计算机视觉应用与实践实验报告

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

SIFT 算法的核心是在不同尺度空间上查找图像的特征点，也称关键点，并对特征点进行匹配的问题，所谓关键点是一些在图片中表现比较突出的点，是一些带有方向的且相对稳定的局部极值点，这些点不会因光照条件、拍摄角度、噪声等的影响而消失。既然这些关键点不会因条件的改变而消失，那么就可以通过算法来检测图片之间的相互匹配的关键点,然后对图片进行拼接。其实现主要包括以下几个步骤：
1. 构建 DOG 尺度空间 2.寻找关键点 3.确定关键点的方向 4.对关键点进行描述 5.关键点匹配 6.图像拼接

2.1 构建 DOG 尺度空间：

图像的尺度空间是一幅图像在不同解析度下的表示。把原始图像作为输入通

过高斯滤波产生几组(octave)图像，一组图像里面可以包含多张图像。构造尺度空间传统的方法即构造一个高斯金字塔,把原始图像作为金字塔的最底层，然后对图像进行高斯模糊再降采样(2 倍)作为下一层图像,由于高斯模糊降低了图像的大小,为了避免虚假的高频信息出现在下采样的图像中，一般在降采样之前先对图像进行低通滤波处理,由此循环迭代下去。构建尺度空间其实就是模拟图像在人的视觉上由近及远形成的过程,所以尺度空间越大,图像的尺寸越大,图像越模糊。构建尺度空间是初始化操作,二维图像的尺度空间定义为:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

其中 G 是二维高斯函数：

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

公式中*为卷积运算符号，x 和 y 分别代表图像像素的横坐标和纵坐标，σ是尺度参数。

由于 SIFT 只能处理灰度图像,I 是目标图像经过转换成的灰度图像，再经过高斯卷积后，就可以得到已付滤波后的图像，并且σ为第三维(尺度坐标)，这样就构成了二维图像的空间。图像的平滑程度由σ大小来决定，σ越大图像越平滑，所以调大σ即提高了周边像素对中心点的影响程度,图像越模糊，而σ越小图像越清晰,所以大尺度可以检测到图像概貌特征,小尺度检测到的是图像的细节特征。因此金字塔越靠上其σ越大，即低分辨率需要用大σ来产生更多细节特征。

然后将每一层（octave）上两个相邻的高斯图像进行差分，就可以得到高斯差分尺度空间函数(DOG 函数)：

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

S 为每一组的层数（一般为 3~5），金字塔每一层的 σ 与 k 定义如下：

$$2^{i-1}(\sigma, k\sigma, k^2\sigma, \cdots, k^{n-1}\sigma) = 2^{1/s}$$

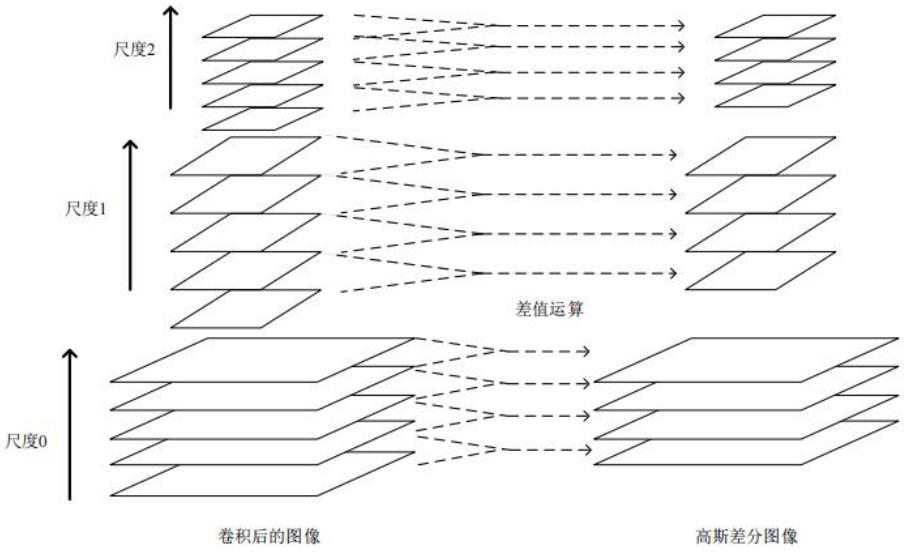


图 2.1 高斯差分金字塔模型

根据图 2.1 可知，尺度空间的构建分两步进行。首先，对输入图像进行高斯卷积操作，并迭代多次，于此同时，对每一阶段的图像进行缩放。然后，对相邻阶段的卷积图像做差值处理，形成不同尺度下的尺度空间。

