计算机视觉与应用实践实验报告（一）

目录

一．实验目的

- 理解关键点检测算法 DOG 原理。
- 理解尺度变化不变特征 SIFT。
- 采集一系列局部图像，自行设计拼接算法并使用 Python 实现图像拼接算法。

二．实验原理

下面分步骤介绍实现图像拼接所使用的算法原理。

2.1 图像特征点检测

完成图像拼接的首要任务是获取参考图像和待拼接图像的图像特征，这里使用 SIFT 算法进行图像特征点的检测。

2.1.1 SIFT 算法简介

SIFT(Scale-Invariant Feature Transform)，尺度不变特征转换特征检测算法，由 1999 年 David Lowe 首次提出，并在 2004 对该算法进行完善。SIFT 算法用于查找图像中的局部特征。SIFT 特征的查找是基于图像中物体上的一些局部特征的兴趣点完成的，这些点不受图像大小、平移和旋转等变换因素的影响，同时对于光线、视角与噪声等因素的敏感度较低。正是由于 SIFT 特征具有上述特性，导致其在图像中明显且容易获取，可以辨识图像且准确率较高。

该算法的主要思想是将图像之间的匹配转化成特征向量之间的匹配，关键是提取图像的局部特征，在尺度空间寻找极值点，提取其位置、尺度、旋转不变量。Lowe 等人将 SIFT 算法分解为如下四步：

Step1：尺度空间极值点检测，搜索所有尺度上的图像位置。通过高斯微分函数来识别潜在的对于尺度和旋转不变的兴趣点；

Step2：关键点定位，在每个候选的位置上，通过拟合一个精细的模型来确定位置和尺度。关键点的选择依据于它们的稳定程度。在关键点定位步骤中会剔除低对比度的候选点和边缘候选点。

Step3：方向确定，基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，从而提供对于这些变换的不变性。

Step4：关键点描述，在每个关键点周围的邻域内，在选定的尺度上测量图像局部的梯度。这些梯度被变换成一种表示，这种表示允许比较大的局部形状的变形和光照变化。

2.1.2 尺度空间极值点检测

尺度空间极值点检测，是以构建尺度空间用于原始图像的表示为基础，在不同的尺度空间下查找极值点，通过查询到的极值点信息锁定像素点位置，并将其作为候选的图像特征点，用于寻找不同的输入图像中（待拼接的图像）尺度未发生改变但是相对位置发生变化的目标物体。具体步骤如下：

- **生成尺度空间：**高斯卷积核是进行尺度变换的唯一变换核，一个二维图像的尺度空间定义为：

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{2.1}$$

式中$G(x,y,\sigma)$是高斯核函数：

$$G(x,y,\sigma) = \frac{\frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)}}{2\sigma^2} \tag{2.2}$$

式中，*表示卷积操作，(x, y)表示图像中像素的位置， σ 表示尺度空间的因子，通过改变其大小可以控制图像平滑的程度，值越大图像平滑的越厉害，相应的尺度也就越大。大尺度描述的是图像的概貌特征，小尺度描述的是图像的细节特征。

为了在不同尺度空间检测到相应的稳定关键点或称极值点，提出高斯差分尺度空间（Difference-of-Gaussian, DoG)。利用不同尺度的高斯差分函数与图像卷积生成。首先通过构建高斯金字塔模拟生成图像尺度空间。高斯金字塔使用参数(o,s)来表示，其中(o)和(s)分别称为组和层，(o)说明图片有多少不同的分辨率，即根据图片的分辨率进行组划分，(s)说明在每一组进行了多少次卷积。高斯金字塔需要高斯平滑和降采样两个过程反复操作来实现。将原始图像作为金字塔第一

组的第一层，将第一层图像进行高斯平滑后得到第二层，此时平滑因子为σ，取1.6。然后用 k σ 作为新的平滑因子构建高斯低通滤波器，k =$2^{1/s}$，与第一组的第二层卷积得到第三层，如此重复，获得每层相应的平滑因子为：0， σ , k σ ，k^2 σ .....，第一组的最后一层图像进行降采样获得下一组的第一层，再对该组内的每一层图像进行相应的高斯滤波，得到的高斯金字塔如图 2.1 所示：

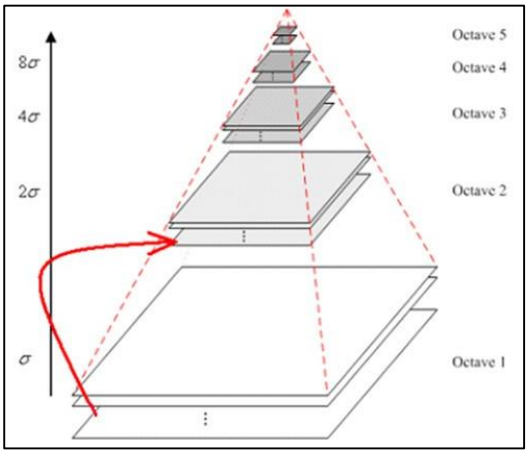

图 2. 1 高斯金字塔构建过程

接着在高斯金字塔的基础上生成高斯差分金字塔，高斯差分金字塔也存在组和层的概念，其中每一组对应高斯金字塔的一组，每一层是由它该组上一层与下一层的差得到，计算公式如式 2.3 所示，构造过程如图 2.2 所示。

$$D(x,y,\sigma) = \big(G(x,y,k\sigma) - G(x,y,\sigma)\big) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma) \quad (2.3)$$

