广东工业大学

计算机学院

《计算机视觉》

实验报告



|  |  |
| --- | --- |
| 课程名称： | 计算机视觉 |
| 实验名称： | SIFT特征点与图像拼接 |
| 指导教师： | 赵靖亮 |
| 学生姓名： | 欧炜标 |
| 班级学号： | 3121005358 |
| 实验日期： | 2024/5/28 |
| 实验成绩： |  |

实验报告撰写要求

实验操作是教学过程中理论联系实际的重要环节，而实验报告的撰写又是知识系统化的吸收和升华过程，因此，实验报告应该体现完整性、规范性、正确性、有效性。现将实验报告撰写的有关内容说明如下：

1、 实验报告模板为电子版。

2、 下载统一的实验报告模板，学生自行完成撰写和打印。报告的首页包含本次实验的一般信息：

*  组 号：例如：2-5 表示第二班第5组。
*  实验日期：例如：05-10-06 表示本次实验日期。(年-月-日)……
*  实验编号：例如：No.1 表示第一个实验。
*  实验时间：例如：2学时 表示本次实验所用的时间。

实验报告正文部分，从五个方面（目的、内容、步骤等）反映本次实验的要点、要求以及完成过程等情况。模板已为实验报告正文设定统一格式，学生只需在相应项内填充即可。（第一部分已经填好）。

3、 实验报告正文部分具体要求如下：

一、实验目的

本次实验所涉及并要求掌握的知识点。

二、实验环境

实验所使用的设备名称及规格，网络管理工具简介、版本等。

三、实验内容与实验要求

实验内容、原理分析及具体实验要求。

四、实验过程与分析

根据具体实验，记录、整理相应命令、运行结果等，包括截图和文字说明。

详细记录在实验过程中发生的故障和问题，并进行故障分析，说明故障排除的过程及方法。

五、实验结果总结

对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的改进意见。

注：本部分仅为展示报告内容，具体报告内容见下一页。

一、实验目的

*本次实验所涉及并要求掌握的知识点。（可多页）*

 了解图像拼接的一般过程。掌握基于SIFT特征点的图像拼接算法流程和代码实现。

二、实验环境

**硬件环境**

设备名称 LAPTOP-BHF8HOH1

处理器 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz

机带 RAM 16.0 GB (15.8 GB 可用)

设备 ID 8A9CD625-D5C1-40B0-AFDA-7C0219502457

产品 ID 00342-36199-79021-AAOEM

系统类型 64 位操作系统, 基于 x64 的处理器

笔和触控 没有可用于此显示器的笔或触控输入

**软件环境**

python3.12

opencv

numpy

**IDE**

pycharm

三、实验内容与实验要求

*实验内容、原理分析及具体实验要求。（包括SIFT特征点、图像拼接。）（可多页）*

1. 实验内容

本实验主要分为以下四个部分：

* 两幅图像拼接：使用OpenCV的Stitcher类将两幅图像拼接成一幅图像。
* 单独计算图像的高斯差分图像：对图像进行高斯模糊和高斯差分处理，并将结果保存。
* 提取SIFT特征点和特征向量：使用OpenCV的SIFT\_create()函数提取图像的特征点及其特征向量，并将结果绘制在原图上。
* 特征点匹配：使用BFMatcher类对两幅图像的特征点进行匹配，并将匹配结果保存。

1. 原理分析

* 图像拼接：图像拼接的基本流程包括特征点提取、特征点匹配、图像扭曲和融合。我们使用SIFT（Scale Invariant Feature Transform）算法来提取特征点，并利用特征点匹配来确定图像间的几何变换，从而实现图像的拼接。
* 高斯差分：通过对图像进行高斯模糊处理并计算原始图像与模糊图像之间的差分，可以获得图像的高斯差分图像。高斯差分图像能够突出图像中的边缘和细节部分，这对于后续的特征点提取非常重要。
* SIFT特征点提取：SIFT算法通过构建多尺度高斯金字塔，提取图像中的局部极值点作为特征点。随后，计算每个特征点的主方向，并统计特征点附近的梯度信息，形成特征向量。这些特征点和特征向量在不同尺度和旋转情况下具有稳定性。
* 特征点匹配：通过计算特征点特征向量的欧式距离，可以将两幅图像中的相似特征点匹配起来。匹配点之间的距离越小，说明这对特征点越可能是对应的。

