础知识的理解和应用，从而进一步掌握彩色图像处理的知识和技术。

二、实验内容

在下列题目中任意选择一个题目完成实验。（我选择题目一）

1. 目标与背景的分割与提取 1

主要要求：提取红苹果

（1）将已知图像进行消噪处理

（2）对彩色图像进行目标和背景分析

（3）通过阈值法将图像进行分割

（4）提取目标

难点：

（1）确定目标区域的特征；　（2）边界修复与区域分割。

2. 形状检测认知

主要要求：实现图中九种形状检测。

（1）目标分割。图中目标与背景色彩及亮度差别较大，轮廓清晰，可以采用 Canny 边缘检测、形态滤波、区域填充的方式实现分割；

（2）特征参数提取，如色彩特征、边界特征、形状特征等；

（3）利用角点数、长宽比、圆度、色彩等特征区别各形状。

难点：

（1）选取合适的特征；　（2）特征参数比对的阈值选择。

三、实验考核

程序演示+实验报告

四、实验报告要求

1. 实验报告主体部分应包括：算法原理、程序流程、算法部分主要函数代码以及功能注释、运行结果及分析四部分。

2. 鼓励使用 Visual C++编写，鼓励和前面的实验整合成一个项目文件，鼓励形成可脱离编译环境的 exe 可执行文件。

3. 实验报告5000左右，8页以内。

4. 提交所有源代码。

五、实验方案

1. 程序流程：Original_display(image1, image2, Pixel)：此函数采用两个图像（image1 和 image2）和一个像素值作为输入。它分析第二个图像 (image2) 并根据红色像素的存在来定位感兴趣区域 (ROI)。然后，它通过在第一张图像 (image1) 中在 ROI 周围绘制黑色方块来突出显示 ROI。正方形的大小由像素参数决定。mean_filter_24bits(image, size)：该函数对 24 位 RGB 图像执行均值滤波操作。它以图像和过滤器大小作为输入。过滤器大小应该是两个相等的奇数整数的元组。该函数使用指定的滤波器大小将均值滤波器应用于图像并返回滤波后的图像。RGB2HSV(image)：此函数将 RGB 图像转换为 HSV 颜色空间。它将图像作为输入，并将 RGB 到 HSV 转换公式应用于每个像素。该函数返回 HSV 颜色空间中的图像。Threshold(image)：此函数对 HSV 颜色空间中的图像执行阈值处理。它将图像作为输入并应用特定的阈值范围进行红色检测。阈值范围内的像素设置为黑色 (0)，而范围之外的像素设置为白色 (1)。该函数返回阈值化的二值图像。侵蚀（图像，类型，次数）：此函数对二值图像执行侵蚀。它采用图像、侵蚀类型（“完整”或“交叉”）以及侵蚀迭代次数作为输入。该函数使用指定的类型和迭代次数对图像应用腐蚀，并返回腐蚀后的图像。Expansion(image, type, times)：该函数对二值图像执行扩展（膨胀）。它采用图像、扩展类型（“完整”或“交叉”）以及扩展迭代次数作为输入。该函数使用指定的类型和迭代次数对图像进行扩展，并返回扩展后的图像。最后，主函数按照特定的顺序调用这些函数来处理输入的红苹果图像。它在单独的子图中显示原始图像、均值滤波图像、转换为 HSV 颜色空间的图像、阈值二值图像、腐蚀图像和膨胀图像。

```python
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt  # plt 用于显示图片
import matplotlib.image as mpimg  # mpimg 用于读取图片
import numpy as np
```

图 1 仅使用读取图片的库，不使用图像处理的库

```python
# 读取目标路径
image_apple = mpimg.imread('./red_apple.bmp')


# 显示原图像
def original_display(image1, image2, pixel):...


# 均值滤波，24位(R, G, B)
def mean_filter_24bits(image, size):...


# RGB变HSV
def RGB2HSV(image):...


# 阈值分割
def threshold(image):...


# 腐蚀
def erosion(image, type, times):...


# 膨胀
def expansion(image, type, times):...
```

