

**“数字图像处理基础”实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 题目 | 实验八：基于边缘的图像分割 | | |
| 姓名 | 22211374 | 学号 | 钟军凯 |
| 指导教师 | 彭亚辉 | | |
| 日期 | 2025年4月19日 | | |

**电子信息工程学院**

目 录

[1. 实验目的分析 3](#_Toc191473896)

[2. 实验方案设计 3](#_Toc191473897)

[3. 实验结果 4](#_Toc191473898)

[4. 分析与讨论 7](#_Toc191473899)

[5. 问题发现与探究 9](#_Toc191473900)

[6. 实验总结 9](#_Toc191473901)

[指导老师评价 1](#_Toc191473902)6

1. 实验目的分析

本实验旨在通过实际操作理解图像分割的基本原理，学习如何利用边缘检测技术将图像中的目标区域分离出来。首先，学习基于边缘检测的图像分割方法，并尝试对指定图像进行区域划分；其次，通过实验观察不同边缘检测算法（如Sobel算子与Canny算子）在效果上的差异，分析其特点；最后，将使用Python编程语言完整实现基于边缘检测的图像分割过程。

1. 实验方案设计
   1. **Sobel边缘检测**

Sobel边缘检测可以通过Sobel算子与图像进行二维卷积完成，实验方案在实验七中已经有详细的描述，此处不再赘述。代码方面，Sobel边缘检测的关键部分直接使用滤波器实验中的Sobel滤波器函数。

* 1. **Canny边缘检测**

关于Canny边缘检测，由于Canny边缘检测实现起来具有一定难度，所以考虑直接使用OpenCV的Canny边缘检测函数。如图1所示为Canny边缘检测流程图，此处给出不使用OpenCV的Canny边缘检测的具体实现过程：

首先，进行高斯滤波平滑图像，使用高斯滤波器核函数与图像进行二维卷积，直接使用滤波器实验中的高斯滤波器完成。

第二步，使用Sobel算子，将图像的粗边缘检测出来，同时也得到x和y方向的梯度，使用滤波器实验中的Sobel滤波器完成。因为需要x和y方向的梯度，又因为滤波器实验中的Sobel滤波器函数只返回了滤波后的图像，所以稍作修改，使其同时返回x和y方向的梯度。

第三步，进行非极大值抑制，根据每个像素点的x和y方向的梯度，得到梯度方向，采用线性插值，得到像素点在正负梯度方向上的相邻点，将像素点与相邻点进行比较，如果是极大值，则保留，不然置0。

第四步，设置两个阈值，一个是较大阈值，另一个是较小阈值，像素值大于较大的阈值，则保留并设置为强边缘，处于两个阈值之间，则保留并设置为弱边缘，小于较小的阈值，则舍弃该像素点。

第五步，对于被保留下来的像素点，一种是强边缘，另一种是弱边缘。此时需要找出弱边缘中与强边缘相连的像素点，将其升级为强边缘，其他没有与强边缘相连的弱边缘则被舍弃。遍历所有强边缘像素点，对于一个强边缘像素点，以其为中心，向外扩张，扩张第一层，也就是邻域的8个像素点，如果有弱边缘点，则升级为强边缘点，然后扩张到第二层，如果没有弱边缘点，则停止；对于第二层，寻找有无与上一层的强边缘点（包括升级后的弱边缘点）相邻的弱边缘点或者与本层强边缘点（如果本层有强边缘点）相邻的弱边缘点，如果有，则将弱边缘点升级为强边缘点，然后扩张到第三层，如果没有，则停止；对于第三层，根据之前的过程，以此类推，继续进行，直到扩张到了某一层符合其中一条结束条件，结束条件为：这一层没有弱边缘点和强边缘点；这一层没有强边缘点但是有弱边缘点，但是没有与上一层的强边缘点相邻。此时结束搜索。注意，如果一个强边缘点是由弱边缘点升级而来的，则不会参与遍历。这个流程类似于广度优先搜索。如图2所示为连通性验证的过程。

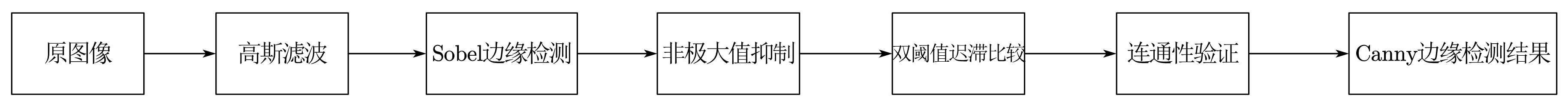


图 1：Canny边缘检测过程

在GUI设计方面，使用两个按键分别控制Sobel边缘检测和Canny边缘检测，使用滑动块来控制两种边缘检测的相关参数。对于Sobel边缘检测，使用一个滑动块来控制Sobel算子的大小；对于Canny边缘检测，使用三个滑动块来控制两个阈值和Sobel算子的大小。