1. 具体实验要求

实验步骤：

1. 两幅图像拼接

读取两幅待拼接的图像。

使用Stitcher类进行图像拼接。

保存拼接结果。

单独计算图像的高斯差分图像

1. 读取原始图像，并将图像类型转换为double。

对原始图像进行高斯模糊处理。

计算高斯差分图像。

对高斯差分图像进行灰度值归一化。

保存高斯差分图像。

提取SIFT特征点和特征向量

1. 读取两幅待匹配的图像。

使用SIFT\_create()函数提取图像的特征点及其特征向量。

在原始图像上绘制特征点。

保存绘制结果。

特征点匹配

1. 调用BFMatcher类，对特征向量集合进行匹配。

根据预设的ratio值筛选匹配点。

将相互匹配的特征点用线连接起来。

保存匹配结果。

四、实验过程与分析

1. 图像拼接

首先，读取读取图像，读取位于**sift** 文件夹中的多张图片，用于拼接。

import os

import sys

import numpy as np

import cv2 as cv

imgPath = 'sift\\'

imgList = os.listdir(imgPath)

imgs = []

for imgName in imgList:

pathImg = os.path.join(imgPath, imgName)

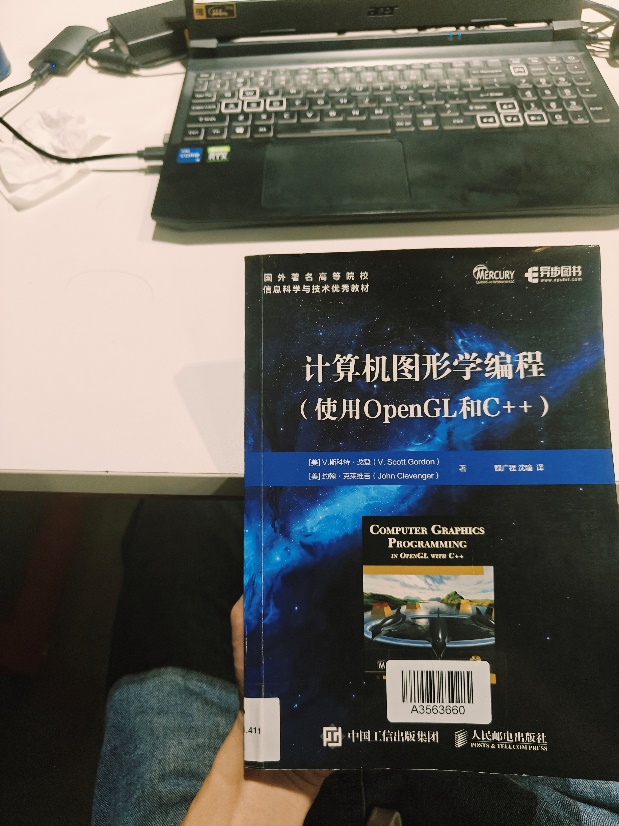
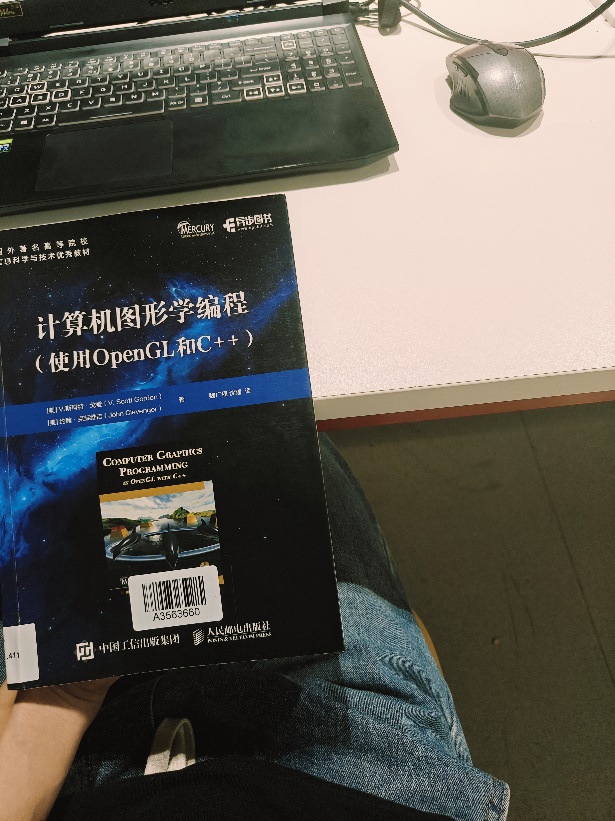
img = cv.imread(pathImg)

if img is None:

print("图片不能读取：" + imgName)

sys.exit(-1)

imgs.append(img)

原始图像p1 原始图像p2

接着，使用 OpenCV 的 Stitcher 类进行图像拼接，并将结果保存到 **result** 文件夹。

stitcher = cv.Stitcher.create(cv.Stitcher\_PANORAMA)

\_result, pano = stitcher.stitch(imgs)

