

**“数字图像处理基础”实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 题目 | 实验六：基于阈值的图像分割 | | |
| 姓名 | 钟军凯 | 学号 | 22211374 |
| 指导教师 | 彭亚辉 | | |
| 日期 | 2025年3月27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 9](#_Toc191473899)

[5. 问题发现与探究 9](#_Toc191473900)

[6. 实验总结 1](#_Toc191473901)2

[指导老师评价 2](#_Toc191473902)1

1. 实验目的分析

本实验旨在掌握基于阈值的图像分割方法，包括对灰度图像中特定结构的分割（如骨骼等），通过合适的阈值（如大津法）进行处理并分析效果。此外，还将学习基于颜色阈值（如HSV空间）的分割方法，并使用Python实现这些分割操作。

1. 实验方案设计
   1. **CT图像骨骼分割**

考虑使用ostu算法进行自动阈值选取。ostu算法的流程如下：

1. 计算图像的归一化直方图，得到每一个灰度级的频率。



其中，是灰度级i的像素点个数，N为总的像素点个数。

1. 遍历所有灰度级，假设当前灰度级为T，背景的灰度级为0~T，前景的灰度级为(T + 1) ~( L - 1)。
2. 计算背景和前景的概率。



1. 计算背景和前景的均值。



1. 计算类间方差。



1. 选取最大类间方差对应的灰度级作为阈值，像素值小于阈值的像素点置为0，像素值大于或等于阈值的像素点置为255。

此处为了防止多次进行求和计算，所以考虑采用递推的方式。



* 1. **提取彩色图像特定颜色**

相对于RGB空间来说，HSV空间更加适合用于提取颜色。HSV三个字母的含义分别是：色相（范围0~360）、饱和度（0~1）、亮度（0~1）。每一种颜色对会有相应的色相范围。饱和度控制颜色的纯度，饱和度越高，颜色越鲜艳；饱和度越低，颜色越暗淡。亮度控制颜色的明暗。可以通过色相选取颜色，通过控制饱和度和亮度来实现更精准的颜色选取。使用Image.convert()将图像从RGB空间转换到HSV空间。

点击“彩色图像颜色提取”按键后，出现弹窗提示用户选择需要提取的颜色，此处提供了四种颜色：红色、蓝色、黄色和绿色。点击相应的按键，即可实现颜色的提取。通过上网搜索不同颜色在HSV空间中的取值，将对应取值之外的像素点的亮度设置为0，即设置为黑色。在弹窗中的按键被按下之前，程序处在阻塞状态，直到有按键被按下，使用root.wait\_variable()实现。