2.2 寻找关键点：

寻找关键点的第一步是求极值操作。如下图 2.2 所示，首先，需要对高斯差值金字塔中的每一个像素的邻域像素进行提取，并与之比较，找出不同尺度下的灰度值的极值点并将其所在的尺度信息和像素位置信息予以记录。

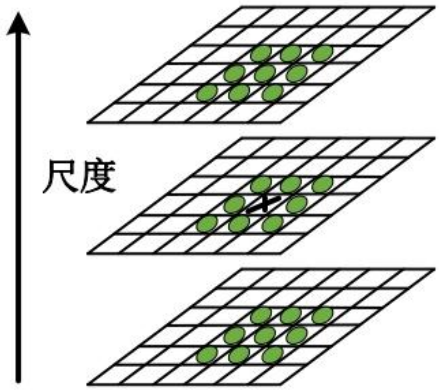


图 2.2 像素点位置示意图

然而，上述过程求解出的极值点并非都是关键点，接下来还需要对该极值点集合进行筛选过滤，剔除掉一些对比度差和稳定性差的点。

关于对比度差的极值点的剔除，由于在 SIFT 算法中，是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。则，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。如下图 2.3，该图描述了实际检测过程中的极值点关系。

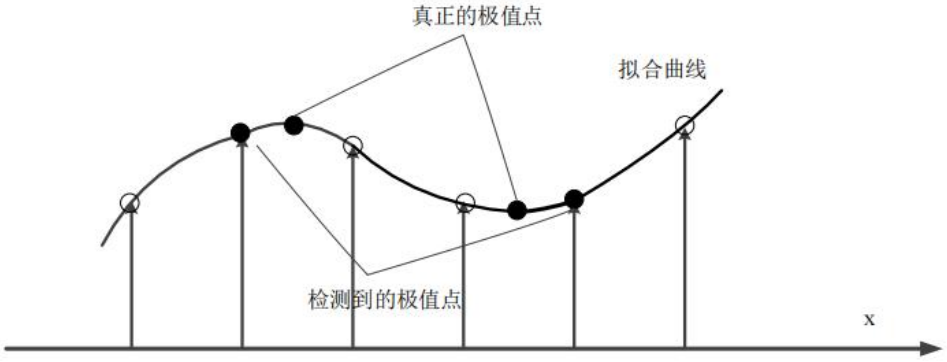


图 2.3 实际极值点分布示意图

关于稳定性差的边缘极值点的剔除，通过边缘极值点的梯度计算，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 Hession 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：

- 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 Hession 矩阵；
- 计算 Hession 矩阵的两个特征值；
- 设定阈值大小，决定极值点的去和留。

2.3 确定关键点的主方向

对关键点的描述，一般就是用向量对关键点进行描述，也就是找到关键点的方向。以关键点为中心画一个圆形邻域区域进行采样，分为 4x4=16 个子域，然后通过直方图来统计每个子域内像素点的梯度方向，直方图的峰值代表的就是关键点的方向。最终得到每个子域 8 个方向上的梯度幅值和。所以这个关键点的整个邻域区域上就得到了一个 4x4x8=128 维的筛特征向量。在 SIFT 算法中得到的关键点具有缩放不变的特性，每个关键点的方向也具有特征不变性，同时通过像素点(x，y)的梯度表示：

$$gradI(x,y) = (\frac{\partial I}{\partial x}\frac{\partial I}{\partial y})$$

梯度幅值：

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

梯度方向：

$$\theta(x,y) = \tan^{-1}[\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}]$$