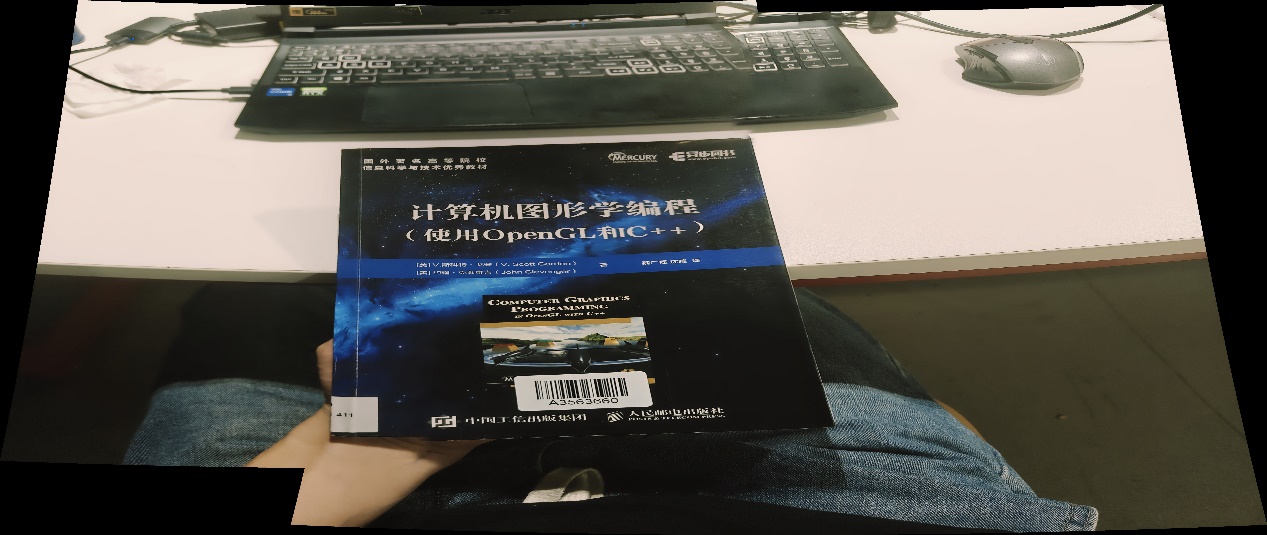
if \_result == cv.Stitcher\_OK:

cv.imwrite('result\\p12.png', pano)

print("拼接完成并成功保存为 'p12.png'")

else:

print("拼接失败：需要更多的图像")



拼接图像p12

1. 高斯差分

对图像进行高斯模糊，并计算差分图像。

im1 = cv.imread('sift\\p1.png')

im1 = (im1[:,:,0]).astype(np.double)

sz = 7

sig = 3

im\_gs = cv.GaussianBlur(im1, (sz, sz), sig)

im3 = im1 - im\_gs

进行灰度值归一化，将差分图像的灰度值归一化至 [0,255]。

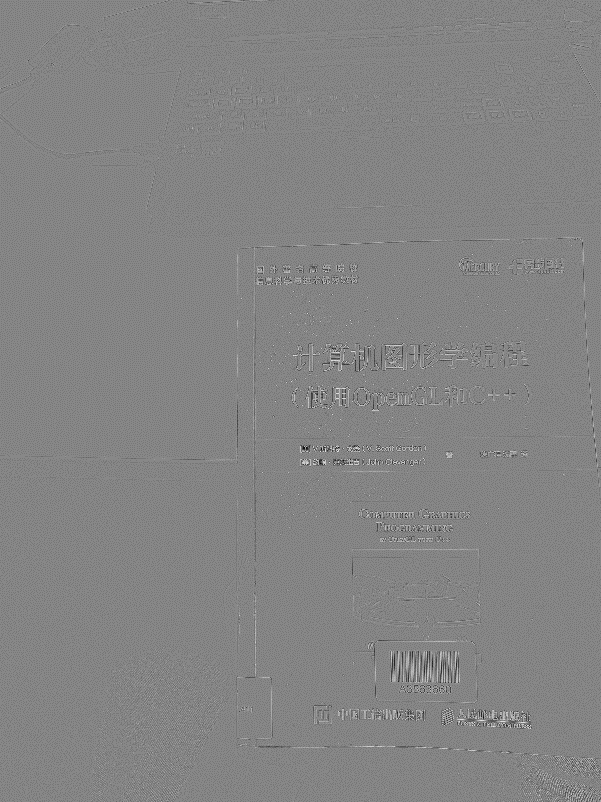
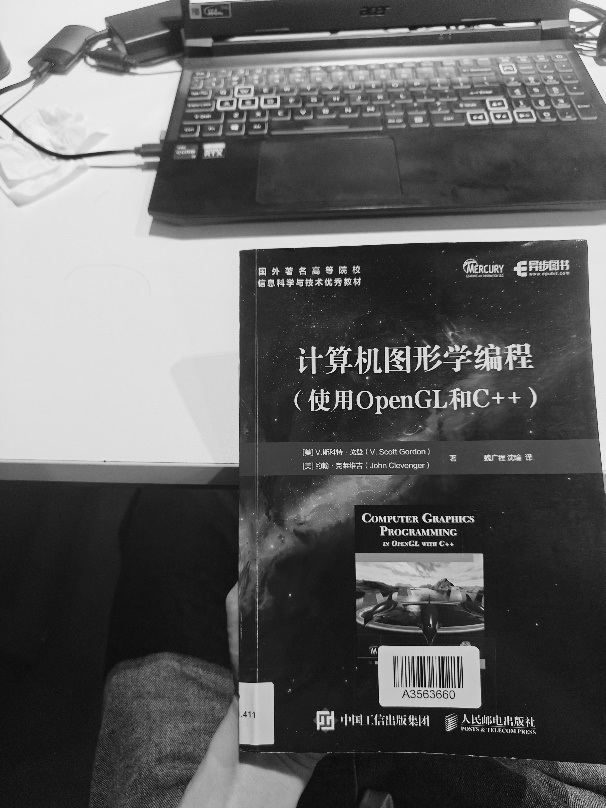
im3 = cv.normalize(im3, None, 0, 255, cv.NORM\_MINMAX)