3. 实验结果

初始界面如图1所示。

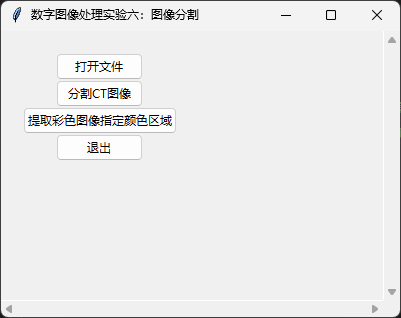


图 1

CT图像分割效果如图2、图3、图4和图5所示。

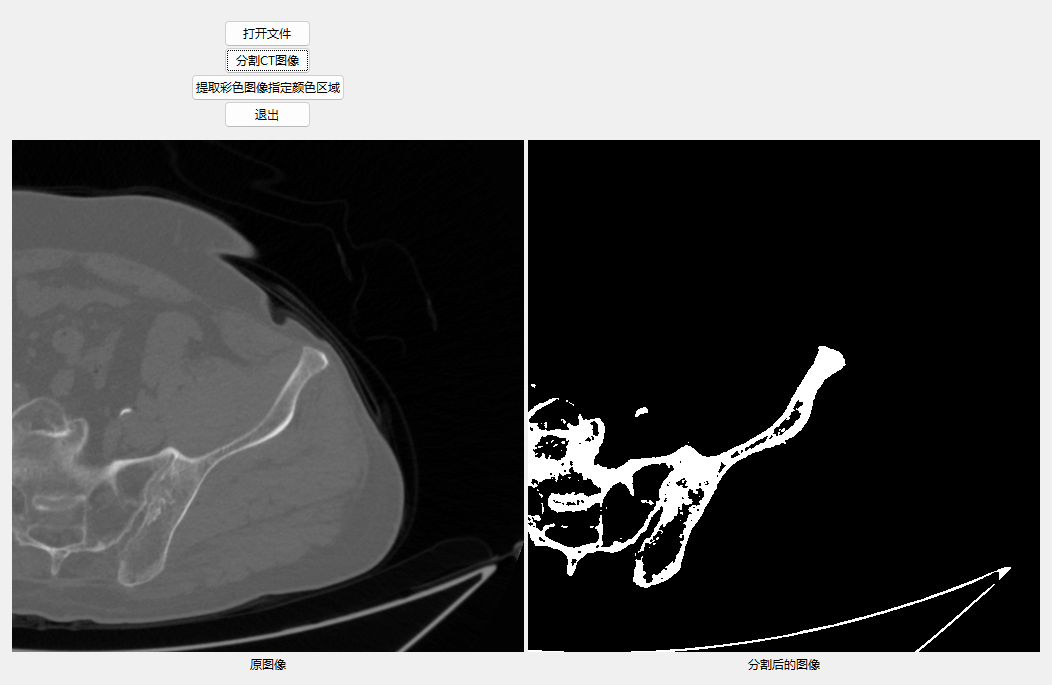


图 2

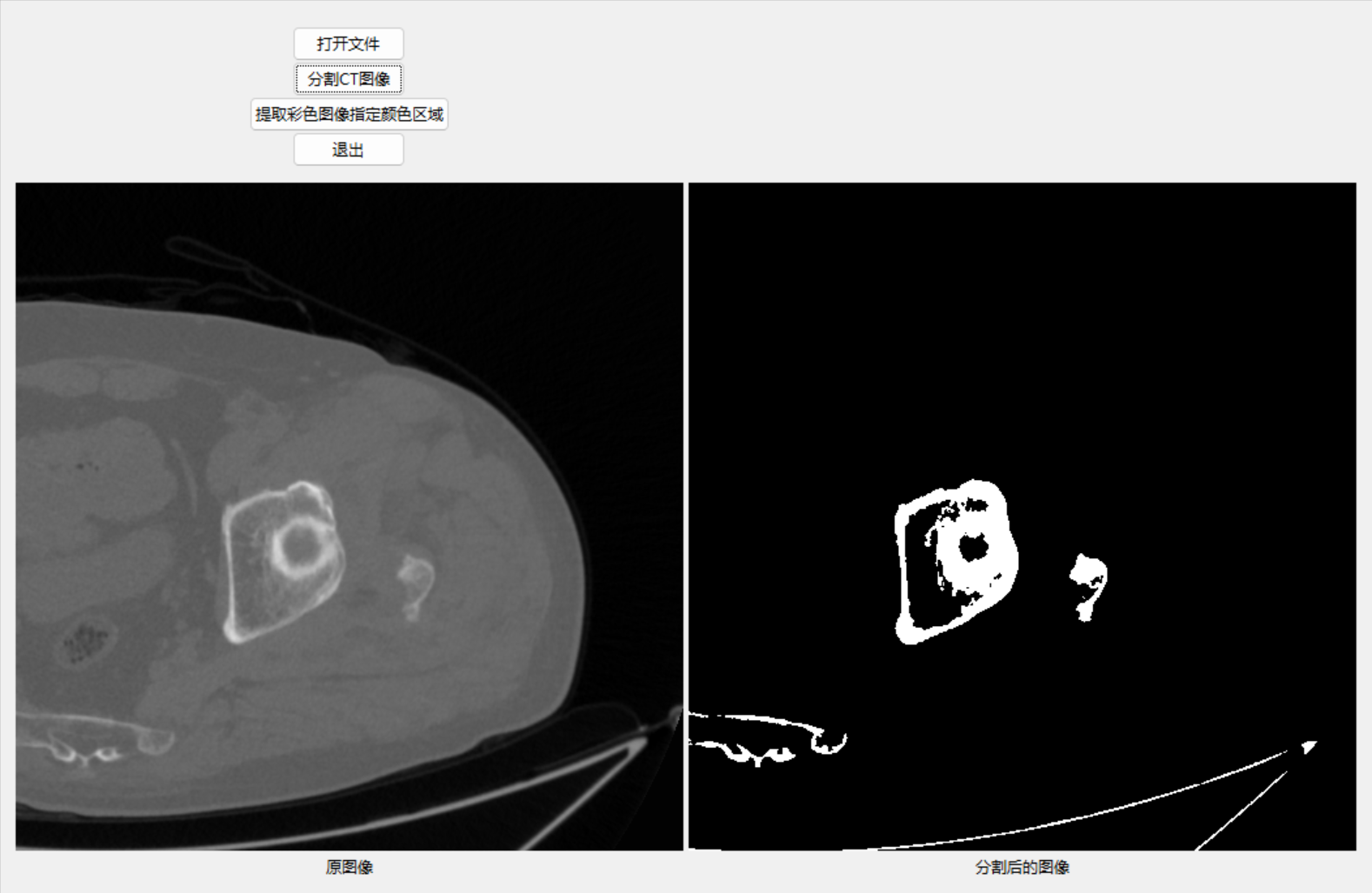


图 3

