云南大学

本科实验报告

课程名称：　图像理解与计算机视觉

实验名称：　实验五. 图像分割实验

学院（系）：　信息学院

专　　业：

班　　级：

姓　　名：

学　　号：

指导教师：

成　　绩：

评　　语：

Steven

一. 实验目的

通过编程实现使学生能够掌握常用的图像分割方法，包括正交梯度算子法、方向梯度算子法、二阶导数算子法等边缘点检测方法，边缘线跟踪方法，及区域生长等分割方法。

二. 实验内容

（1）分别采用Roberts梯度算子法、各向同性Sobel算子法、基尔希方向梯度算子法、8邻域拉普拉斯算子法编程实现下图边缘点检测，其中基尔希方向梯度算子法给出8个方向的梯度结果图；

（2）基于8邻域拉普拉斯算子法梯度计算结果，采用光栅扫描跟踪法对（1）中图像实现边缘线检测；

（3）采用迭代阈值法对下图实现图像分割；

（4）采用质心生长法对下图实现图像分割。

三. 实验环境

Matlab软件是图像处理领域广泛使用的仿真软件之一。本实验基于Matlab 2022a版本完成。

四. 实验代码（详细注释，Times New Roman/宋体 五号字体 单倍行距）

主函数

```
%% 使用: 在命令行内调用函数 Exp5(func)，func 是不同功能的名字。要将文件和.m 文件放在同一路径下。
%% 每题的 demo 调用格式如下:
% 1. 各种边缘点检测方法: Exp5("Ques1");
% 2. 使用光栅扫描跟踪法的边缘线检测: Exp5("Ques2");
% 3. 使用迭代阈值法进行图像分割: Exp5("Ques3");
% 4. 使用质心生长法进行图像分割: Exp5("Ques4");
function Exp5(ques)

    if ques == "Ques1"
        img = im2double(imread("cameraman.tif"));
        img_edge_Roberts = Ques1(img, "Roberts", 0.05);
        img_edge_IsotropySobel = Ques1(img, "IsotropySobel", 0.05);
        img_edge_8_Laplacian = Ques1(img, "8-Laplacian", 0.2);
        img_edge_Kirschs = Ques1(img, "Kirsch", 0.8);
        % 画图
        subplot(3, 4, 1); imshow(img); title("原图");
        subplot(3, 4, 2); imshow(img_edge_Roberts); title("Roberts 算子，阈值 0.05");
        subplot(3, 4, 3); imshow(img_edge_IsotropySobel); title("各向同性 Sobel 算子，阈值 0.05");
        subplot(3, 4, 4); imshow(img_edge_8_Laplacian); title("8 邻域 Laplacian 算子，阈值 0.2");
        for i = 1:8 % 画 Kirsch 8 方向梯度滤波的图
```

```
                subplot(3, 4, i + 4); imshow(img_edge_Kirschs(:, :, i)); title("Kirsch 方向梯
度，偏转方向: "+num2str(45 * (i - 1))+"°");
            end

        elseif ques == "Ques2"
            img = imread("cameraman.tif");
            img = im2double(img);
            new_img = Ques2(img, 0.2, 0.2);
            imshow(new_img);

        elseif ques == "Ques3"
            img = imread("building.tif");
            img = im2double(img);
            img_res = Ques3(img);
            imshow(img_res);

        elseif ques == "Ques4"
            img = imread("rose_form.tif");
            img = img(:, :, 1);
            new_img = Ques4(img, 98);
            subplot(2, 1, 1); imshow(img); title("原图");
            subplot(2, 2, 3); imshow(new_img(:, :, 1)); title("前景图");
            subplot(2, 2, 4); imshow(new_img(:, :, 2)); title("背景图");
        end

end
```