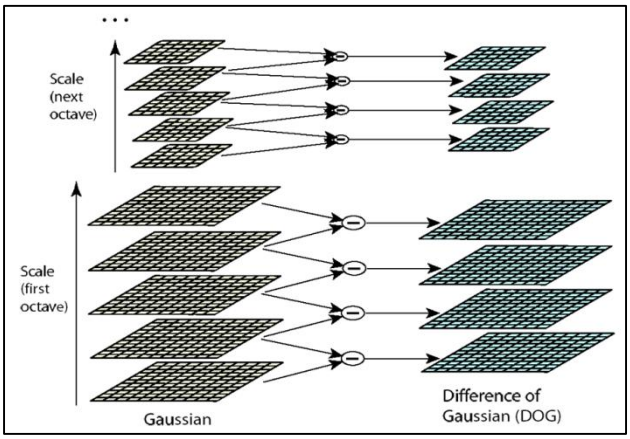

图 2. 2 高斯差分金字塔构建过程

- **极值点初步检测**：为了寻找尺度空间的极值点，每一个采样点要和它所有的相邻点比较，看其是否比它的图像域和尺度域的相邻点大或者小。如图 2.3 所示，将高斯差分图像中的每个像素与它的邻域、它对应上一层图像的邻域、对应下一层图像的邻域一共 26 个像素点比较，若其为最值，则记录此点的位置和

它所在的尺度，作为候选特征点也就是关键点。

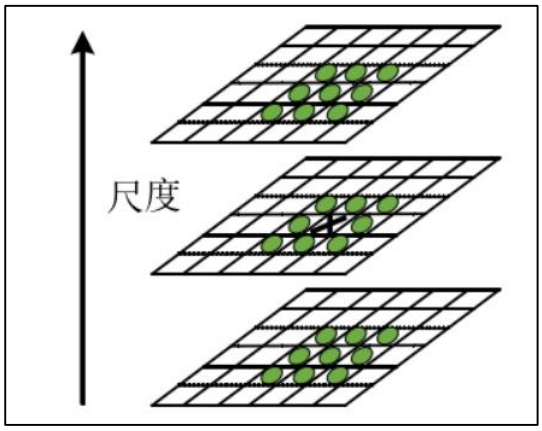

图 2. 3 DOG 尺度空间局部极值检测

2.1.3 关键点定位

上述方法检测到的是离散空间的极值点，并不是真正的连续空间特征点。由于某些极值点响应较弱，且 DOG 算子会产生较强的边缘响应，这样检测得到的极值点并非都是稳定的特征点。所以接下来需要精确定位极值点并去除边缘响应点，来增强匹配的稳定性、提高抗噪声能力。图 2.4 显示了二维函数离散空间得到的极值点与连续空间极值点的差别。

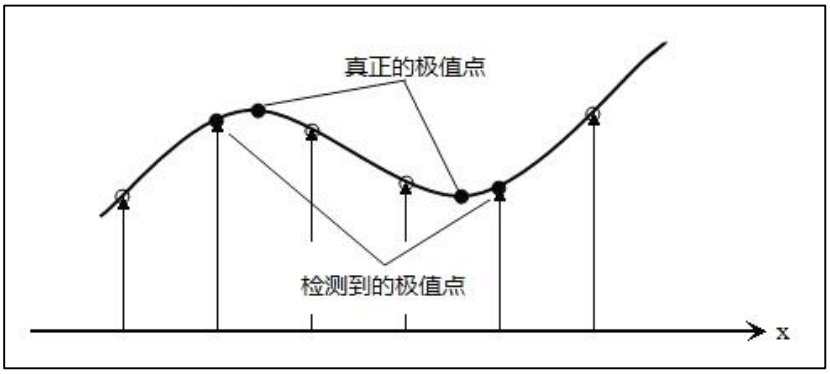

图 2. 4 离散空间与连续空间极值点分布

关于对比度差的极值点的剔除，在 SIFT 算法中，利用算子的响应值来体现图像中局部目标物体的像素点对比度，并且算子实际上是用于描述一个像素点与其邻域内像素点的差异，对比度大小与差异大小成正比。通过去除差异小的像素点来满足特征点的高可靠性。然而，在实际操作中，许多时候极值点都不一定能满足恰好在像素点上，而是在像素点邻域内的某一位置，即实际极值点的位置与理论上极值点的位置存在一个偏移量。为了消除这个偏移量，使得实际极值点与理论极值点位置重合达到高精度的要求，SIFT 算法中采用二阶泰勒展开式来拟合算子的响应曲线，若响应值的绝对值低于 0.03，则去除该极值点。

关于稳定性差的边缘极值点的剔除，通过计算边缘极值点的梯度，可以发现其梯度朝向唯一，会形成一个梯度主方向。在 SIFT 算法中，期望不同方向上的梯度值差异要小，通过设定一个阈值，挑选出梯度主方向与其他方向的比值大于阈值的极值点进行剔除。同时，被剔除的所有极值点具有一个共性，水平方向上图像曲率小，垂直方向上图像曲率大。由于 Hessian 矩阵的两个特征值恰好表示水平和垂直两个方向上的曲率大小，则可以利用两个特征值的比值来决定阈值的大小。具体的实现步骤如下：

Step1：对极值点集合中的每一元素计算图像在水平和垂直两个方向上的二阶偏导数以及混合偏导数，并构建 Hessian 矩阵；

Step2：计算出 Hessian 矩阵的两个特征值；

Step3：设定阈值大小，筛选极值点。

2.1.4 方向确定

在过滤掉对比度低的极值点和稳定性差的极值点后，为了达到 SIFT 要求的平移、旋转等变换保持不变性，则需要对每一个保留下的极值点进行方向确定，保证任意方向都具有较强的稳定性。对于每一个关键点，通过其四邻域可以得到对应位置的梯度幅度 *m(x, y)*和梯度方向 $\theta(x, y)$：

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y)^2) + (L(x,y+1) - L(x,y-1)^2)} \quad (2.4)$$

$$\theta(x,y) = tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \quad (2.5)$$