图 2 流程代码框架

```python
if __name__ == '__main__':
    img_filter = mean_filter_24bits(image_apple, (3, 3))
    img_HSV = RGB2HSV(img_filter)
    img_binary = threshold(img_HSV)
    img_erosion = erosion(img_binary, 'cross', 5)
    img_opening = expansion(img_erosion, 'cross', 3)
    original_display(image_apple, img_opening, 2)

    plt.tight_layout()
    plt.show()
```

图 3 主函数

2. 算法原理和解释：

2.1 均值滤波

对时域图像，有定义：

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

此时，我们可以理解为函数的高频低频被均匀分担，消除了噪点。

我们首先定义函数，输入量为原函数和核大小：

```python
def mean_filter_24bits(image, size):
    # 初始化变量
    img = image
    if size[0] == size[1] and size[0] % 2 == 1:
        size = size[0]
    else:
        print("错误的核大小！")
        exit()

    val = 1 / size ** 2
    kernel = np.full((size, size), val)
    extend = int(size / 2)

    img_height = img.shape[0]
    img_width = img.shape[1]
```

图 4 均值滤波函数初始化

然后我们使用五次 for 遍历图像的三个维度和核的两个维度，并利用公式累加：

```python
for c in range(3):
    for j in range(range_h):
        for k in range(range_w):
            for a in range(size):
                for b in range(size):
                    cnt += img_right[j+a][k+b][c] * val

            img_right[j+extend][k+extend][c] = int(cnt)

            if img_right[j+extend][k+extend][c] < 0:
                img_right[j+extend][k+extend][c] = 0
            elif img_right[j+extend][k+extend][c] > 255:
                img_right[j+extend][k+extend][c] = 255

            cnt = 0
```

图 5 均值滤波函数主体

2.2 RGB 空间转换为 HSV 空间

对于 RGB 空间，我们人类很难确定阈值，但是对于 HSV 空间，人类很容易主观确定阈值，对于 RGB 空间转换为 HSV 空间公式，有：

$$R' = R/255$$
$$G' = G/255$$
$$B' = B/255$$
$$C\max = \max(R', G', B')$$
$$C\min = \min(R', G', B')$$
$$\Delta = C\max - C\min$$

H 计算：

Hue

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} + 0\right) & , C\max = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C\max = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C\max = B' \end{cases}$$

S 计算：

Saturation

$$S = \begin{cases} 0 & C\max = 0 \\ \frac{\Delta}{C\max} & C\max \neq 0 \end{cases}$$

图 6 RGB 空间转换为 HSV 空间公式

对于代码，有对应的矩阵运算：

```
def RGB2HSV(image):
    img = image

    R = img[:, :, 0] / 255
    G = img[:, :, 1] / 255
    B = img[:, :, 2] / 255
    img_height = img.shape[0]
    img_width = img.shape[1]
    max_matrix = np.zeros((img_height, img_width))
    min_matrix = np.zeros((img_height, img_width))
    H = np.zeros((img_height, img_width))
    S = np.zeros((img_height, img_width))
    V = np.zeros((img_height, img_width))
    delta = np.zeros((img_height, img_width))

    for i in range(img_height):
        for j in range(img_width):
            max_matrix[i, j] = max(R[i, j], G[i, j], B[i, j])
            min_matrix[i, j] = min(R[i, j], G[i, j], B[i, j])
            delta[i, j] = max_matrix[i, j] - min_matrix[i, j]
```