1. 实验结果

如图2所示为不同参数的sobel边缘检测的结果。

|  |  |
| --- | --- |
|  |  |
|  |  |

**图 2**：sobel边缘检测结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）sobel算子大小为3的分割图像；（3）sobel算子大小为5的分割图像；（4）sobel算子大小为7的分割图像

如图3所示为不同参数的canny边缘检测的结果。

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

**图 3**：canny边缘检测结果。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）sobel算子大小等于3、较小阈值等于50、较大阈值等于150；（3）sobel算子大小等于3、较小阈值等于30、较大阈值等于150；（4）sobel算子大小等于3、较小阈值等于50、较大阈值等于200；（5）sobel算子大小等于3、较小阈值等于100、较大阈值等于150；（6）sobel算子大小等于3、较小阈值等于50、较大阈值等于100；（7）sobel算子大小等于3、较小阈值等于200、较大阈值等于255；（8）sobel算子大小等于5、较小阈值等于50、较大阈值等于150；（9）sobel算子大小等于7、较小阈值等于50、较大阈值等于150

如图4所示为基于阈值的图像分割和基于边缘的图像分割的对比，以femur图像为例。

|  |  |
| --- | --- |
|  |  |
|  |  |

**图 4**：基于阈值和基于边缘的图像分割比较。按从左到右、从上到下的顺序，从第1行开始依次编号。（1）原图；（2）sobel算子大小为3的sobel边缘检测；（3）canny边缘检测，参数为sobel算子大小等于3、较小阈值为50、较大阈值为220；（4）基于otsu算法的阈值分割

1. 分析与讨论
   1. **Sobel边缘检测**

如图1所示，当sobel算子的大小增大，检测出来的边缘变得更加模糊、边缘的宽度变得更粗。这是因为，sobel算子的增大，导致更多的像素点能够参数到梯度的计算中，相当于一种平滑作用，所以边缘会变得模糊，而且正是由于一个像素点的梯度的计算被更多的周围的像素点参与，导致参与梯度计算的像素点之中，有较大像素值变化的概率增大了，所以导致更多的像素点被认为是边缘，导致边缘变粗。

* 1. **Canny边缘检测**

如图2所示，当Canny边缘检测中的sobel算子增大后，一些原本不是边缘的地方却被认为是边缘，导致出现如图2中的第（8）、（9）张图中的情况。sobel算子的增大，会导致一些只有细微像素值变化的像素点被认为是边缘，就类似于微分和差分之间的关系，用于差分间隔越小，差分就越接近微分，而sobel算子的大小就相当于这个间隔。

Canny边缘检测的两个阈值对边缘检测有着重要的影响。由图2可知，在大阈值不变的时候，当小阈值较低，弱边缘点就会增多，与强边缘点连接的弱边缘点也会增多，导致出现一些噪声边缘；当小阈值增大，弱边缘点就会减少，与强边缘点连接的弱边缘点也会减少，所以有一些边缘会变短，有一些边缘会消失。在小阈值不变的时候，当大阈值增大，一些由强边缘点和弱边缘点构成的边缘可能会消失，因为这些边缘中的强边缘点的个数会随着大阈值的增大而减少，导致这条边缘完全进入弱边缘的区域，又因为没有与强边缘点相连，所以不被认为是边缘；当大阈值减小的时候，强边缘点的数量增多，一些全部位于弱边缘区域的边缘，可能变为由强边缘点和弱边缘点构成的边缘，所以边缘会增多。

* 1. **基于阈值的图像分割和基于边缘的图像分割**

观察图3，可以发现，基于阈值的图像分割方法分割出来的是一片区域，基于边缘的图像分割方法分割出来的是一个轮廓。对于基于阈值的图像分割方法，可以通过选取合理的阈值的方式，抑制分割得到的图像中的噪声，对于基于边缘的图像分割方法，需要通过调整核函数大小，或者算法过程中的某一个参数来抑制分割结果的噪声。但是基于边缘的图像分割方法得到的分割结果中的噪声一般会比基于阈值的图像分割方法要多，因为图像像素值的微小变化，可能就会导致基于边缘的图像分割方法的分割结果出现噪声。