功能函数1：边缘点检测

```
%% Ques1: 边缘点检测
% Input/输入:
%     img: 二维矩阵，表示灰度图像
%     filter: 字符串，选择卷积类型，可选类型为"Roberts"、"IsotropySobel"、"8-
Laplacian"、"Kirsch"
%     threshold: 阈值，用于在二值化中进行判定
% Output/输出:
%     new_img: 2 维灰度矩阵（选择 filter == "Kirsch"时是 3 维灰度矩阵），边缘检测结
果
function new_img = Ques1(img, filter, threshold)
    [width, height] = size(img);
    new_img = zeros(size(img));
    % 构建卷积核
    if filter == "Roberts" % Roberts 算子
        kernel_h = [-1, 0, 0; 0, 1, 0; 0, 0, 0]; % 水平方向卷积核
        kernel_v = [0, -1, 0; 1, 0, 0; 0, 0, 0]; % 垂直方向卷积核
    elseif filter == "IsotropySobel" % 各向同性 Sobel 算子
```

```
            kernel_h = 1 / (2 + sqrt(2)) * [-1, 0, 1; -sqrt(2), 0, sqrt(2); -1, 0, 1];
            kernel_v = 1 / (2 + sqrt(2)) * [-1, -sqrt(2), -1; 0, 0, 0; 1, sqrt(2), 1];
        elseif filter == "8-Laplacian" % 8 邻域 Laplacian 算子
            kernel = [-1, -1, -1; -1, 8, -1; -1, -1, -1]; % 8 邻域拉普拉斯算子
        elseif filter == "Kirsch" % 8 方向 Kirsch 算子
            new_img = zeros([size(img), 8]); % 8 方向 Kirsch 算子需要生成 8 张图，所以重新定义返回值矩阵为 8 维
            origin_kernel = [-3, -3, 5; -3, 0, 5; -3, -3, 5];
            kernel = zeros(3, 3, 8);
            for i = 1:8 %  算出 8 个方向的卷积核
                kernel(:, :, i) = imrotate(origin_kernel, 45 * (i - 1), 'crop');
            end
        else
            disp("边缘点检测算子有误");
            return
        end
        %  进行边缘点检测
        m = 3; n = 3; %  卷积核长宽
        for i = floor(m / 2) + 1:floor(width - m / 2) + 1 % i,j 即中心点
            for j = floor(n / 2) + 1:floor(height - n / 2) + 1
                if filter == "Roberts" || filter == "IsotropySobel" % Roberts 算子和各向同性 Sobel 算子
                    val_h = sum(sum(img(i - floor(m / 2):i + floor(m / 2), j - floor(n / 2):j + floor(n / 2)) .* kernel_h));
                    val_v = sum(sum(img(i - floor(m / 2):i + floor(m / 2), j - floor(n / 2):j + floor(n / 2)) .* kernel_v));
                    new_img(i, j) = (val_h > threshold)    ||    (val_v > threshold); %  合并两个方向的结果
                elseif filter == "8-Laplacian" % 8 邻域拉普拉斯算子
                    val = sum(sum(img(i - floor(m / 2):i + floor(m / 2), j - floor(n / 2):j + floor(n / 2)) .* kernel));
                    new_img(i, j) = val > threshold;
                elseif filter == "Kirsch" % 8 方向 Kirsch 算子
                    for k = 1:8 %  依次计算 8 个方向
                        val = sum(sum(img(i - floor(m / 2):i + floor(m / 2), j - floor(n / 2):j + floor(n / 2)) .* kernel(:, :, k)));
                        new_img(i, j, k) = val > threshold;
                    end
                end
            end
        end
    end
```

功能函数2：使用光栅扫描跟踪法的边缘线检测

```
%% Ques2:  使用光栅扫描跟踪法的边缘线检测
```

```
% Input/输入:
%     img: 二维矩阵，表示灰度图像
%     threshold_detect: 检测门限（阈值）
%     threshold_track: 跟踪门限（阈值）
% Output/输出:
%     new_img: 2 维灰度矩阵，边缘检测结果
function new_img = Ques2(img, threshold_detect, threshold_track)
    new_img = zeros(size(img));
    [width, height] = size(img);
    kernel = [-1, -1, -1; -1, 8, -1; -1, -1, -1]; % 8 邻域拉普拉斯算子
    m = 3; n = 3; %  卷积核长宽为 3
    %  标记所有的检测点
    for i = floor(m / 2) + 1:floor(width - m / 2) + 1 % i,j 即中心点
        for j = floor(n / 2) + 1:floor(height - n / 2) + 1
            val = sum(sum(img(i - floor(m / 2):i + floor(m / 2), j - floor(n / 2):j + floor(n / 2)) .* kernel));
            new_img(i, j) = val > threshold_detect; %  高于检测门限，则进行标记
        end
    end
    %  逐行扫描，检测其下一行的三个邻接像素是否满足跟踪门限
    for i = floor(m / 2) + 1:floor(width - m / 2) + 1 % i,j 即中心点
        for j = floor(n / 2) + 1:height - 1
            if new_img(i, j) == 1
                new_img(i + 1, j - 1) = abs(img(i, j) - img(i + 1, j - 1)) >= threshold_track;
                new_img(i + 1, j) = abs(img(i, j) - img(i + 1, j)) >= threshold_track;
                new_img(i + 1, j + 1) = abs(img(i, j) - img(i + 1, j + 1)) >= threshold_track;
            end
        end
    end
end
```