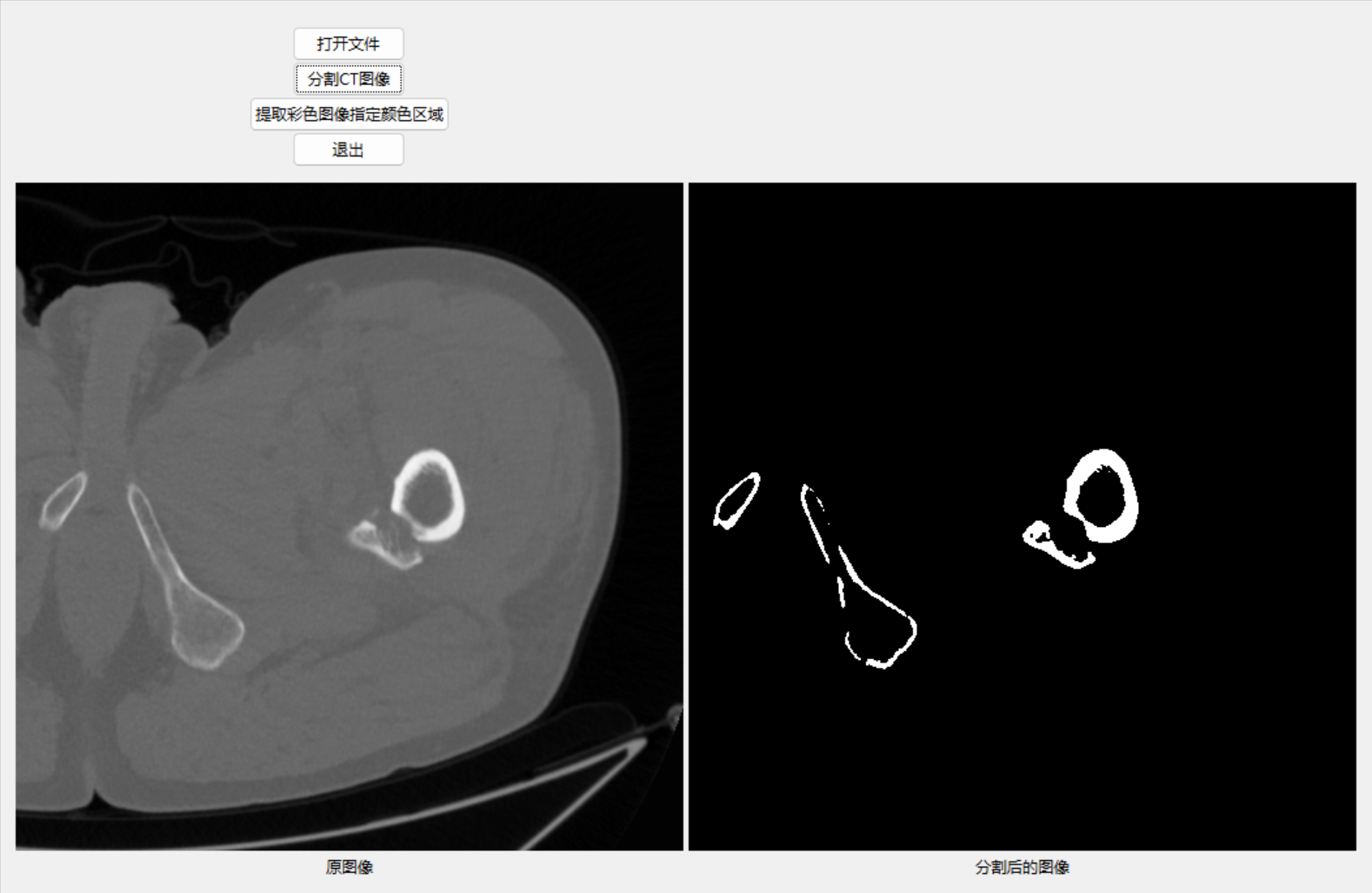


图 4

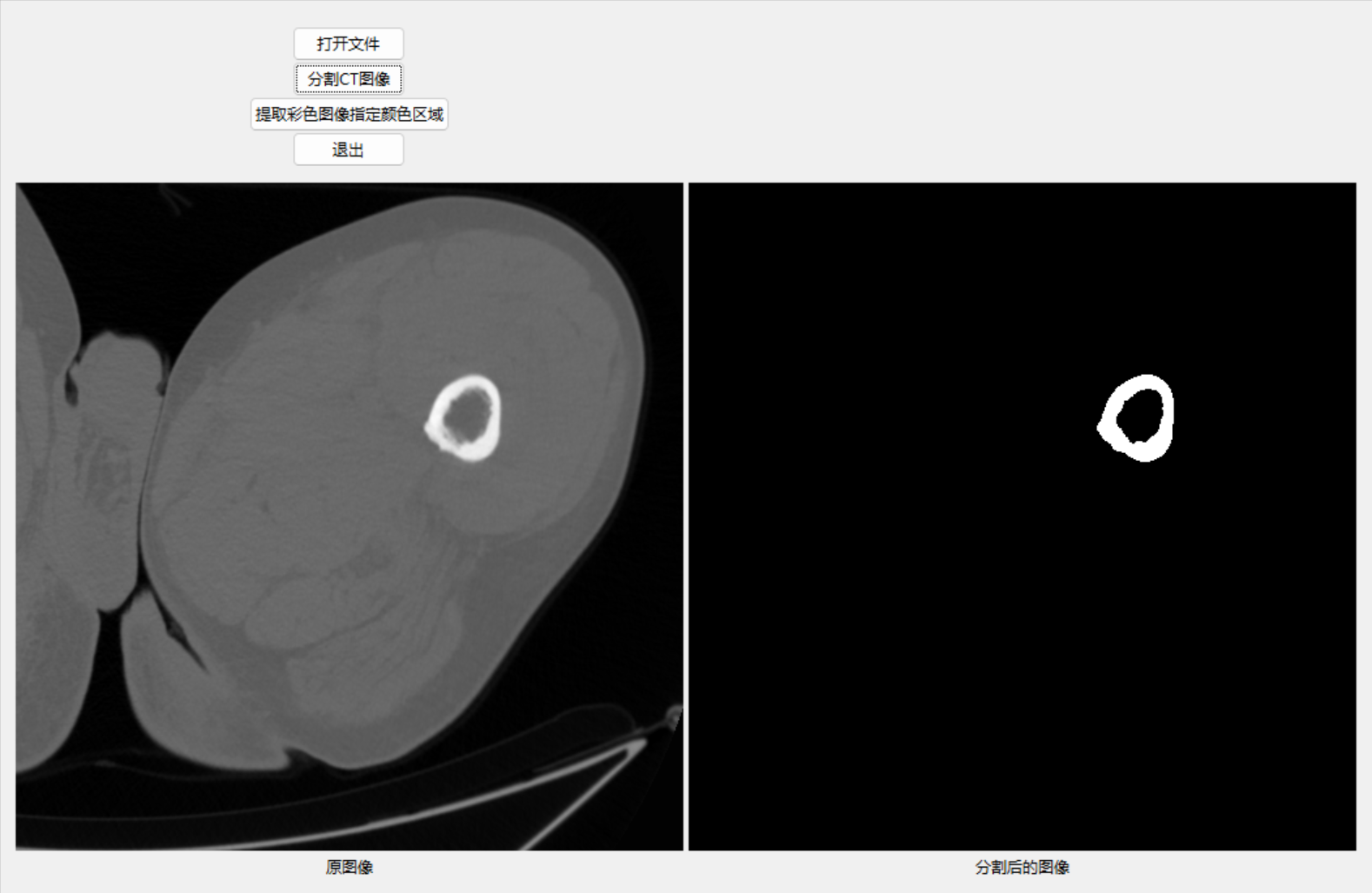


图 5

彩色图像颜色提取效果如图6、图7、图8、图9和图10所示。本实验实现了对红色、蓝色、黄色和绿色的提取。



图 6：选择提取颜色的弹窗

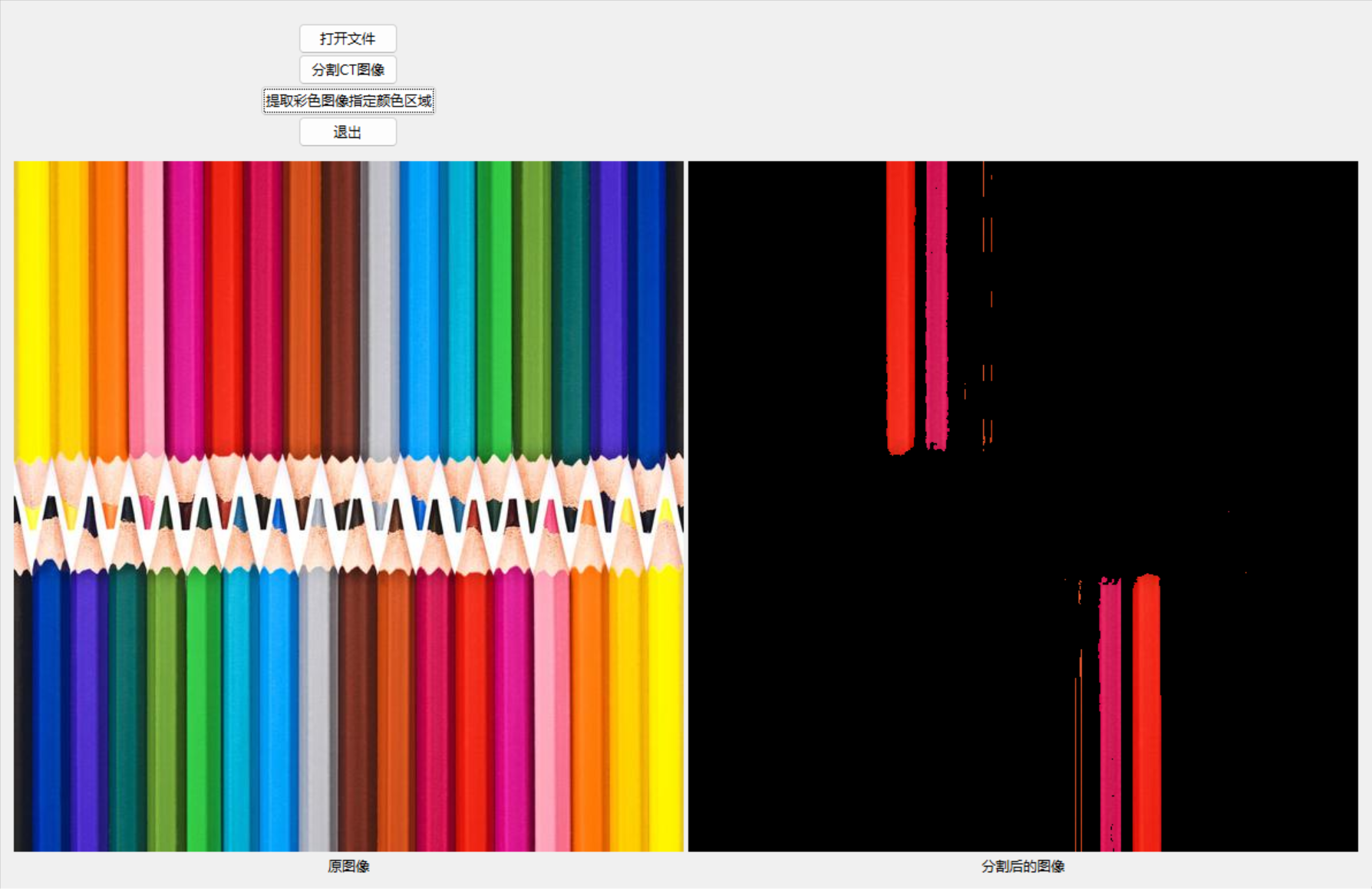


图 7：对红色的提取