1. 问题发现与探究

【**问题描述**】

在最初的时候，就是还在想着如何自己实现canny边缘检测的时候，在强边缘点和弱边缘点的连通性验证部分，我最初的想法是这样的：遍历所有弱边缘点，对于某一个弱边缘点，查找其邻域8个像素点中，有无强边缘点，如果有，则升级为强边缘点，如果没有，则舍弃。但是这个想法存在问题：就算弱边缘点邻域8个像素点没有强边缘点，该弱边缘点还是可以与强边缘点相连，只不过不是直接相连，中间隔了很多弱边缘点。

【**问题探究**】

验证强边缘点和弱边缘点的连通性，我最初是以弱边缘点为主体来研究的，我在意识到这一点之后，突然想到，如果以强边缘点为主体呢，也就是遍历所有强边缘点，结合之前发现的问题，所以我认为需要进行逐层的搜索。这个时候，最终的方案的雏形就产生了，此时需要确定停止搜索的条件和一个弱边缘点升级为强边缘点的条件。

这两个条件的第一个版本是：寻找有无与上一层的强边缘点（包括升级后的弱边缘点）相邻的弱边缘点，如果有，则将弱边缘点升级为强边缘点，然后扩张到下一层，如果没有，则停止。但是又想到了，如果本层也有强边缘点呢，所以有了第二个版本。

第二个版本是：如果有弱边缘点，除了与上一层强边缘点相邻可以升级为强边缘点，如果本层也有强像素点且与其相邻，也可以升级为强边缘点，如果没有弱边缘点就停止。此时又想到，没有弱边缘点，但是如果有强边缘点呢，于是有了最终版本。

最终版本是：寻找有无与上一层的强边缘点（包括升级后的弱边缘点）相邻的弱边缘点或者与本层强边缘点（如果本层有强边缘点）相邻的弱边缘点，如果有，则将弱边缘点升级为强边缘点，然后继续扩张，结束条件为：这一层没有弱边缘点和强边缘点；这一层没有强边缘点但是有弱边缘点，但是没有与上一层的强边缘点相邻。

1. 实验总结

通过本次实验，我深入理解了图像分割的基本原理，特别是基于边缘检测的图像分割方法在实际应用中的实现过程与效果表现。在实验中，我分别采用了Sobel算子与Canny算子对指定图像进行了边缘检测，从而提取出图像中的边缘。通过使用Python语言完成整个图像分割流程，不仅提升了我对图像处理技术的理解，也增强了我在图像处理应用方面的实践能力。