最后将原图像、模糊图像和差分图像保存到 **result** 文件夹。

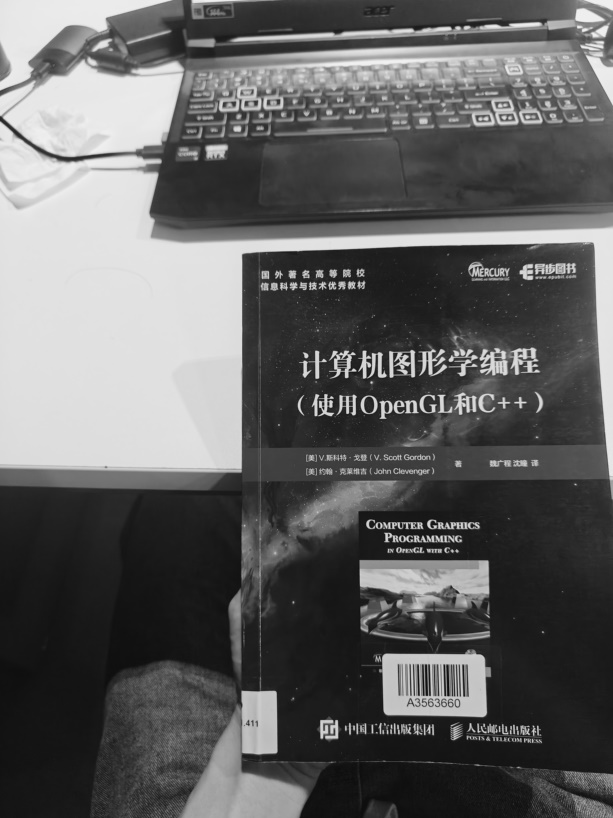
cv.imwrite('result\\s1-g.png', im1)

cv.imwrite('result\\s1-gs.png', im\_gs)

cv.imwrite('result\\s1-cf.png', im3)

差分图像 模糊图像



原图像

1. SIFT 特征点检测与匹配

先使用 SIFT 算法检测特征点，并绘制这些特征点。

sift = cv.SIFT\_create()

im1 = cv.imread('sift\\p1.png')

im2 = cv.imread('sift\\p2.png')

(kp1, des1) = sift.detectAndCompute(im1, None)