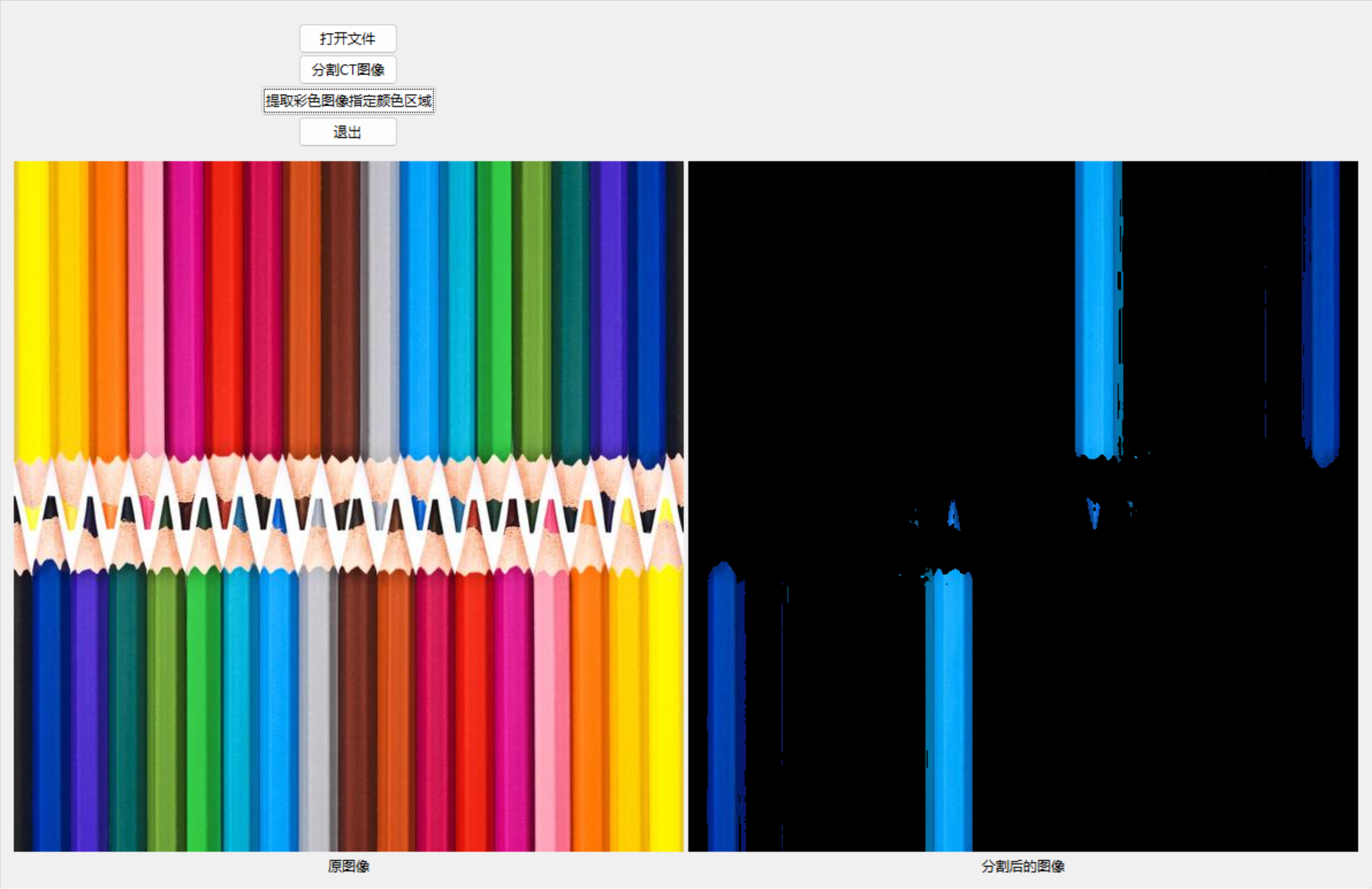


图 8：对蓝色的提取

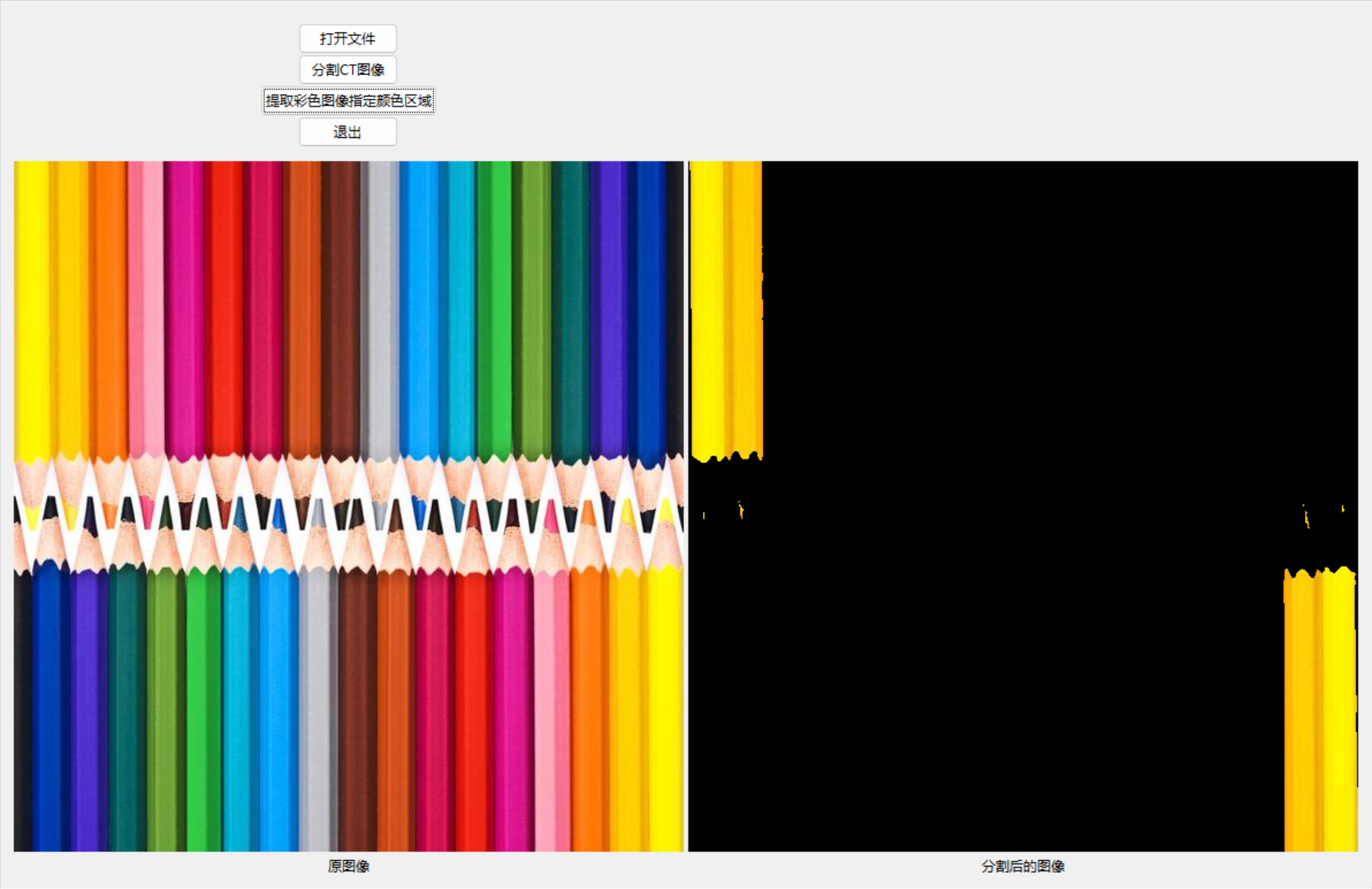


图 9：对黄色的提取

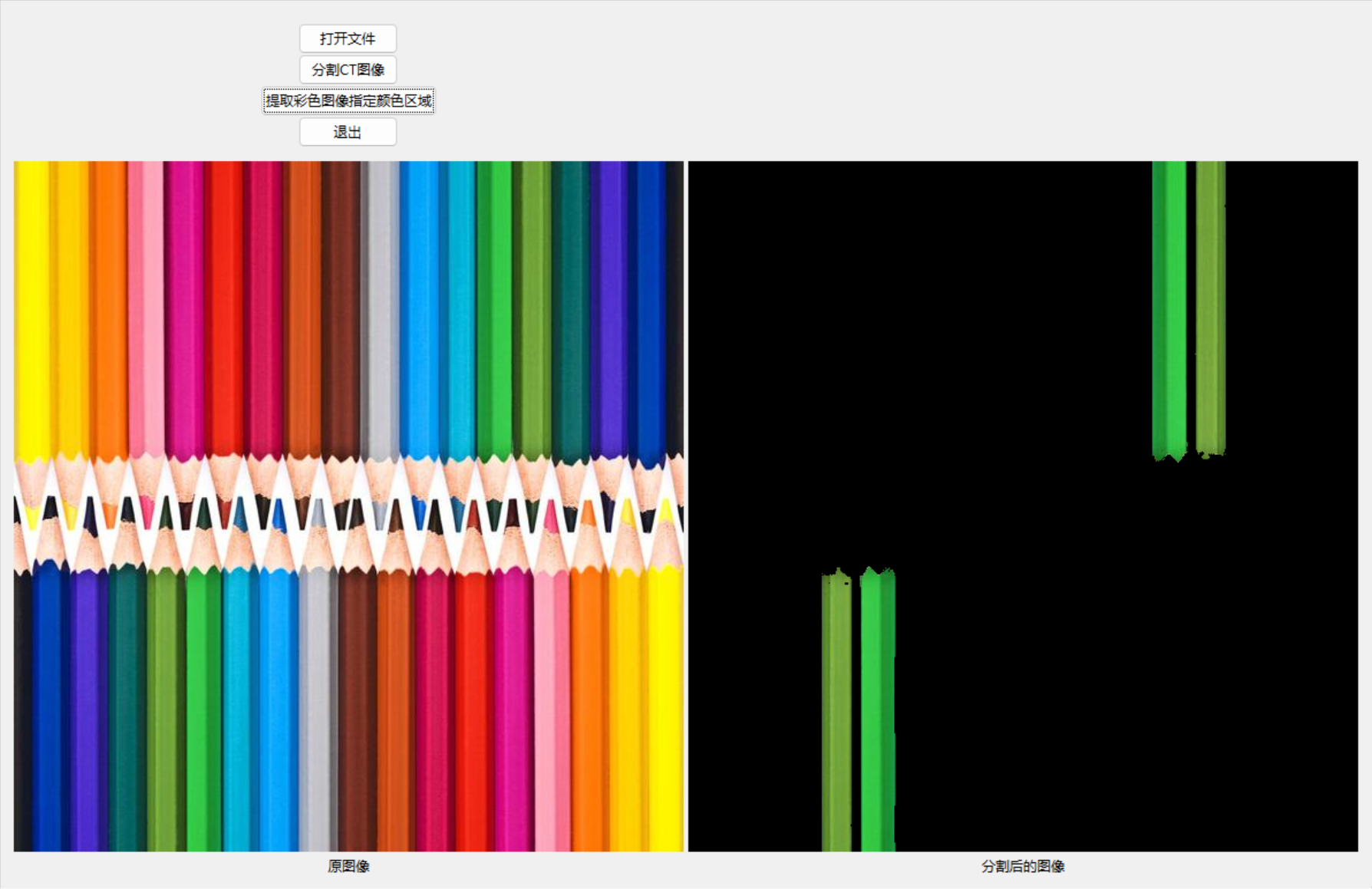


图 10：对绿色的提取

1. 分析与讨论

本次实验实现了对CT图像中的骨骼区域进行分割，同时也完成了对彩色图像的特定颜色的提取。

在提取彩色图像的特定颜色的过程中，发现PIL这个库的HSV三个值的数据范围都是[0，255]，而不是像原始定义那样，H的范围是[0，360]，S和V的范围都是[0，1]。所以需要进行不同范围的映射。但是，搜索到的不同颜色对应的HSV的数值其实都不是非常的准确，搜索到的一个颜色对应的HSV对应的数值的范围还是偏大的，所以最终需要对数据进行微调，从而达到对单一颜色的提取。