式中，尺度 L 为每个关键点各自所在的尺度。

实际计算时，为了凸显关键点的作用，以关键点为中心，用高斯窗对其邻域采样，并用直方图统计邻域像素的梯度方向。梯度方向的度数范围为[0，360°]，每 10°一个区间，共分为 36 个区间。其中，直方图的最大值代表关键点处邻域梯度的主方向，将其作为关键点的方向。

如图 2.5 所示，当计算方向直方图时，还使用一个参数等于 1.5 倍的关键点尺度的高斯直方图加权窗口，图 2.5 中用蓝色圈表示，使得中心的权重最大，边缘的权重很小。特征点的方向直方图中的峰值所指方向代表了特征点邻域梯度的方向，并作为特征点的主方向。要提高匹配的精度，某个方向梯度的峰值大于主方向梯度峰值的 80%作为辅助方向。所以，对于相同的梯度值会有多个峰值点位置，在相同的位置和尺度将超过一个关键点但有不同的方向。只有 15%的关键点被赋予多个方向，但可以明显改善关键点匹配的稳定性。

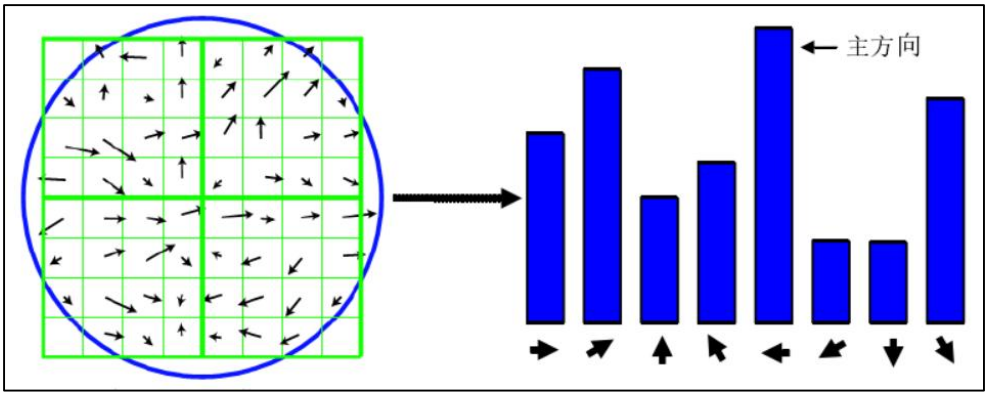

图 2. 5 关键点的方向直方图

这样，图像的关键点已经完成了检测，每个点包含三个信息：位置、尺度和方向。

2.1.5 关键点描述

为了使关键点更具有独特性和稳定性，使匹配更加精确，现在要为每个关键点建立一个描述子，也即图像的特征向量，用来描述关键点附近邻域的梯度信息。通过对关键点周围图像区域分块，计算块内梯度直方图，生成具有独特性的向量，称为特征点描述子向量，这个向量是多维特征向量，是该区域图像信息的一种抽象，具有唯一性。特征描述符的生成大致有三个步骤：

1. 校正旋转主方向，确保旋转不变性：以特征点为中心，在附近邻域内将坐标轴旋转 θ (特征点的主方向)角度，即将坐标轴旋转为特征点的主方向。旋转后邻域内像素的新坐标为：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad (x, y \in [-r, r])$$

2. 生成描述子，最终形成一个 128 维的特征向量：旋转后以主方向为中心取 8×8 的窗口。如图 2.6 所示，左图的中央为当前关键点的位置，每个小格代表为关键点邻域所在尺度空间的一个像素，求取每个像素的梯度幅值与梯度方向，箭头方向代表该像素的梯度方向，长度代表梯度幅值，然后利用高斯窗口对其进行加权运算。最后在每个 4×4 的小块上绘制 8 个方向的梯度直方图，计算每个梯度方向的累加值，即可形成一个种子点，如右图所示。每个特征点由 4 个种子点组成，每个种子点有 8 个方向的向量信息。这种邻域方向性信息联合增强了算法的抗噪声能力，同时对于含有定位误差的特征匹配也提供了比较理性的容错性。与求主方向不同，此时每个种子区域的梯度直方图在 0-360° 之间划分为 8 个方向区间，每个区间为 45 度，即每个种子点有 8 个方向的梯度强度信息。

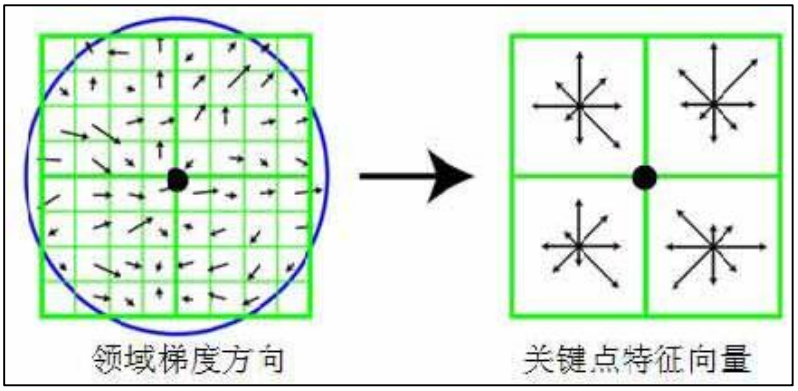

图 2. 6 由关键点邻域梯度信息生成特征向量

在实际的计算过程中，为了增强匹配的稳健性，如图 2.7 所示对每个关键点使用 4×4 共 16 个种子点来描述，这样一个关键点就可以产生 128 维的 SIFT 特征向量。