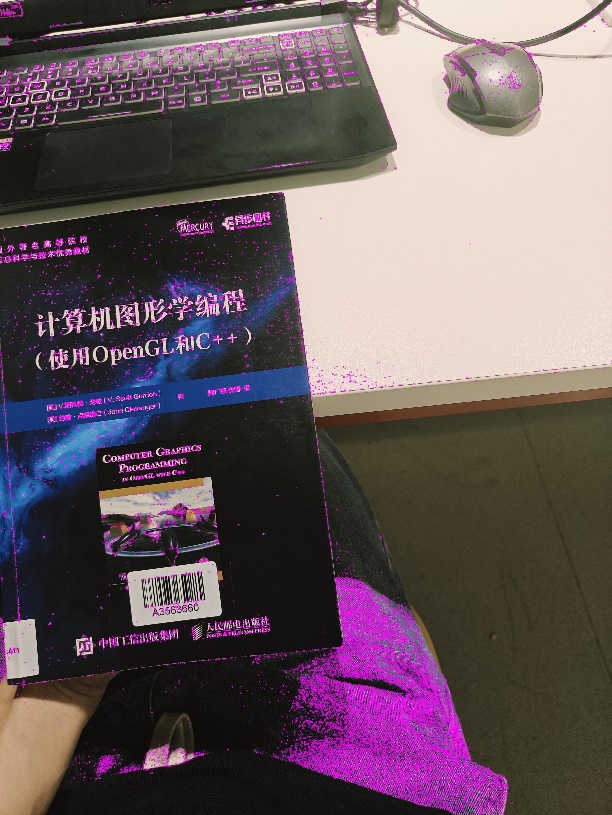
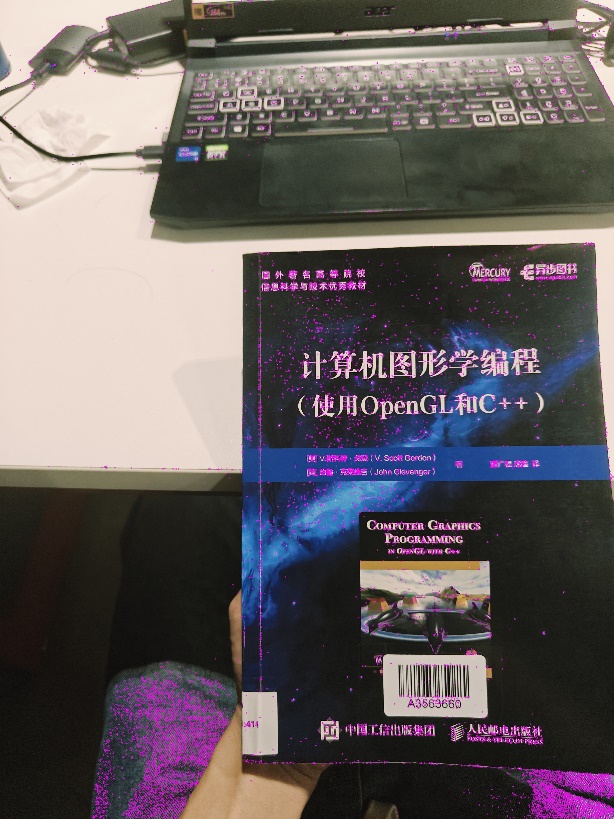
(kp2, des2) = sift.detectAndCompute(im2, None)

sift\_1 = cv.drawKeypoints(im1, kp1, None, color=(255, 0, 255))

sift\_2 = cv.drawKeypoints(im2, kp2, None, color=(255, 0, 255))

cv.imwrite('result\\sift\_1.png', sift\_1)

cv.imwrite('result\\sift\_2.png', sift\_2)

p1检测点 p2检测点

然后使用BFMatcher 进行特征点匹配，并绘制匹配结果。

bf = cv.BFMatcher()

matches1 = bf.knnMatch(des1, des2, k=2)

ratio1 = 0.5

good1 = []

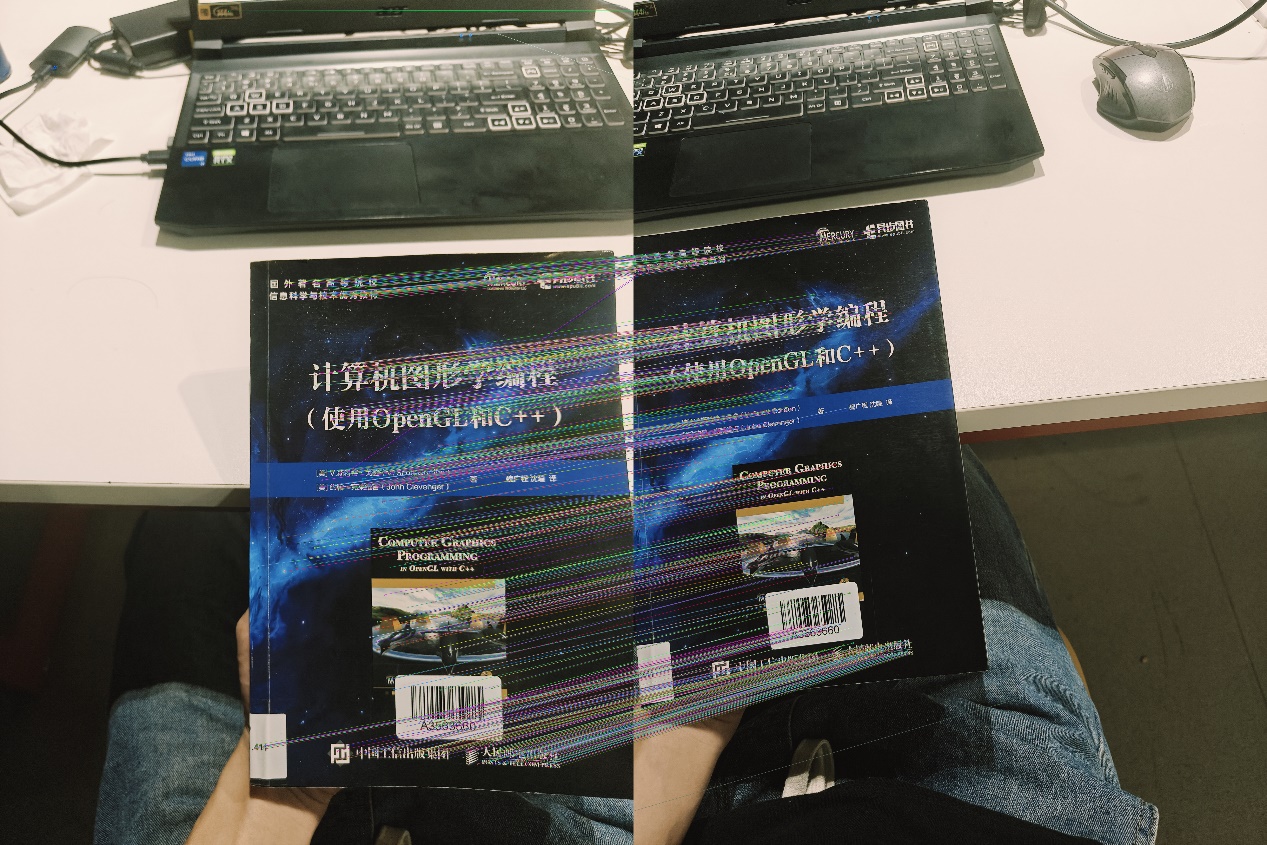
for m1, n1 in matches1:

