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：

2022年 11 月 15 日

**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。

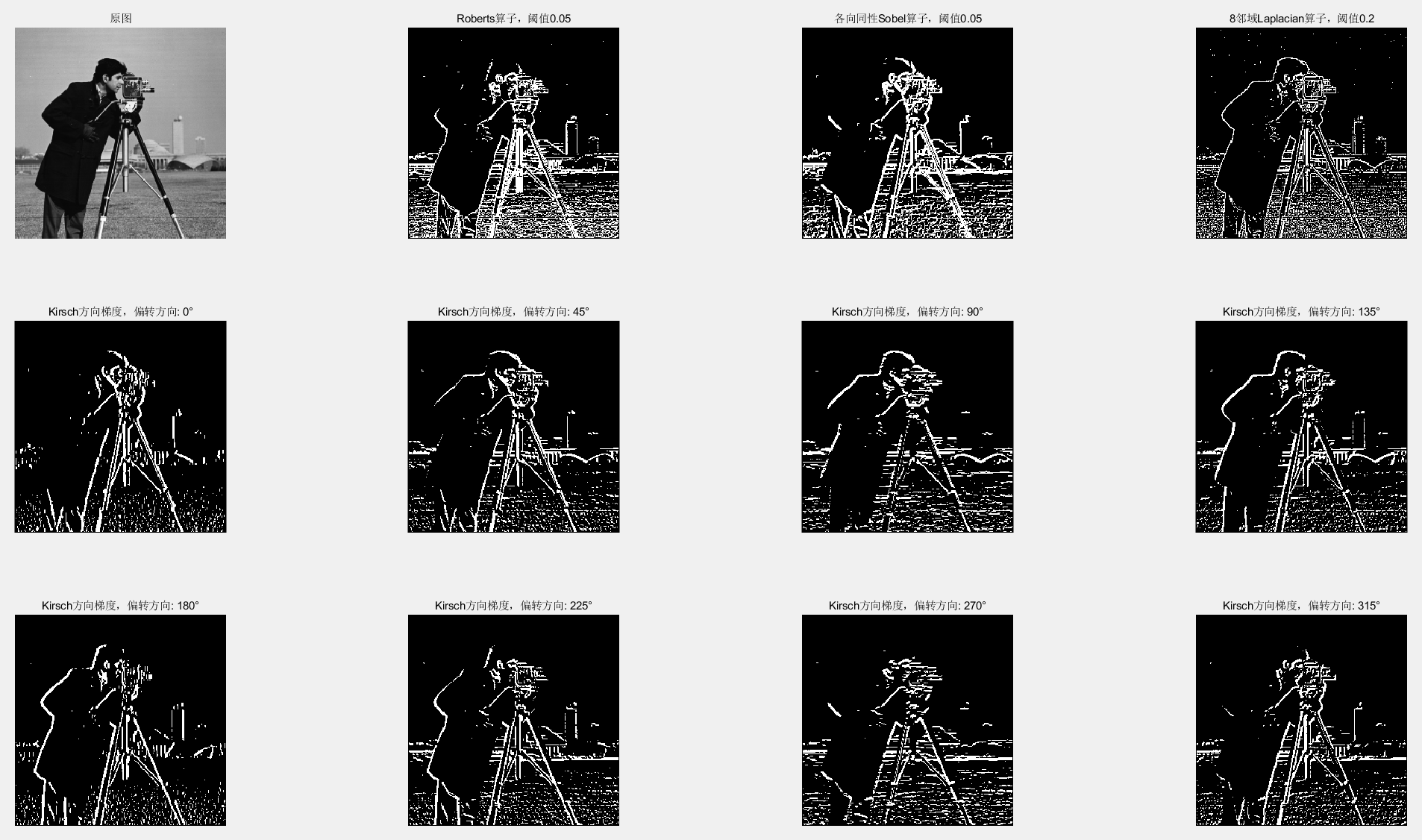
**四. 实验代码**（详细注释，Times New Roman/宋体 五号字体 单倍行距）

|  |
| --- |
| 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 |

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；

图形用户界面

中度可信度描述已自动生成

（3）迭代阈值法进行图像分割的结果图；

工程绘图

描述已自动生成

（4）质心生长法进行图像分割的结果图。

图形用户界面, 日程表

描述已自动生成

**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。