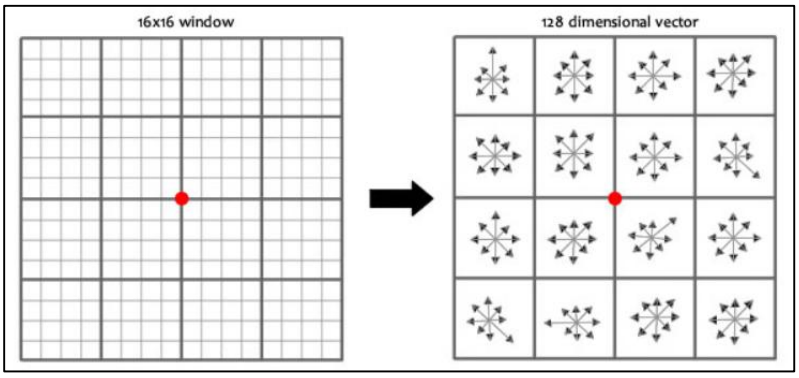

图 2. 7 128 维特征向量

3. 归一化处理：将特征向量长度进行归一化处理，进一步去除光照的影响。

通过对特征点周围的像素进行分块，计算块内梯度直方图，生成具有独特性的向量，这个向量是该区域图像信息的一种抽象，具有唯一性。至此便得到了 SIFT 特征点与特征向量。

2.2 图像特征点配准

使用 SIFT 算法分别在参考图像和待拼接图像提取特征后，需要根据特征点对向量对两幅图像的相同区域进行匹配，找到重合区域的特征点。特征点匹配目的是通过特征点的坐标关系建立两幅图像之间的坐标关系，即投影关系模型。再将两幅不同图像的坐标映射到指定的同一坐标空间，完成图像拼接。

2.2.1 SIFT 特征点粗匹配算法

在进行特征点匹配时，首先应对相似性进行度量。匹配过程中，使用欧氏距离最小距离与次小距离的比值与阈值进行比较。如果该比值小于阈值，那么说明匹配正确，若比值大于阈值说明匹配失败，丢弃这对点。假设参考图像中某个特

征点的特征向量设为 $T_i=(t_{i1},t_{i2},...,t_{i128})$，图像特征点的描述符向量表示为 $S_i=(s_{i1},s_{i2},...,s_{i128})$，欧氏距离的计算公式如 2.6 所示：

$$d(T_i,S_i)=\sqrt{\sum_{k=1}^{128}(t_{ik}-s_{ik})^2} \tag{2.6}$$

根据上述公式可以算出最小距离 d_1，和次小距离,，设阈值为 r，当 $d_1/d_2\leqslant r$ 时，(T_i,S_i)是一对匹配点。

确定了两个匹配点的相似性度量算子后，使用 k-d 树对每个特征点最临近的特征点进行优先查找。k-d 树，即 K-Dimensional Tree，是一种高维索引树型数据结构。常用于大规模高维数据空间的邻近或者 K 邻近查找。

2.2.2 SIFT 特征点精确匹配算法

使用 K-D Tree 对特征点集进行第一次匹配处理后，由于光照、设备、角度的影响会使图像上像素点的灰度和几何结构产生极大的相似匹配，造成了误匹配现象。如果利用误匹配的点直接计算得到参数模型，会使图像配准不准确，影响接下来的融合处理及拼接效果。所以，在初步得到匹配点后，使用 RANSAC 算法剔除误匹配点，进行精确匹配。

RANSAC (Random Sample Consensus)算法，即随机抽样一致算法，可用于基于特征点匹配的数据提纯。RANSAC 算法使用迭代的方式从一群离散型数据群中计算出最能描述该数据群的数学模型及参数。在样本数据群中包括正确数据，即那些可以被该同一参数模型描述的数据，也同样包括一些偏离正常范围，不能被参数模型描述的数据。这些偏离正常范围的数据点可能是由于试验中错误的测量、计算方式造成。算法思想如下：

设有数据集合 P，包含 N 个数据点，它们中绝大部分的数据点都包含在一个参数模型中，这些点可以被该模型描述，我们假设数据点的数量为 n，n>N。通过 n 个数据点可以求出这个模型的参数。步骤如下：

1. 从数据集合 P 中随机抽取 n 个数据点作为样本子集 M；

2. 通过样本子集 M 中的 n 个数据点计算出可以描述 M 中绝大部分数据点的数学参数模型 S；

3. 计算数据集合 P 中其余 N-n 个数据点与数学模型 S 之间的距离。设定一个阈值，记录在阈值范围内的 P 中数据点的个数 m。

4. 重复循环执行步骤 1 至 3，其中对应 m 最大值的模型即为所求模型。

2.3 图像融合

在完成图像配准后，进行图像融合操作。由于用来拼接的原始图像拍摄角度、设备可能不同，直接对两张图像进行简单的叠加拼合，得到的结果图像会在拼接位置上存在明显的裂缝，重叠区域附近会出现模糊，甚至失真现象。因此，采用加权平均法进行图像的融合操作，用以消除图像色彩或光照强度的在拼接过程中出现的不连续性。

加权平均法将图像重叠区域的像素赋予了权重，与图像重叠区的像素进行相乘，将结果累加，合成新的图像。可以实现重叠区域的像素值平滑过渡，消除图像由于拼接产生的缝隙。其中心思想即为对两幅待拼接图像的重叠区域分配初始权重，并且权重的具体数值依据像素点的位置进行动态调整，而非重叠区域的像素值保持不变。