图 7 RGB 空间转换为 HSV 空间函数初始化

```
    # 计算V
    V = max_matrix

    # 计算S
    for i in range(img_height):
        for j in range(img_width):
            if max_matrix[i, j] == 0:
                S[i, j] = 0
            else:
                S[i, j] = delta[i, j] / max_matrix[i, j]

    # 计算H
    for i in range(img_height):
        for j in range(img_width):
            if max_matrix[i, j] == R[i, j]:
                H[i, j] = (0 + (G[i, j] - B[i, j]) / delta[i, j]) / 6
            elif max_matrix[i, j] == G[i, j]:
                H[i, j] = (2 + (G[i, j] - B[i, j]) / delta[i, j]) / 6
            elif max_matrix[i, j] == B[i, j]:
                H[i, j] = (4 + (G[i, j] - B[i, j]) / delta[i, j]) / 6

            if H[i, j] < 0:
                H[i, j] += 360
```

图 8 RGB 空间转换为 HSV 空间主函数

2.3 阈值分割

对 HSV 颜色空间中的图像执行阈值处理。它将图像作为输入并应用特定的阈值范围进行红色检测。阈值范围内的像素设置为黑色，而范围之外的像素设置为白色，对应代码有：

```
# 阈值分割
def threshold(image):
    lower_red = np.array([20, 20, 70]) / 255
    upper_red = np.array([190, 255, 255]) / 255
    img = image
    img_height = img.shape[0]
    img_width = img.shape[1]
    img_binary = np.zeros((img_height, img_width))

    for i in range(img_height):
        for j in range(img_width):
            if lower_red[0] <= img[i, j, 0] <= upper_red[0] and lower_red[1] <= img[i, j, 1] <= upper_red[1] and lower_red[2] <= img[i, j, 2] <= upper_red[2]:
                img_binary[i, j] = 0
            else:
                img_binary[i, j] = 1
```

图 9 阈值分割主函数

2.4 开操作

2.4.1 腐蚀

对于腐蚀操作，可以理解为，移动结构 B，如果结构 B 与结构 A 的交集完全属于结构 A 的区域内，则保存该位置点，所有满足条件的点构成结构 A 被结构 B 腐蚀的结果，有原理图：

图 10 腐蚀操作原理图

2.4.2 膨胀

对于膨胀操作，可以理解为将结构 B 在结构 A 上进行卷积操作，如果移动结构 B 的过程中，与结构 A 存在重叠区域，则记录该位置，所有移动结构 B 与结构 A 存在交集的位置的集合为结构 A 在结构 B 作用下的膨胀结果，有原理图：

图 11 膨胀操作原理图

2.4.3 开操作

先腐蚀后膨胀的操作称之为开操作，有原理图：

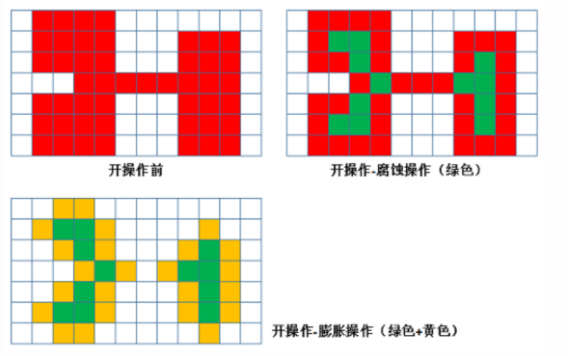

图 11 开操作原理图

考虑到阈值分割后噪声很大，故采用开操作，对应不同核，有代码（腐蚀和膨胀类似，处于篇幅原因只显示腐蚀）：

```
if type == 'full':
    for i in range(times):
        for j in range(range_h):
            for k in range(range_w):
                for a in range(size):
                    if isErosion == 1:
                        break
                    for b in range(size):
                        if img_copy[j+a-1][k+b-1] == 0:
                            isErosion = 1
                            img_right[j, k] = 0
                            break
                isErosion = 0
        img_copy = np.copy(img_right)

if type == 'cross':
    for i in range(times):
        for j in range(range_h):
            for k in range(range_w):
                if img_copy[j-1][k] == 1 and img_copy[j][k] == 1 and img_copy[j][k-1] == 1 and img_copy[j+1][k] == 1 and img_copy[j][k+1] == 1:
                    img_right[j, k] = 1
                else:
                    img_right[j, k] = 0
        img_copy = np.copy(img_right)
```