功能函数3：使用迭代阈值法进行图像分割

```
%% Ques3:  使用迭代阈值法进行图像分割
% Input/输入:
%     img:  二维矩阵，表示灰度图像
% Output/输出:
%     new_img: 2 维灰度矩阵，图像分割结果
function new_img = Ques3(img)
    T = mean2(img); %取均值作为初始阈值
    flag = false; %  是否停止迭代
    i = 0;
    % while 循环进行迭代
    while ~flag
        renge1 = find(img <= T); %小于阈值的部分
        renge2 = find(img > T); %大于阈值的部分
```

```
        T_temp = (mean(img(renge1)) + mean(img(renge2))) / 2; %计算分割后两部分的阈值均值的均值
        flag = (abs(T_temp - T) < 1) || (i >= 30); % 收敛或达到迭代次数上限
        T = T_temp; % 更新 T 的值
        i = i + 1;
    end
    new_img(renge1) = 0; %将小于阈值的部分赋值为 0
    new_img(renge2) = 1; %将大于阈值的部分赋值为 1
    new_img = reshape(new_img, size(img)); % 上面两行代码是一维操作，这里将一维向量重新转为二维矩阵
end
```

功能函数4：使用质心生长法进行图像分割

```
%% Ques4: 使用质心生长法进行图像分割
% Input/输入:
%     img: 灰度图矩阵
%     T: 门限
% Output/输出:
%     new_imgs: 3 维灰度矩阵，分别表示前景图和背景图
function new_imgs = Ques4(img, T)
    img = double(img);
    [width, height] = size(img);
    obj = zeros(size(img)); obj(64, 64) = 1; %种子点
    new_imgs = uint8(zeros(width, height, 2));
    flag = true;
    % 迭代
    while flag
        flag = false; % 是否继续迭代
        for i = 2:width - 1
            for j = 2:height - 1
                if obj(i, j) == 1
                    for x = -1:1 % 生长过程
                        for y = -1:1
                            if obj(i + x, j + y) == 0 && abs(img(i + x, j + y) - sum(sum(img .* obj)) / sum(sum(obj))) <= T
                                flag = true;
                                obj(i + x, j + y) = 1;
                            end
                        end
                    end
                end
            end
        end
    end
    new_imgs(:, :, 1) = uint8(img .* obj);
```