假设 $F_1(x_1,y_1)$和 $F_2(x_2,y_2)$是待融合的两幅图像分别在点(x_1,y_1)、(x_2,y_2)处的像素值，$F_3(x_3,y_3)$是融合后的图像在(x_3,y_3)处的像素值。W_{F1} 和 W_{F2} 分别是待融合的两幅图像在重叠区域的权值，则融合时应当满足式 2.7。

$$F_3(x_3,y_3)=\begin{cases}F_1(x_1,y_1),(x_3,y_3)\in F_1\\ w_{F_1}F_1(x_1,y_1)+w_{F_2}F_2(x_2,y_2),(x_3,y_3)\in F_1\cap F_2\\ F_2(x_2,y_2),(x_3,y_3)\in F_2\end{cases} \tag{2.7}$$

其中，权值 W_{F1} 和 W_{F2} 与两幅图像重叠区域的大小有关，且 W_{F1} 和 W_{F2} 满足 $0< W_{F1}, W_{F2}<1$，$W_{F1}+W_{F2}=1$。实际操作时，在两幅图像的重叠区域中，让 W_{F1} 从 1 逐渐减小到 0，W_{F2} 从 O 逐渐增加到 1，从而实现权值的平滑过渡。

三．实验步骤

实验步骤如图 3.1 所示，首先使用 SIFT 算法提取参考图像与待拼接图像的特征，接着进行特征点匹配，最后进行图像融合得到拼接后的图像。

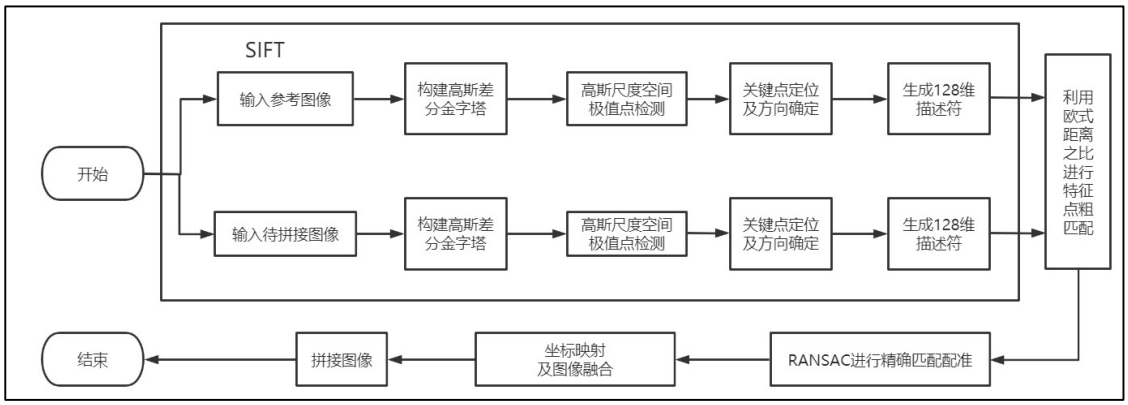

图 3. 1 实验步骤

四．数据集

如图 4.1 所示，实验使用手机拍摄的两张校园图片作为输入数据，包含部分重叠区域，水平与垂直方向上各有移动。

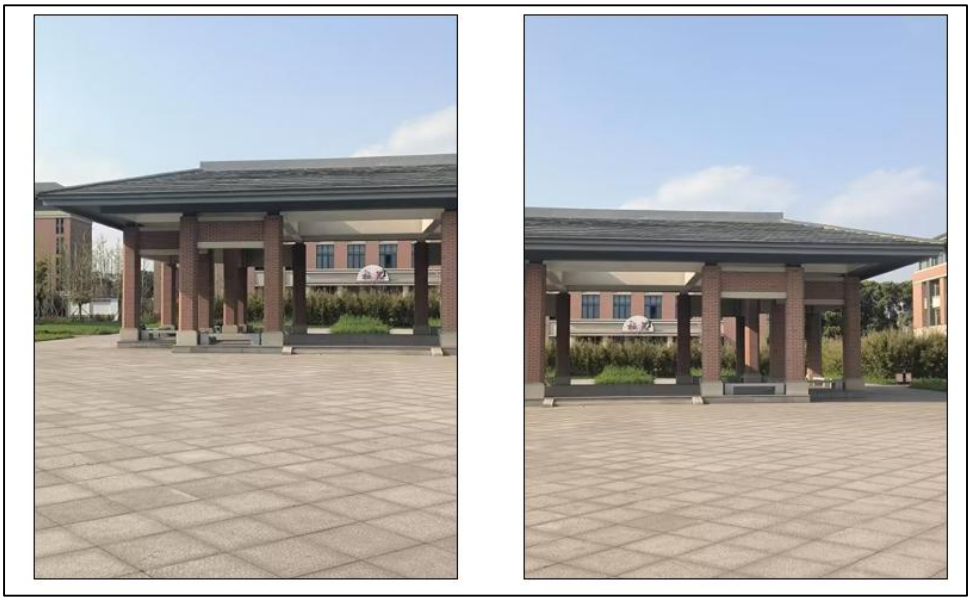

图 4. 1 数据集

五．程序代码

本小节给出图像拼接程序的关键代码的实现与注释，以及相关代码运行的中间结果示意图。

- 高斯金字塔：使用 opencv 库提供的 API 对输入图像进行高斯模糊，继而下采样得到下一层图像，如此重复得到输入图像的高斯金字塔。图 5.1 代码所示，计算出高斯金字塔的第一组图像并打印。

```
# 高斯金字塔示意图
f, ax = plt.subplots(2, 3, figsize=(12, 8))
ax[0, 0].imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
ax[0, 0].set_title("original")
ax[0, 0].set_xticks([])
ax[0, 0].set_yticks([])

for i in range(1, 6):
    # 高斯模糊
    image_blur = cv2.GaussianBlur(img1, (15, 15), i)
    # 计算子图
    ax[int(i/3), i % 3].imshow(cv2.cvtColor(image_blur, cv2.COLOR_BGR2RGB))
    ax[int(i/3), i % 3].set_title("σ" + str(i))
    ax[int(i/3), i % 3].set_xticks([])
    ax[int(i/3), i % 3].set_yticks([])
plt.show()
```