图 12 腐蚀代码

2.5 框定原图像

对于图像阈值，找到四方极值点，做直线，有代码：

```
for i in range(img_height - 10):
    for j in range(img_width - 10):
        if img2[i, j] == 1 and i < y_top:
            y_top = i
        elif img2[i, j] == 1 and i > y_bottom:
            y_bottom = i
        if img2[i, j] == 1 and j < x_left:
            x_left = j
        elif img2[i, j] == 1 and j > x_right:
            x_right = j

for i in range(x_left, x_right):
    for j in range(y_top-pixel_width, y_top+pixel_width):
        img_copy[j, i, :] = 0
for i in range(x_left, x_right):
    for j in range(y_bottom-pixel_width, y_bottom+pixel_width):
        img_copy[j, i, :] = 0
for i in range(y_top, y_bottom):
    for j in range(x_left-pixel_width, x_left+pixel_width):
        img_copy[i, j, :] = 0
for i in range(y_top, y_bottom):
    for j in range(x_right-pixel_width, x_right+pixel_width):
        img_copy[i, j, :] = 0
```

图 13 画框代码

六、实验结果及分析

对于本次实验，我们可以看到二值化噪声很大，但是经过开操作后，可以发现噪声消失，二值化效果显著，图一中画框效果也非常好。

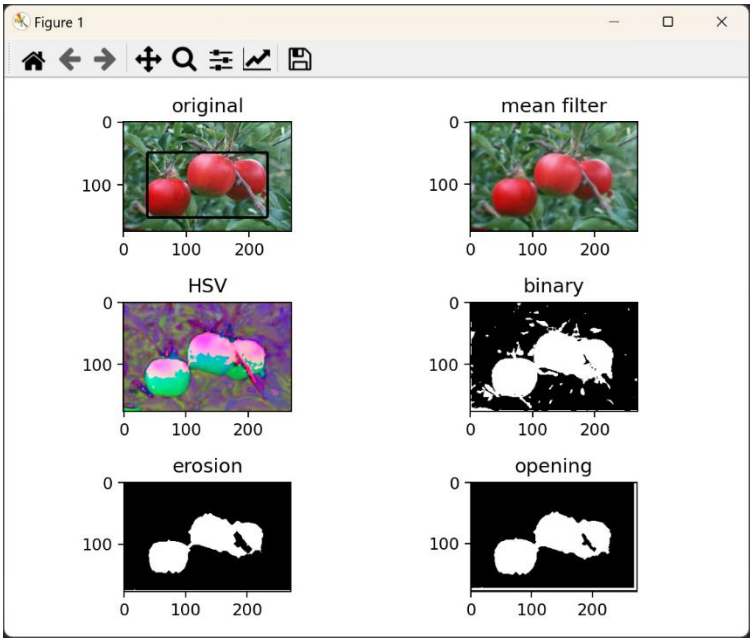

图 13 结果界面

七、实验总结：

本次实验室本学期医学图像处理的最后一次实验，在这七次实验中，我都是独自完成所有代码，对基础知识的掌握和代码水平突飞猛进，对此我很感谢老师们和助教哥哥。回到本次实验的总结，我觉得这次实验并非是最难的那个，但是是最应用和综合的一题，我们从现实出发，做出一个红色阈值识别的程序，从彩色图像的滤波，到色彩空间的转换，到二值化，到去除噪声，到框定范围，我了解到了一整个目标识别的流程，同时最终效果也非常喜人，不使用 openCV 对我们的底层学习帮助无疑是巨大的，我在其中收获了很多，我打算在硕士阶段跨申生物医学工程，这门课也为我打下了良好的基础，再次感谢各位看到这里的老师和助教。