1. 问题发现与探究
   1. **问题一**
      1. **问题描述**

如果只是简单地将otsu算法应用于对CT图像中骨骼的分割，则会出现如图11、图12等类似的情况。此时，otsu算法只能将主体的整体分割出来，但是骨骼是存在于主体之中的。

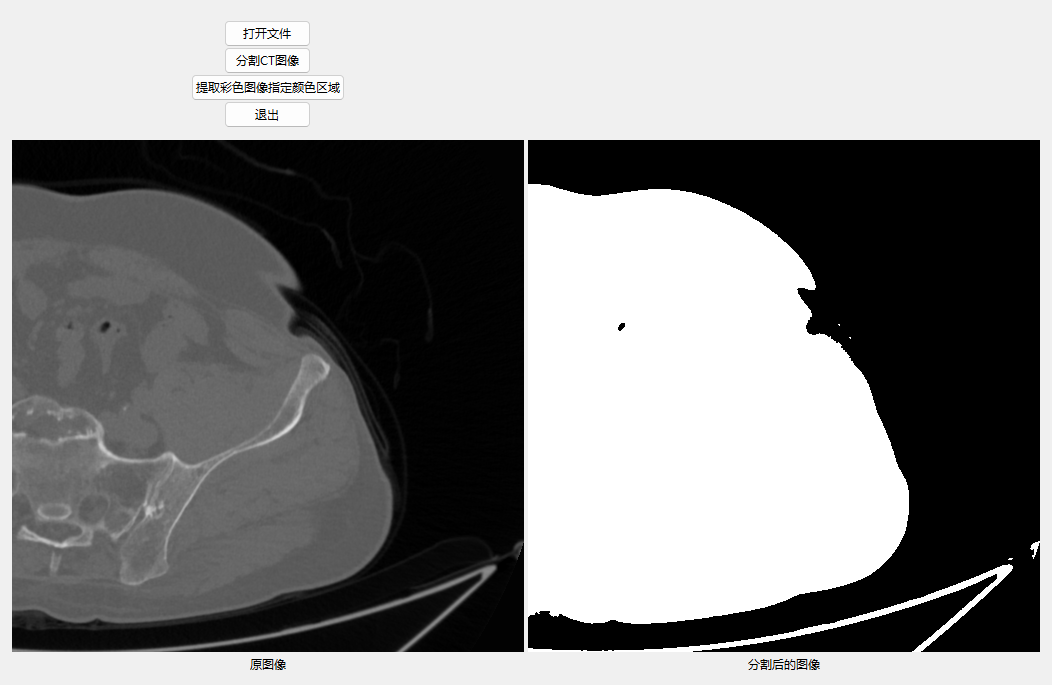


图 11

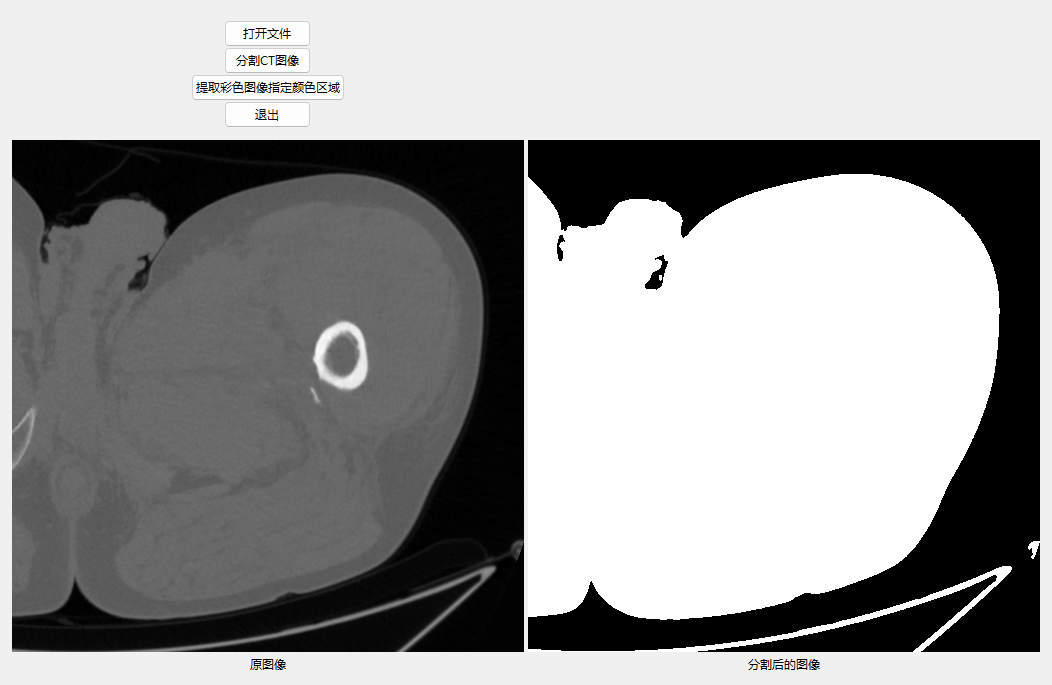


图 12

* + 1. **问题探究**

因为ostu算法比较适合用于双峰值图像的阈值分割，又因为CT图像中，背景的黑色和主体的灰色占比比较多，所以表示骨骼的偏白的部分对ostu算法的阈值选择的影响是非常小的，所以导致整个主体倍分割出来，而不是只有骨骼被分割出来。

此时观察图1、图2和其他类似的CT图像被分割后的图像，会发现，此时的ostu算法将CT图像中的主体部分和背景部分分离开了，而骨骼部分存在于主体部分中，所以此时可以考虑对主体部分再次进行一次图像分割，也是使用ostu算法，这样就可以将骨骼部分和主体部分分离开了，此时骨骼部分相当于主体部分，主体部分相当于背景部分。

因为对ostu算法的第一次使用已经得到分割主体部分和背景部分所需要的阈值，所以此时可以将第一次使用ostu算法得到的阈值设置为灰度范围的下界，将255设置为上界，然后对这个灰度范围再次使用ostu算法，得到分割阈值，此时即可将骨骼部分和CT图像的主体部分分割开来。

此时需要注意的是，在新的灰度范围内使用ostu算法的时候，相应的概率需要进行对应的归一化操作，也就是使用对于整幅图像而言的概率除以新的灰度范围的总概率，从而得到对于新的灰度范围而言的概率。