图 5. 1 高斯金字塔实现

如图 5.2 所示，得到输入图像的第一层高斯金字塔，可发现随着高斯核尺寸的增加，图像变得越来越模糊。

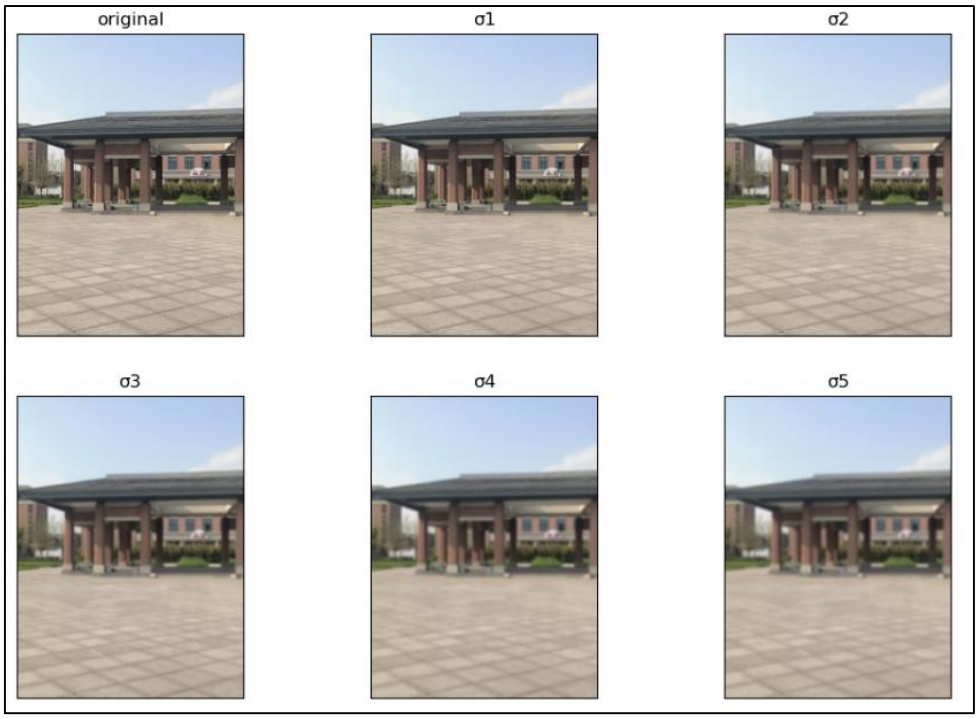

图 5.2 高斯金字塔第一层

- 高斯差分金字塔：得到输入图像的高斯金字塔后，对其每层进行两两做差，可得高斯差分金字塔。图 5.3 代码所示，计算出高斯金字塔的第一层的两个高斯差分图并打印。

```
# 高斯差分金字塔示意图
image_blur1 = cv2.GaussianBlur(img1, (15, 15), 1) # 一次高斯模糊
image_blur2 = cv2.GaussianBlur(img1, (15, 15), 2) # 二次高斯模糊
DOG_12 = image_blur1 - image_blur2                # 高斯差分图1-2
image_blur7 = cv2.GaussianBlur(img1, (15, 15), 7) # 七次高斯模糊
image_blur8 = cv2.GaussianBlur(img1, (15, 15), 8) # 八高斯模糊
DOG_78 = image_blur7 - image_blur8                # 高斯差分图7-8
# 绘图
title = ['σ1', 'σ2', 'σ1-σ2', 'σ7', 'σ8', 'σ7-σ8']
images = [image_blur1, image_blur2, DOG_12, image_blur7, image_blur8, DOG_78]
f, ax = plt.subplots(2, 3, figsize=(12, 8))
for i in range(6):
    ax[int(i/3), i % 3].imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
    ax[int(i/3), i % 3].set_title(title[i])
    ax[int(i/3), i % 3].set_xticks([])
    ax[int(i/3), i % 3].set_yticks([])
plt.show()
```