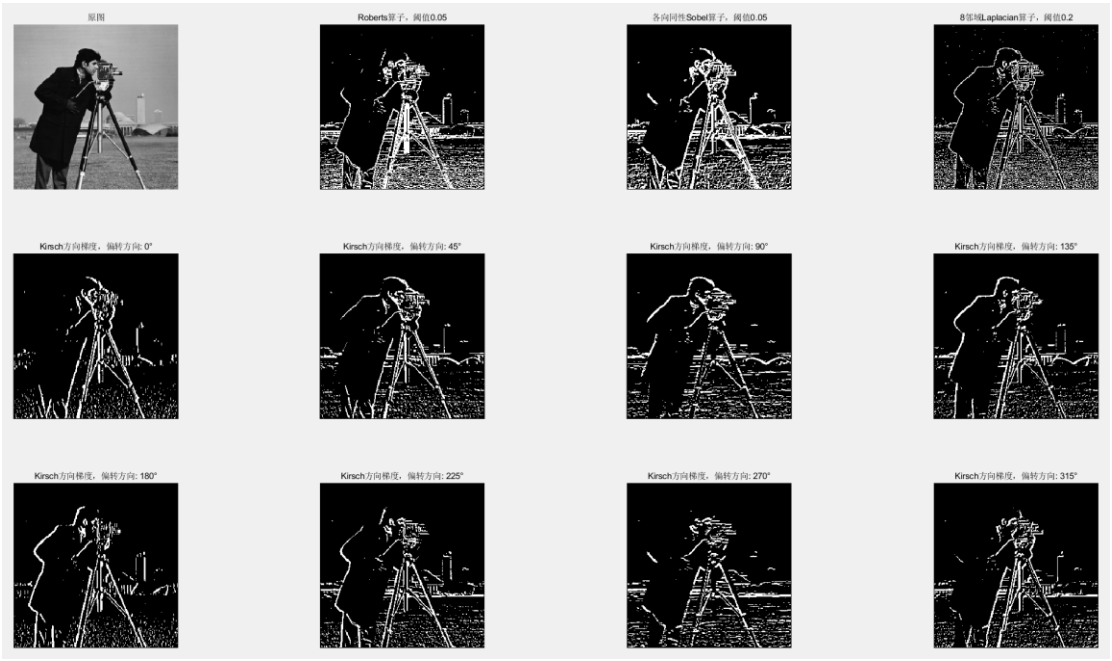
```
        new_imgs(:, :, 2) = uint8((-obj + 1) * 255);
end
```