个人认为这种方式会比选取经验阈值的方式要好。

* 1. **问题二**
     1. **问题描述**

在提取彩色图像的颜色的时候，出现了如图13所示的报错。

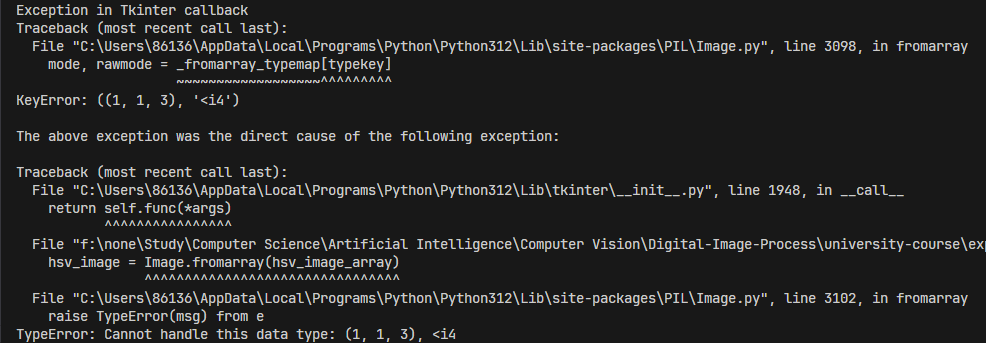


图 13

* + 1. **问题探究**

根据报错提示，是Image.fromarray()的使用出了问题。其中出现“<i4”，这个符号的意思是小端序的4字节整数，也就是int32类型。此时我使用.dtype查看数组数据类型，发现果然是int32类型的。此时，我回想起在操作灰度图像的时候是没有报错的，此时对于灰度图像，我也使用.dtype查看数据类型，发现也是int32。

通过查阅资料，发现Image.fromarray()主要支持uint8类型的数据，当处理的对象是二维数组的时候，PIL会将数据映射为模式“I”（32 位整型图像），这种情况通常不会报错，所以在对灰度图像操作的时候没有报错。但是对于三维数组（比如说RGB图像、HSV图像）来说，PIL没有直接对应的32位彩色图像模式，因此直接使用int32数组可会报错。

此时我又想，数据类型是什么时候开始变为int32的，是否在得到该数组的时候就已经是int32类型的了。我是通过image\_array = numpy.array(Image.getdata())和image\_array = image\_array.reshape((Image.size[0], Image.size[1], 3))这两句代码得到RGB图像的数组的。此时我使用.dtype来查看数据类型，发现这时就已经是int32类型了。

通过查阅资料发现，最合理的获取数组的方式是这样：image\_array = numpy.array(Image)。numpy.array(Image.getdata())会导致图像数组的第一维和第二维合成为一个维度，从而导致维度消失，导致需要重塑数组。而image\_array = numpy.array(Image)不会导致维度消失，而且得到的数组的数据类型是uint8，所以就不会出现如图13所示的报错了。

6. 实验总结

在本次实验中，在CT图像分割骨骼的实现过程中，在刚开始的时候，看到使用ostu算法得到的结果，个人感觉分割效果是不好的，完全没有把骨骼分割出来。但是，后来经过思考，想到了一种在不同灰度范围中使用ostu算法的方法，从而实现了对骨骼的分割，这应该算是一次灵光一现吧。与此同时，在提取彩色图像颜色的实验中，对HSV空间有了一定程度的了解，也对PIL库中相应的函数对数据类型的限制有了印象。