图 5.3 高斯差分金字塔实现

如图 5.4 所示，得到了高斯金字塔的第一层的两个高斯差分图。每个高斯差分图都是由高斯金字塔每层中相邻的两张高斯模糊图做差得到。

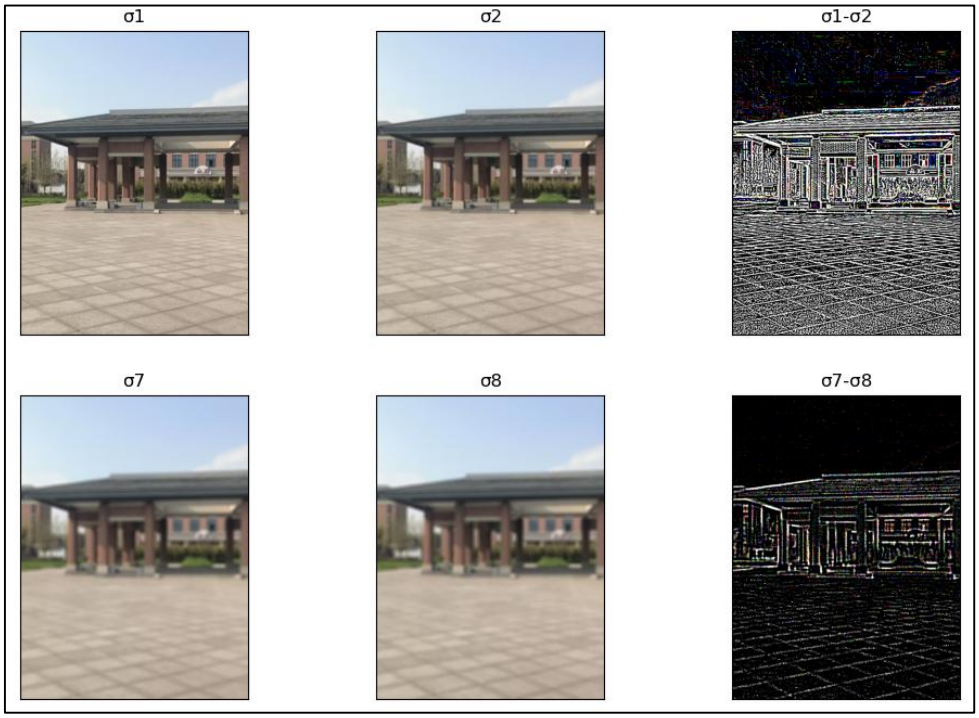

图 5.4 高斯差分图

- SIFT 特征点检测：如图 5.5 代码所示，使用 opencv 库中自带的 SIFT 特征检测器进行图像的特征点检测，最后得到 128 维的特征点向量。

```
# 检测sift特征点
sift = cv2.SIFT_create() # 使用cv2库中自带的sift特征检测器
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
print(des1.shape, des2.shape) # 128的特征向量
des1
```

```
(1646, 128) (1436, 128)
```

图 5.5 SIFT 特征点检测

- SIFT 特征点粗匹配：如图 5.6 代码所示，使用 KDTree 进行 SIFT 特征向量的欧式距离最近次近点对查询，对特征点进行筛选，保留最近邻距离小于 0.75 倍次近邻距离的点。

```
from scipy.spatial import KDTree
# 使用KDTree进行最近邻查询
tree = KDTree(des2)
dist, indices = tree.query(des1, k=2)
# 对特征点进行筛选，保留最近邻距离小于0.75倍次近邻距离的点
good_indices = np.where(dist[:, 0] < 0.75 * dist[:, 1])[0]
matches = []
for idx in good_indices:
    matches.append(cv2.DMatch(idx, indices[idx][0], dist[idx][0]))
```

图 5.6 特征点粗匹配

- SIFT 特征点精确匹配：如图 5.7 代码所示，使用 RANSAC 算法进行 SIFT 特征点的精确匹配，并计算出待拼接图像对源图像的透射变换矩阵，用于后续图像融合。

```
# 使用RANSAC算法进行精确匹配
src_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)
M, mask = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC, 5.0)
M #透视变换矩阵
```

```
array([[ 8.32967122e-01,  3.49863140e-02,  3.81599198e+02],
       [-1.33166513e-01,  1.00829504e+00, -7.72634255e+01],
       [-2.80048841e-04,  4.98799707e-05,  1.00000000e+00]])
```

图 5.7 特征点精确匹配

- 图像融合：如图 5.8 代码所示，为加权平均法图像重叠区域像素值权重初始化的实现。创建用于图像拼接的掩码，实现边缘渐变融合效果，使左右两个图像之间的过渡更加平滑。

```
def create_mask(img1, img2, version):
    """
    创建用于图像拼接的掩码，实现边缘渐变融合效果。
    Args:
        img1: 第一幅图像。
        img2: 第二幅图像。
        version: 当前图像是左图还是右图，取值为'left_image'或'right_image'。
    Returns:
        一个形状为`(height_panorama, width_panorama, 3)`的掩码数组，其中每个元素的值表示对应像素在拼接后的图像中的透明度
    """
    # 获取输入图像的大小信息
    height_img1 = img1.shape[0]
    width_img1 = img1.shape[1]
    width_img2 = img2.shape[1]
    height_panorama = height_img1
    width_panorama = width_img1 + width_img2

    # 创建大小为`(height_panorama, width_panorama)`的全0数组作为掩码
    mask = np.zeros((height_panorama, width_panorama))

    # 设置渐变区域的大小和位置
    smoothing_window_size = 20
    offset = int(smoothing_window_size / 2)
    barrier = img1.shape[1] - int(smoothing_window_size / 2)

    # 如果当前图像是左图
    if version== 'left_image':
        mask[:, :barrier - offset] = 1  # 将`mask`的左半部分设为1
        # 右半部分使用线性渐变将值从1逐渐降低到0，使得左图的边缘逐渐融合到右图中
        mask[:, barrier - offset: barrier + offset ] = np.tile(np.linspace(1, 0, 2 * offset).T, (height_panorama, 1))
    # 如果当前图像是右图
    else:
        mask[:, barrier + offset:] = 1  # 将`mask`的右半部分设为1
        # 左半部分使用线性渐变将值从0逐渐增加到1，使得右图的边缘逐渐融合到左图中
        mask[:, barrier - offset :barrier + offset ] = np.tile(np.linspace(0, 1, 2 * offset).T, (height_panorama, 1))

    # 将掩码沿着通道方向复制3次，得到形状为`(height_panorama, width_panorama, 3)`的掩码
    return cv2.merge([mask, mask, mask])
```

图 5.8 像素权重初始化

如图 5.9 代码所示，使用加权平均法对两张图像进行融合，实现图像连接处平滑过渡。

```
# 创建全零矩阵，大小为拼接后全景图像的尺寸，维度为3，RGB图像
panorama1 = np.zeros((height_panorama, width_panorama, 3))
# 创建掩码数组
mask1 = create_mask(img1, img2, version='left_image')
# 将左图img1复制到全景图像的左边，与左边界对齐
panorama1[0: img1.shape[0], 0: img1.shape[1], :] = img1
# 左图像素与掩码相乘，平滑像素
panorama1 *= mask1
# 创建右图img2的掩码矩阵
mask2 = create_mask(img1, img2, version='right_image')
# 使用透视变换矩阵M将右图img2进行透视变换，并将变换后的图像乘以掩膜矩阵mask2
panorama2 = cv2.warpPerspective(img2, M, (width_panorama, height_panorama)) * mask2
# 将左图img1与经过透视变换的右图img2融合
result = panorama1 + panorama2
result = result.astype('uint8')
```