附录：

【仿真程序】

|  |
| --- |
| import tkinter as tk  from tkinter import ttk  from tkinter import filedialog as fd  from PIL import Image  from PIL import ImageTk  import numpy as np  from matplotlib import pyplot as plt  from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk  from matplotlib.figure import Figure  import struct  import math  from filter\_design import ImgFilter  import cv2 as cv  '''  @brief 获取图像数组  @param image: PIL的Image类对象  @return height: 图像的高（行数）  @return width: 图像的宽（列数）  @return image\_array: 图像对应的数组  '''  def get\_image\_data (image):  (width, height) = image.size # 获取图像的宽和高  image\_array = np.array(image) # 获取图像的像素数据并转化为数组  return height, width, image\_array  '''  @brief 读取raw文件  @param file\_name: raw文件路径  @return raw\_array: raw文件图像数组  '''  def read\_raw (file\_name):  # 获取raw文件表示的图片的数据  raw\_file = open(file\_name, "rb") # 打开文件，以只读、二进制的方式打开  raw\_width = struct.unpack("i", raw\_file.read(4))[0] # 获取raw文件表示的图片的宽度  raw\_height = struct.unpack("i", raw\_file.read(4))[0] # 获取raw文件表示的图片的高度  raw\_data = struct.unpack(f"{raw\_width \* raw\_height}B", raw\_file.read()) # 获取raw文件表示的图片的数组  raw\_file.close() # 关闭文件  # 将获取到的数组转换为二维数组  raw\_array = np.array(raw\_data).reshape((raw\_height, raw\_width))  return raw\_array  '''  @brief 显示分割后的图像  @param image\_array: 图像数组  '''  def show\_edge\_image (image\_array):  global edge\_image\_tk  # 转换  image = Image.fromarray(np.uint8(image\_array))  edge\_image\_tk = ImageTk.PhotoImage(image)  # 显示分割后的图片  edge\_image\_label.config(image = edge\_image\_tk)  def Sobel\_Segmentation (kernel\_size):  global edge\_method  edge\_method = "sobel"  img\_filter.set\_kernel\_size(kernel\_size)  seg\_image\_array = img\_filter.img\_Sobel\_filter()  canny\_upper\_thres\_tip.grid\_forget()  canny\_upper\_thres\_scale.grid\_forget()  canny\_lower\_thres\_tip.grid\_forget()  canny\_lower\_thres\_scale.grid\_forget()  show\_edge\_image(seg\_image\_array)  edge\_image\_tip.config(text = f"{edge\_method}边缘检测")  scale\_frame.grid(row = 3, column = 1)  def Canny\_Segmentation (lower\_thres, upper\_thres, sobel\_size):  global edge\_method  edge\_method = "canny"  canny\_image\_array = cv.Canny(image\_array, lower\_thres, upper\_thres, apertureSize = sobel\_size)  canny\_upper\_thres\_tip.grid(row = 2, column = 0)  canny\_upper\_thres\_scale.grid(row = 2, column = 1)  canny\_lower\_thres\_tip.grid(row = 1, column = 0)  canny\_lower\_thres\_scale.grid(row = 1, column = 1)  show\_edge\_image(canny\_image\_array)  edge\_image\_tip.config(text = f"{edge\_method}边缘检测")  scale\_frame.grid(row = 3, column = 1)  def update\_sobel\_kernel\_size\_and\_image (kernel\_size):  global current\_sobel\_size, current\_upper\_thres, current\_lower\_thres, edge\_method  current\_sobel\_size = int(kernel\_size)  if edge\_method == "sobel":  Sobel\_Segmentation(current\_sobel\_size)  elif edge\_method == "canny":  Canny\_Segmentation(current\_lower\_thres, current\_upper\_thres, current\_sobel\_size)  def update\_canny\_lower\_thres\_and\_image (lower\_thres):  global current\_sobel\_size, current\_upper\_thres, current\_lower\_thres  current\_lower\_thres = int(lower\_thres)  Canny\_Segmentation(current\_lower\_thres, current\_upper\_thres, current\_sobel\_size)  def update\_canny\_upper\_thres\_and\_image (upper\_thres):  global current\_sobel\_size, current\_upper\_thres, current\_lower\_thres  current\_upper\_thres = int(upper\_thres)  Canny\_Segmentation(current\_lower\_thres, current\_upper\_thres, current\_sobel\_size)  def file\_operation ():  global origin\_image\_tk, img\_filter, image\_array  # 获取文件路径  file\_path = fd.askopenfilename()  # 获取文件格式  file\_format = []  for i in reversed(range(len(file\_path))):  if file\_path[i] == '.': # 由于只需要文件格式，所以遇到'.'