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  '''  @brief RGB空间映射到HSV空间  @param image: PIL的Image类  @return image: HSV空间的图像的Image类  '''  def RGB\_to\_HSV (image):  image = image.convert("HSV")  return image  '''  @brief HSV空间映射到RGB空间  @param image: PIL的Image类  @return image: RGB空间的图像的Image类  '''  def HSV\_to\_RGB (image):  image = image.convert("RGB")  return image  '''  @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 显示分割后的图像  @param image\_array: 图像数组  '''  def show\_seg\_image (image\_array):  global seg\_image\_tk  # 转换  image = Image.fromarray(np.uint8(image\_array))  seg\_image\_tk = ImageTk.PhotoImage(image)  # 显示分割后的图片  seg\_image\_label.config(image = seg\_image\_tk)  '''  @brief 计算图像归一化直方图  @param image\_array: 图像数组  @return histogram: 图像归一化直方图  '''  def get\_histogram (image\_array):  # 获取图像行数、列数  rows = image\_array.shape[0]  columns = image\_array.shape[1]  # 计算归一化直方图  if len(image\_array.shape) == 2:  histogram = np.zeros(256)  for i in range(256):  histogram[i] = np.count\_nonzero(image\_array == i) / (rows \* columns)  elif len(image\_array.shape) == 3:  histogram = np.zeros((3, 256))  for i in range(3):  for j in range(256):  histogram[i, j] = np.count\_nonzero(image\_array[:, i] == j) / (rows \* columns)    return histogram  '''  @brief 获取由otsu算法得到的图像分割阈值  @param image\_array: 图像数组  @param min\_grayscale: 算法作用范围的下界  @param max\_grayscale: 算法作用范围的上界  @return otsu\_threshold: 通过otsu算法得到的最佳阈值点  '''  def get\_otsu\_threshold (image\_array, min\_grayscale, max\_grayscale):  # 初始化  prob1 = 0 # 前景概率  prob2 = 0 # 背景概率  mean1 = 0 # 前景均值  mean2 = 0 # 背景均值  var = 0 # 类间方差  otsu\_threshold = 0 # ostu算法阈值  # 获取归一化直方图  histogram = get\_histogram(image\_array)  # 计算算法作用范围的总概率  prob = np.sum(histogram[min\_grayscale : max\_grayscale + 1])  # 遍历所有灰度值  for i in range(min\_grayscale, max\_grayscale + 1):  # 计算前景概率、背景概率、前景均值和背景均值  if i == min\_grayscale:  prob1 = histogram[min\_grayscale] / prob  prob2 = np.sum(histogram[min\_grayscale : max\_grayscale + 1]) / prob  mean1 = 0  mean2 = np.average(np.arange(min\_grayscale, max\_grayscale + 1), weights = histogram[min\_grayscale : max\_grayscale + 1] / prob)  else:  mean1 = (prob1 \* mean1 + i \* histogram[i] / prob) / (prob1 + histogram[i] / prob)  if prob2 - histogram[i] / prob == 0:  mean2 = 0  else:  mean2 = (prob2 \* mean2 - i \* histogram[i] / prob) / (prob2 - histogram[i] / prob)  prob1 = prob1 + histogram[i] / prob  prob2 = prob2 - histogram[i] / prob  # 计算类间方差  if prob1 \* prob2 \* ((mean1 - mean2) \*\* 2) > var:  var = prob1 \* prob2 \* ((mean1 - mean2) \*\* 2)  otsu\_threshold = i  return otsu\_threshold  '''  @brief 分割CT图像  @param none  '''  def threshold\_segmentation ():  global origin\_image  \_, \_, image\_array = get\_image\_data(origin\_image) # 获取图像数组  threshold = get\_otsu\_threshold(image\_array, 0, 255) # 第一次获取ostu算法的阈值  threshold = get\_otsu\_threshold(image\_array, threshold, 255) # 第二次获取ostu算法的阈值  image\_array[image\_array < threshold] = 0 # 小于该阈值的像素值置为0  image\_array[image\_array >= threshold] = 255 # 大于或等于该阈值的像素值置为255  show\_seg\_image(image\_array) # 显示分割后的图像  seg\_image\_tip.config(text = "分割后的图像")  '''  @brief 提取彩色图像指定颜色区域  @param nnoe  '''  def threshold\_color\_segmentation ():  global origin\_image  hsv\_image = RGB\_to\_HSV(origin\_image)  \_, \_, hsv\_image\_array = get\_image\_data(hsv\_image)  seg\_color = None  popup = tk.Toplevel(root)  popup.title("选择颜色")  popup.grid()  red\_button = ttk.Button(popup, text = "红色")  red\_button.grid(row = 0, column = 0)  blue\_button = ttk.Button(popup, text = "蓝色")  blue\_button.grid(row = 0, column = 1)  yellow\_button = ttk.Button(popup, text = "黄色")  yellow\_button.grid(row = 0, column = 2)  green\_button = ttk.Button(popup, text = "绿色")  green\_button.grid(row = 0, column = 3)  '''  @brief 点击了“红色”按键  @param none  '''  def click\_red\_button ():  nonlocal seg\_color  seg\_color = "red"  bool\_var.set(True)  popup.destroy()  '''  @brief 点击了“蓝色”按键  @param none  '''  def click\_blue\_button ():  nonlocal seg\_color  seg\_color = "blue"  bool\_var.set(True)  popup.destroy()  '''  @brief 点击了“黄色”按键  @param none  '''  def click\_yellow\_button ():  nonlocal seg\_color  seg\_color = "yellow"  bool\_var.set(True)  popup.destroy()  '''  @brief 点击了“绿色”按键  @param none  '''  def click\_green\_button ():  nonlocal seg\_color  seg\_color = "green"  bool\_var.set(True)  popup.destroy()  red\_button.config(command = click\_red\_button)  blue\_button.config(command = click\_blue\_button)  yellow\_button.config(command = click\_yellow\_button)  green\_button.config(command = click\_green\_button)  # 等待平移参数输入完毕  root.wait\_variable(bool\_var)  bool\_var.set(False) # 复位bool\_var  bool\_index = None  if seg\_color == "red":  bool\_index = (((hsv\_image\_array[:, :, 0] >= 0) & (hsv\_image\_array[:, :, 0] <= 10)) | \  ((hsv\_image\_array[:, :, 0] >= 240) & (hsv\_image\_array[:, :, 0] <= 255))) & \  ((hsv\_image\_array[:, :, 1] >= 200) & (hsv\_image\_array[:, :, 1] <= 255)) & \  ((hsv\_image\_array[:, :, 2] >= 200) & (hsv\_image\_array[:, :, 2] <= 255))  elif seg\_color == "blue":  bool\_index = ((hsv\_image\_array[:, :, 0] >= 140) & (hsv\_image\_array[:, :, 0] <= 170)) & \  ((hsv\_image\_array[:, :, 1] >= 200) & (hsv\_image\_array[:, :, 1] <= 255)) & \  ((hsv\_image\_array[:, :, 2] >= 100) & (hsv\_image\_array[:, :, 2] <= 255))  elif seg\_color == "yellow":  bool\_index = ((hsv\_image\_array[:, :, 0] >= 25) & (hsv\_image\_array[:, :, 0] <= 45)) & \  ((hsv\_image\_array[:, :, 1] >= 200) & (hsv\_image\_array[:, :, 1] <= 255)) & \  ((hsv\_image\_array[:, :, 2] >= 200) & (hsv\_image\_array[:, :, 2] <= 255))  elif seg\_color == "green":  bool\_index = ((hsv\_image\_array[:, :, 0] >= 60) & (hsv\_image\_array[:, :, 0] <= 105)) & \  ((hsv\_image\_array[:, :, 1] >= 100) & (hsv\_image\_array[:, :, 1] <= 255)) & \  ((hsv\_image\_array[:, :, 2] >= 100) & (hsv\_image\_array[:, :, 2] <= 255))    hsv\_image\_array[:, :, 2][~bool\_index] = 0  hsv\_image = Image.fromarray(hsv\_image\_array, mode = "HSV")  rgb\_image = HSV\_to\_RGB(hsv\_image)  \_, \_, rgb\_image\_array = get\_image\_data(rgb\_image)  show\_seg\_image(rgb\_image\_array)  seg\_image\_tip.config(text = "分割后的图像")  '''  @brief “打开文件”按键的回调函数，打开文件选择界面，并对文件进行相应操作  @param none  '''  def file\_operation ():  global origin\_image, origin\_image\_tk  file\_path = fd.askopenfilename() # 获取文件路径    origin\_image = Image.open(file\_path) # 打开图像得到Image类对象  origin\_image\_tk = ImageTk.PhotoImage(origin\_image) # 转化为PhotoImage类对象  origin\_image\_label.config(image = origin\_image\_tk) # 显示原始图像  origin\_image\_tip.config(text = "原图像") # 显示原始图像标注  seg\_image\_label.config(image = "") # 清空分隔后的图像  seg\_image\_tip.config(text = "") # 清空分割后的图像的标注    if \_\_name\_\_ == '\_\_main\_\_':  # 创建基本界面  root = tk.Tk()  root.title("数字图像处理实验六：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 = 5)  # 创建“退出”按键  quit\_button = ttk.Button(button\_frame, text = "退出", command = root.destroy)  quit\_button.grid(row = 3, column = 0)  # 创建功能按键  seg\_ct\_image\_button = ttk.Button(button\_frame, text = "分割CT图像", command = threshold\_segmentation)  seg\_ct\_image\_button.grid(row = 1, column = 0)  seg\_colorful\_image\_button = ttk.Button(button\_frame, text = "提取彩色图像指定颜色区域", command = threshold\_color\_segmentation)  seg\_colorful\_image\_button.grid(row = 2, column = 0)  # 创建图像标签  origin\_image\_label = ttk.Label(frame)  origin\_image\_label.grid(row = 1, column = 0)  seg\_image\_label = ttk.Label(frame)  seg\_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)  seg\_image\_tip = ttk.Label(frame)  seg\_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