if m1.distance < ratio1 \* n1.distance:

good1.append([m1])

match\_result1 = cv.drawMatchesKnn(im1, kp1, im2, kp2, good1, None, flags=2)

cv.imwrite("result\\sift\_1-2.png", match\_result1)

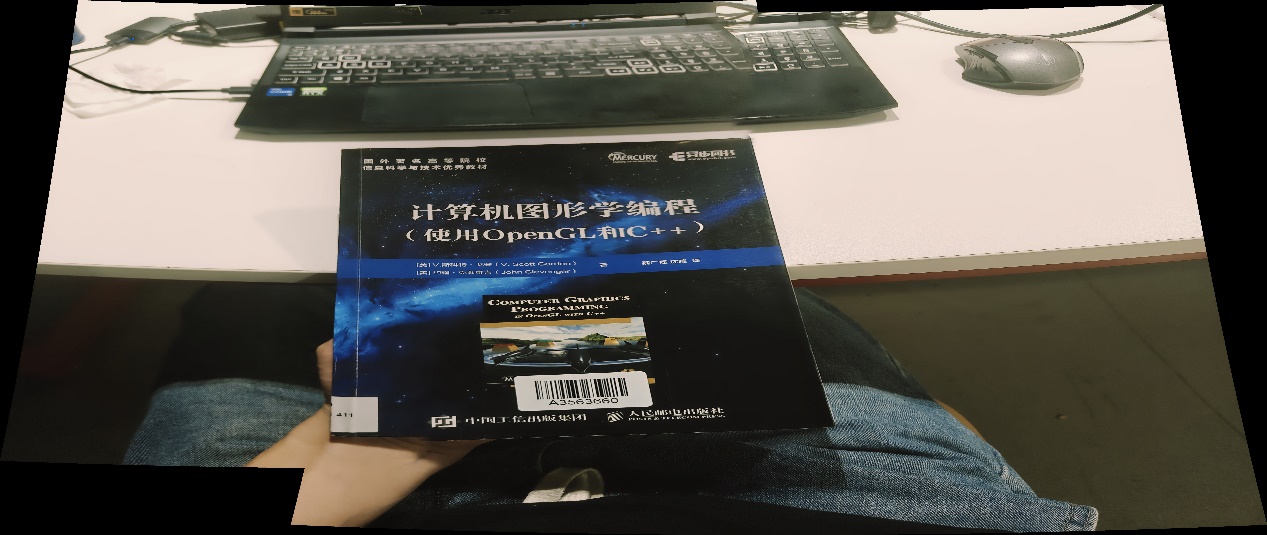


匹配结果图

五、实验结果总结

*对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的****改进意见****。（可多页）*

本次实验对两张示例图片进行拼接，具有良好的展示效果，基本完成了拼接需要，但仍存在部分区域扭曲的现象，说明sift算法对拼接图像的要求高，不具有很好的普适性。



**思考题**

1. 利用OpenCV中的Stitcher类，实现4张或以上图像的拼接。

输入四张原始图片

p1 p2

p3 p4

编写拼接算法，整体流程与实验一致，对图片读取改为4即可。

import os

import sys

import cv2 as cv

# 设置当前工作目录

os.chdir('D:\\WeChat Files\\wxid\_f44hm2h9kdox22\\FileStorage\\File\\2024-05\\实验二')

# 图像所在文件夹相对路径

imgPath = 'sift\\'

imgList = os.listdir(imgPath)

imgs = []

for imgName in imgList:

pathImg = os.path.join(imgPath, imgName)

img = cv.imread(pathImg)

if img is None:

print("图片不能读取：" + imgName)

sys.exit(-1)

imgs.append(img)

stitcher = cv.Stitcher.create(cv.Stitcher\_PANORAMA)