图 5. 9 加权平均法图像融合

六．实验结果

图 6.1 所示为源图与待拼接图像的 SIFT 特征点。图 6.2 所示为源图与待拼接图像的 SIFT 特征匹配结果。

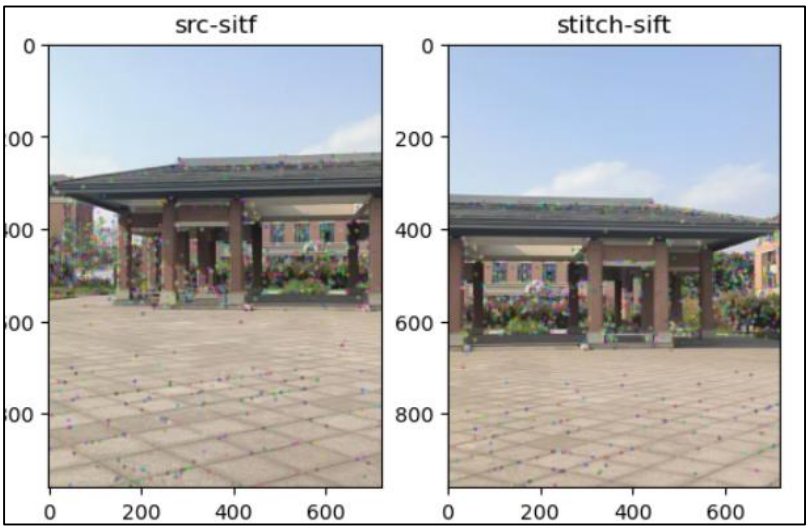

图 6. 1 SIFT 特征点

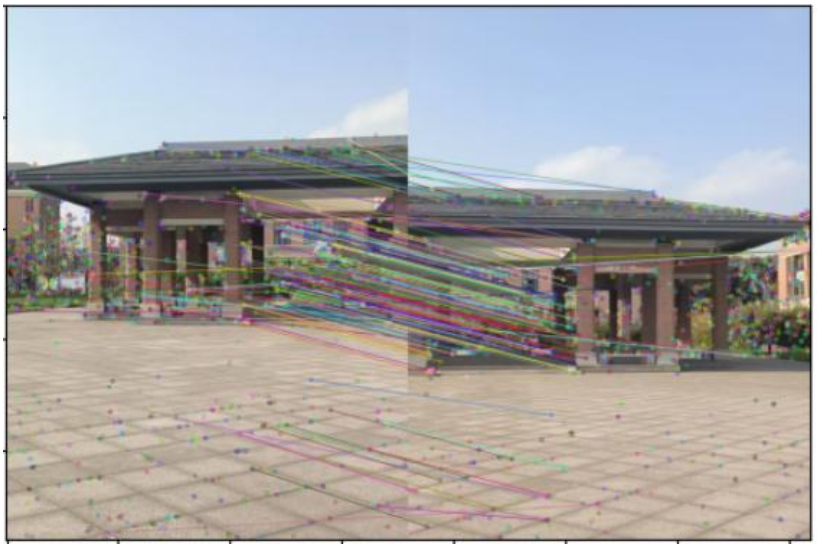

图 6. 2 SIFT 特征点匹配

图 6.3 所示，为最终的拼接图像。

图 6. 3 拼接结果

七．实验分析与总结

7.1 分析

拼接结果出现黑边框现象，原因是透视变换导致图像变形：当进行透视变换时，不同部分的图像在拼接后可能会出现不同程度的拉伸和扭曲，从而导致边缘处出现黑边框。这是因为透视变换会使得原来在图像中心的像素被映射到边缘区域，从而导致边缘处出现黑边框。

在使用 SIFT 算法进行图像拼接实验时，首先需要提取每个图像的 SIFT 特征点，并利用 SIFT 算法计算它们的描述符。然后，需要将两幅图像的特征点进行匹配，以便找到它们之间的对应关系。在此过程中，可以使用基于距离或基于比值的匹配方法。匹配完成后，需要使用 RANSAC 算法估计两个图像之间的单应性矩阵，并将它们拼接在一起。

在实验过程中，得出以下三条注意点：

SIFT 算法对于尺度和旋转变换是不变的，但对于亮度变化和仿射变换是敏感的；

SIFT 算法中的关键参数包括高斯金字塔的层数、高斯核的大小、特征点的阈值等。这些参数的设置会影响特征点的提取和匹配结果；

匹配过程中存在误匹配的情况，因此需要使用 RANSAC 算法来筛选出正确的匹配点，以提高图像拼接的准确性。

7.2总结

通过实验，理解了关键点检测算法 DOG 原理和尺度变化不变特征 SIFT，并基于 SIFT 算法实现了图像拼接程序，完成了实验目的。

在实验过程中，发现 SIFT 算法可以有效地提取图像的特征点，并且能够在多个尺度和旋转角度下保持稳定性。此外，SIFT 算法还能够准确地进行特征点匹配，从而实现图像的拼接。

然而，SIFT 算法也存在一些缺点。首先，SIFT 算法的计算量较大，因此对于大型图像的处理可能会比较慢。其次，SIFT 算法对于光照和视角变化比较敏感，可能会影响特征点的提取和匹配。

总的来说，SIFT 算法是一种非常有用的图像处理算法，特别是在图像拼接方面具有很大的应用潜力。在使用 SIFT 算法时，需要注意算法的计算量和对光照和视角变化的敏感性，并根据实际情况进行参数的调整，以达到更好的效果。