就退出  break  file\_format.append(file\_path[i])  file\_format.reverse() # 列表反转  file\_format\_str = "".join(file\_format) # 转换为字符串  # 判断文件格式并得到Image对象和图像数组  if file\_format\_str == "raw":  image\_array = read\_raw(file\_path)  image = Image.fromarray(image\_array)  else:  image = Image.open(file\_path)  \_, \_, image\_array = get\_image\_data(image)  # 创建图像滤波器类对象  img\_filter = ImgFilter(kernel\_size = init\_kernel\_size, image\_array = image\_array)  # 转换为tkinter能解析的PhotoImage对象  origin\_image\_tk = ImageTk.PhotoImage(image)  # 显示图像  origin\_image\_label.config(image = origin\_image\_tk)  origin\_image\_tip.config(text = "原图像")  edge\_image\_label.config(image = "")  edge\_image\_tip.config(text = "")  scale\_frame.grid\_forget()  if \_\_name\_\_ == '\_\_main\_\_':  # 初始化参数  init\_kernel\_size = 3  init\_lower\_thres = 50  init\_upper\_thres = 150  edge\_method = None  current\_lower\_thres = init\_lower\_thres  current\_upper\_thres = init\_upper\_thres  current\_sobel\_size = init\_kernel\_size  # 创建基本界面  root = tk.Tk()  root.title("数字图像处理实验八：基于边缘的图像分割") # 设置界面标题  root.grid()  root.grid\_columnconfigure(0, weight = 1) # root的第一列会适应界面大小的改变  root.grid\_rowconfigure(0, weight = 1) # root的第一行会适应界面大小的改变  # 创建一个画布，画布可以被滚动条控制  canvas = tk.Canvas(root)  canvas.grid(row = 0, column = 0, sticky = "nsew") # canvas会填充整个界面  # 创建滚动条  scrollbar\_1 = ttk.Scrollbar(root, orient = "vertical", command = canvas.yview)  scrollbar\_1.grid(row = 0, column = 1, sticky = "ns") # 纵轴方向填充  scrollbar\_2 = ttk.Scrollbar(root, orient = "horizontal", command = canvas.xview)  scrollbar\_2.grid(row = 1, column = 0, sticky = "ew") # 横轴方向填充  # canvas与滚动条关联  canvas.config(yscrollcommand = scrollbar\_1.set)  canvas.config(xscrollcommand = scrollbar\_2.set)  # 创建Frame容器，用于存放各种子容器  frame = ttk.Frame(canvas, padding = 10)  frame.grid(row = 0, column = 0)  canvas.create\_window((0, 0), window = frame, anchor = "nw") # 将frame嵌入canvas  frame.bind("<Configure>", lambda event: canvas.configure(scrollregion = canvas.bbox("all"))) # 使滚动条适应frame  # 创建存放按键的Frame容器  button\_frame = ttk.Frame(frame, padding = 10)  button\_frame.grid(row = 0, column = 0)  # 创建“打开文件”按键  open\_file\_button = ttk.Button(button\_frame, text = "打开文件", command = file\_operation)  open\_file\_button.grid(row = 0, column = 0, columnspan = 2)  # 创建“退出”按键  quit\_button = ttk.Button(button\_frame, text = "退出", command = root.destroy)  quit\_button.grid(row = 2, column = 0, columnspan = 2)  # 创建功能按键  sobel\_button = ttk.Button(button\_frame, text = "Sobel边缘检测", command = lambda: Sobel\_Segmentation(current\_sobel\_size))  sobel\_button.grid(row = 1, column = 0)  canny\_button = ttk.Button(button\_frame, text = "Canny边缘检测", command = lambda: Canny\_Segmentation(current\_lower\_thres, current\_upper\_thres, current\_sobel\_size))  canny\_button.grid(row = 1, column = 1)  # 创建存放滑动条的Frame容器  scale\_frame = ttk.Frame(frame, padding = 10)  # scale\_frame.grid(row = 3, column = 1)  # 创建滑动条  sobel\_kernel\_size\_tip = ttk.Label(scale\_frame, text = "sobel算子大小")  sobel\_kernel\_size\_tip.grid(row = 0, column = 0)  sobel\_kernel\_size\_scale = tk.Scale(scale\_frame, from\_ = 3, to = 7, orient = "horizontal", length = 150, command = update\_sobel\_kernel\_size\_and\_image, resolution = 2)  sobel\_kernel\_size\_scale.set(init\_kernel\_size)  sobel\_kernel\_size\_scale.grid(row = 0, column = 1)  canny\_lower\_thres\_tip = ttk.Label(scale\_frame, text = "canny小阈值")  canny\_lower\_thres\_tip.grid(row = 1, column = 0)  canny\_lower\_thres\_scale = tk.Scale(scale\_frame, from\_ = 0, to = 255, orient = "horizontal", length = 150, command = update\_canny\_lower\_thres\_and\_image)  canny\_lower\_thres\_scale.set(init\_lower\_thres)  canny\_lower\_thres\_scale.grid(row = 1, column = 1)  canny\_upper\_thres\_tip = ttk.Label(scale\_frame, text = "canny大阈值")  canny\_upper\_thres\_tip.grid(row = 2, column = 0)  canny\_upper\_thres\_scale = tk.Scale(scale\_frame, from\_ = 0, to = 255, orient = "horizontal", length = 150, command = update\_canny\_upper\_thres\_and\_image)  canny\_upper\_thres\_scale.set(init\_upper\_thres)  canny\_upper\_thres\_scale.grid(row = 2, column = 1)  # 创建图像标签  origin\_image\_label = ttk.Label(frame)  origin\_image\_label.grid(row = 1, column = 0)  edge\_image\_label = ttk.Label(frame)  edge\_image\_label.grid(row = 1, column = 1)  # 创建图像标注  origin\_image\_tip = ttk.Label(frame)  origin\_image\_tip.grid(row = 2, column = 0)  edge\_image\_tip = ttk.Label(frame)  edge\_image\_tip.grid(row = 2, column = 1)  # 创建一个布尔变量，判断“确定”按键是否按下  bool\_var = tk.BooleanVar()  root.mainloop() |

指导老师评价

|  |  |  |
| --- | --- | --- |
|  | | |
| 仿真实验报告成绩 | 交流展示成绩 | 总评成绩 |
|  |  |  |

指导教师：

年 月 日

**实验报告提交说明:**

文件名 **姓名\_学号\_实验x.docx**

文件格式 Microsoft Word