\_result, pano = stitcher.stitch(imgs)

if \_result == cv.Stitcher\_OK:

cv.imwrite('result1\\pano.png', pano)

print("拼接完成并成功保存为 'pano.png'")

else:

print("拼接失败：需要更多的图像或图像内容不足")

拼接后图像：



1. 结合3.2的代码和图2.3，计算一幅图像的高斯差分金字塔。请展示金字塔每一层中 最底层 图像的 高斯模糊和高斯差分效果。（σ=1.6，k=√2）

编写金字塔计算程序，代码如下：

import numpy as np

import cv2 as cv

def generate\_gaussian\_pyramid(img, num\_octaves, sigma=1.6, k=np.sqrt(2)):

gaussian\_pyramid = []

for octave in range(num\_octaves):

gaussian\_octave = []

for level in range(6):

if octave == 0 and level == 0:

gaussian\_img = img

elif level == 0:

gaussian\_img = cv.pyrDown(gaussian\_pyramid[octave-1][-3])

else:

gaussian\_img = cv.GaussianBlur(gaussian\_octave[level-1], (0, 0), sigma \* (k \*\* (level - 1)))

gaussian\_octave.append(gaussian\_img)

gaussian\_pyramid.append(gaussian\_octave)

return gaussian\_pyramid

def generate\_dog\_pyramid(gaussian\_pyramid):

dog\_pyramid = []

for gaussian\_octave in gaussian\_pyramid:

dog\_octave = []

for i in range(1, len(gaussian\_octave)):

dog\_img = cv.subtract(gaussian\_octave[i], gaussian\_octave[i-1])

dog\_octave.append(dog\_img)

dog\_pyramid.append(dog\_octave)

return dog\_pyramid

生成的图像金字塔如下所示：



1. 将找到的极值点以彩色点标注在原始图像上（来自金字塔第一层的极值点-红色，第二层-绿色，第三层-蓝色，第四层-紫色，第五层-黄色）

对思考题2生成的图像金字塔进行极值点标注，编写寻找极值点函数

def find\_extrema(dog\_pyramid, num\_octaves):

keypoints = []

for o in range(num\_octaves):

for i in range(1, len(dog\_pyramid[o]) - 1):

for x in range(1, dog\_pyramid[o][i].shape[0] - 1):

for y in range(1, dog\_pyramid[o][i].shape[1] - 1):

patch = [dog\_pyramid[o][i-1][x-1:x+2, y-1:y+2], # Accessing elements separately

dog\_pyramid[o][i][x-1:x+2, y-1:y+2],

dog\_pyramid[o][i+1][x-1:x+2, y-1:y+2]]

if np.amax(patch) == dog\_pyramid[o][i][x, y] or np.amin(patch) == dog\_pyramid[o][i][x, y]:

keypoints.append((o, i, x, y))

return keypoints

img = cv.imread('sift\\p1.png', cv.IMREAD\_GRAYSCALE)

num\_octaves = 5

gaussian\_pyramid = generate\_gaussian\_pyramid(img, num\_octaves)

dog\_pyramid = generate\_dog\_pyramid(gaussian\_pyramid)

for octave in range(num\_octaves):

cv.imwrite(f'result\\gaussian\_octave\_{octave+1}\_level\_1.png', gaussian\_pyramid[octave][0])

cv.imwrite(f'result\\dog\_octave\_{octave+1}\_level\_1.png', dog\_pyramid[octave][0])

img\_color = cv.imread('sift\\p1.png')

extrema = find\_extrema(dog\_pyramid, num\_octaves)