Exp5.m

附件（.m文件）：

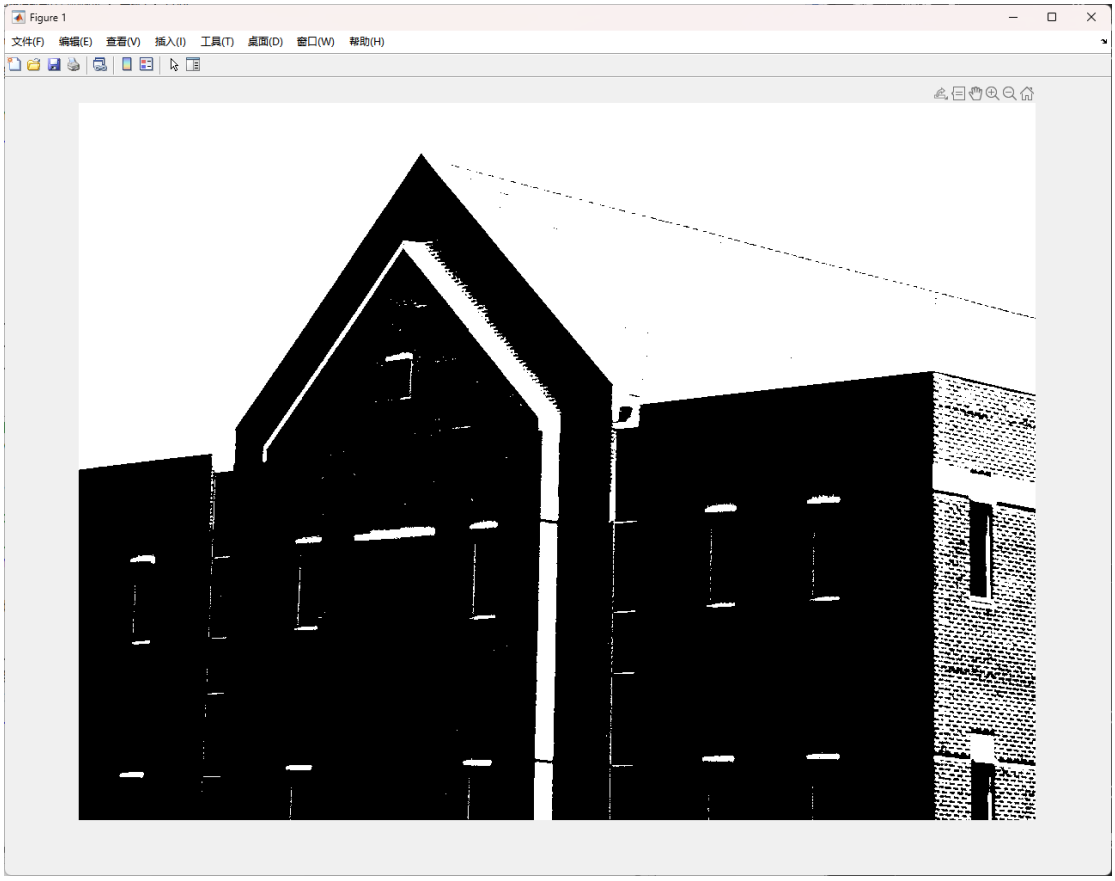
五. 实验结果

（1）Roberts梯度算子法、各向同性Sobel算子法、基尔希方向梯度算子法、8邻域拉普拉斯算子法边缘点检测结果，其中基尔希方向梯度算子法给出8个方向的梯度结果图；

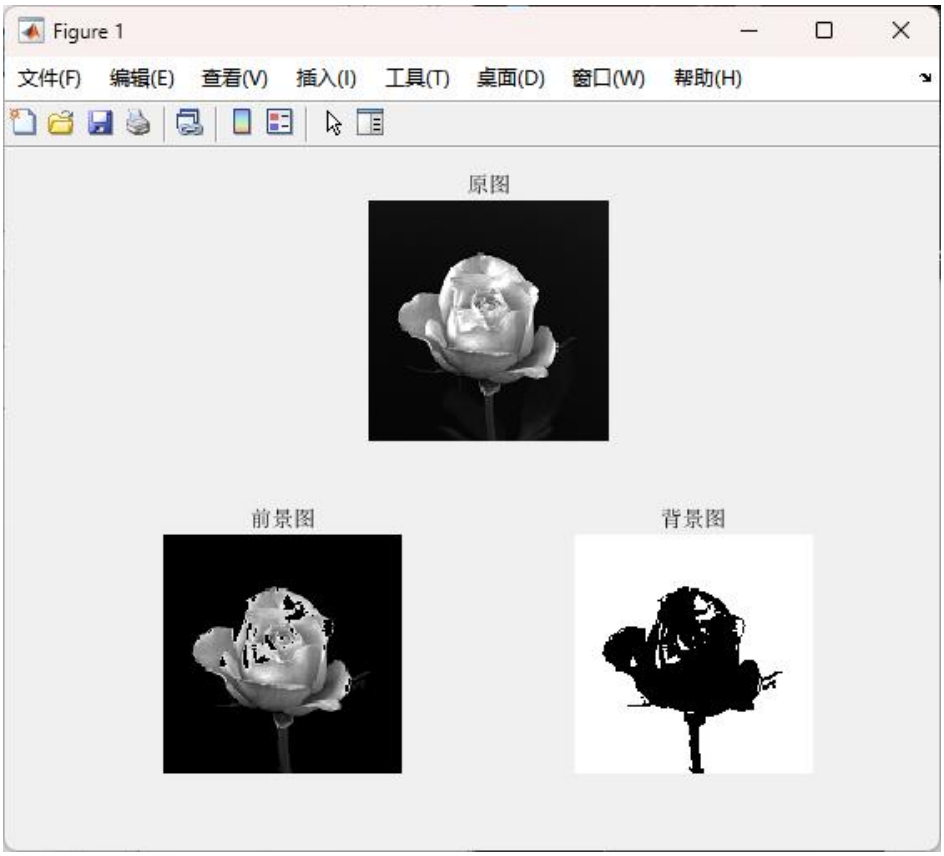


（2）光栅扫描跟踪法检测边缘线结果图；

（3）迭代阈值法进行图像分割的结果图；



（4）质心生长法进行图像分割的结果图。



六. 结果分析及体会

在这次实验的过程中，我越发意识到“阈值”设定的重要性。一个好的阈值不仅能带来好的处理效果，同时也往往能够显著降低实际运算次数。在本次实验中，阈值往往是经过反复测试找到的一个较优解，也就是经验值。但是课堂上也讲了很多动态确定阈限的方法，例如本次第3题，便是反复使用二分的思想来确定分块阈限。这也正是算法的魅力所在吧。