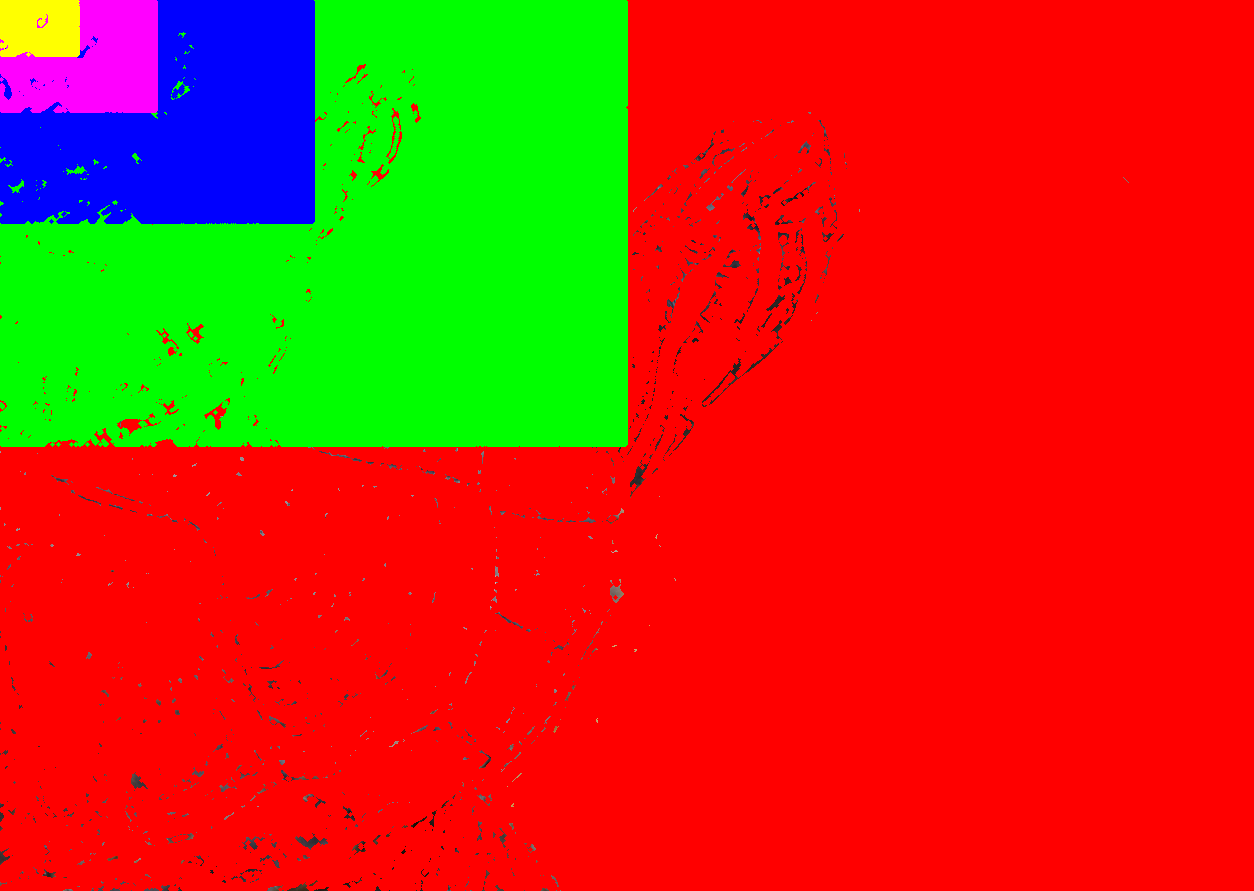
colors = [(0, 0, 255), (0, 255, 0), (255, 0, 0), (255, 0, 255), (0, 255, 255)]

for (o, i, x, y) in extrema:

color = colors[o % len(colors)] # Adjust color selection

cv.circle(img\_color, (y, x), 2, color, -1) # Correct order of (x, y)

cv.imwrite('result\\keypoints.png', img\_color)



1. 结合3.3和3.4的代码，计算能够相互匹配的特征点之间的平均欧式距离（特征向量之间的距离，而不是坐标位置的距离）

编写欧氏距离计算函数，计算欧式距离在拼接图像中展示

import numpy as np

import cv2 as cv

im1 = cv.imread('sift\\p1.png')

im2 = cv.imread('sift\\p2.png')

sift = cv.SIFT\_create()

kp1, des1 = sift.detectAndCompute(im1, None)

kp2, des2 = sift.detectAndCompute(im2, None)

bf = cv.BFMatcher()

matches = bf.knnMatch(des1, des2, k=2)

ratio = 0.5

good\_matches = []

for m, n in matches:

if m.distance < ratio \* n.distance:

good\_matches.append(m)

total\_distance = 0

for match in good\_matches:

total\_distance += match.distance

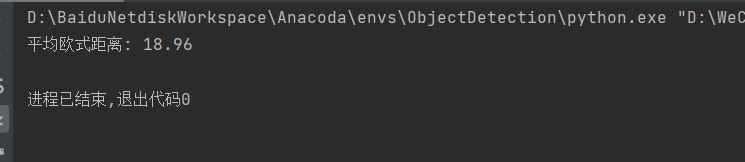
avg\_distance = total\_distance / len(good\_matches) if good\_matches else 0

print(f"平均欧式距离: {avg\_distance:.2f}")

match\_img = cv.drawMatches(im1, kp1, im2, kp2, good\_matches, None, flags=2)

cv.imwrite('result\\matches.png', match\_img)

平均欧氏距离计算结果为：18.96



连接图如下：



通过本次实验，我进一步了解和掌握了SIFT算法的原理及其在图像处理中的应用。通过高斯差分金字塔，能够更好地提取图像中的特征点，并通过这些特征点实现图像的拼接。同时，使用BFMatcher对特征点进行匹配，并计算特征点之间的平均欧式距离，有助于理解图像匹配的精度和鲁棒性。

在实际操作中，遇到了一些问题，如拼接失败和极值点的标注不准确。通过查阅资料和反复调试，最终解决了这些问题。