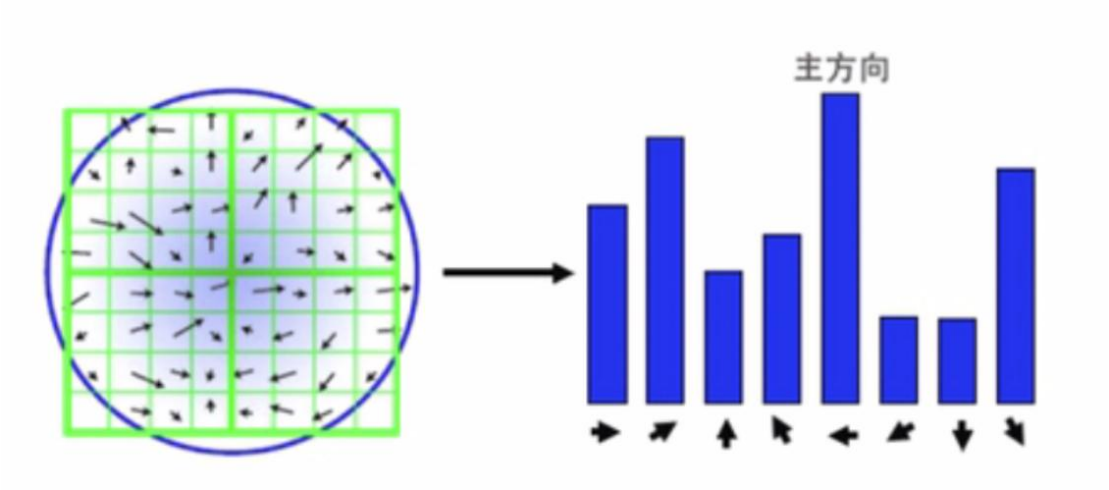


图 2.4 关键点主方向确定

检测完图像的关键点后，我们可以得到每个关键点的位置、缩尺度和方向三个重要向量信息，同时关键点也具备了特征不变性。为了提高关键点的稳定性，对关键点的这个描述子不仅包括关键点，还包括关键点周围有贡献的点，可以通过线性插值来计算关键点周围每个种子点的贡献。我们找到了以关键点为中心的 128 维向量，需要将这些向量进行归一化处理，这样就可以进一步去除光照变换对关键点匹配的影响。所以整个区域，即以每个关键点为中心得到了一个 128 维的向量描述符。

2.4 对关键点进行描述

在过滤掉对比度低的极值点和稳定性差的极值点后,又将剩余候选特征点的主方向进行确定，则此时的候选特征点被扶正成准确的特征点，用于后续的特征点匹配，而在此之前需要对特征点进行组合形成特征向量，即关键点描述。具体的实现步骤如下所述。

首先，将特征点邻域范围的像素点切割成多个 4x4 的像素矩阵，对单个像素矩阵绘制梯度直方图，且梯度直方图的方向总计设定为 8 个，单个方向的划分过程与特征点的主方向相同。假设最后生成的特征向量是-一个高维矢量 η，则 η 的任意一个分量代表梯度直方图的一个方向区间，则可以用一个 8 维矢量来唯一确定一个像素矩阵。本文中，选取特征点邻域 16x16 大小范围的像素点为实验对象，先将其进行分割，形成 16 个 4x4 的像素矩阵，相应地会得到一个 16x8 维的特征描述子。下图 2.5 模拟了特征向量的提取过程。

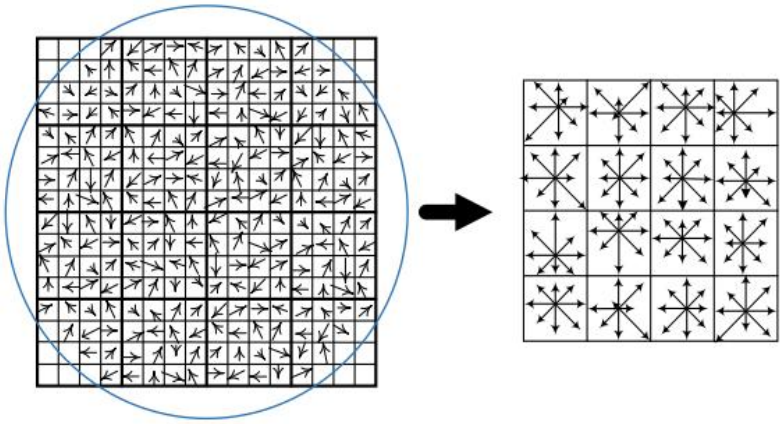


图 2.5 特征向量形成示意图

由上述过程可知，最终提取的特征向量η是 128 维的向量，表示成下式，

$$\eta = [\eta_1, \eta_2, \cdots, \eta_{128}]^T$$

为了进一步计算方便，将特征向量η做归一化处理，满足下式，

$$\begin{cases} \eta^{'} = [\eta_1^{'}, \eta_2^{'}, \cdots, \eta_{128}^{'}]^T \\ \eta_j^{'} = \dfrac{\eta_j}{\sqrt{\sum_{j=1}^{128} \eta_j^2}}, j = 1, 2, \cdots, 128 \end{cases}$$

综上所述，即可得到期望的特征点和特征向量。下面以流程图的形式对上述过程进行归纳演示，如图 2.6 所示。

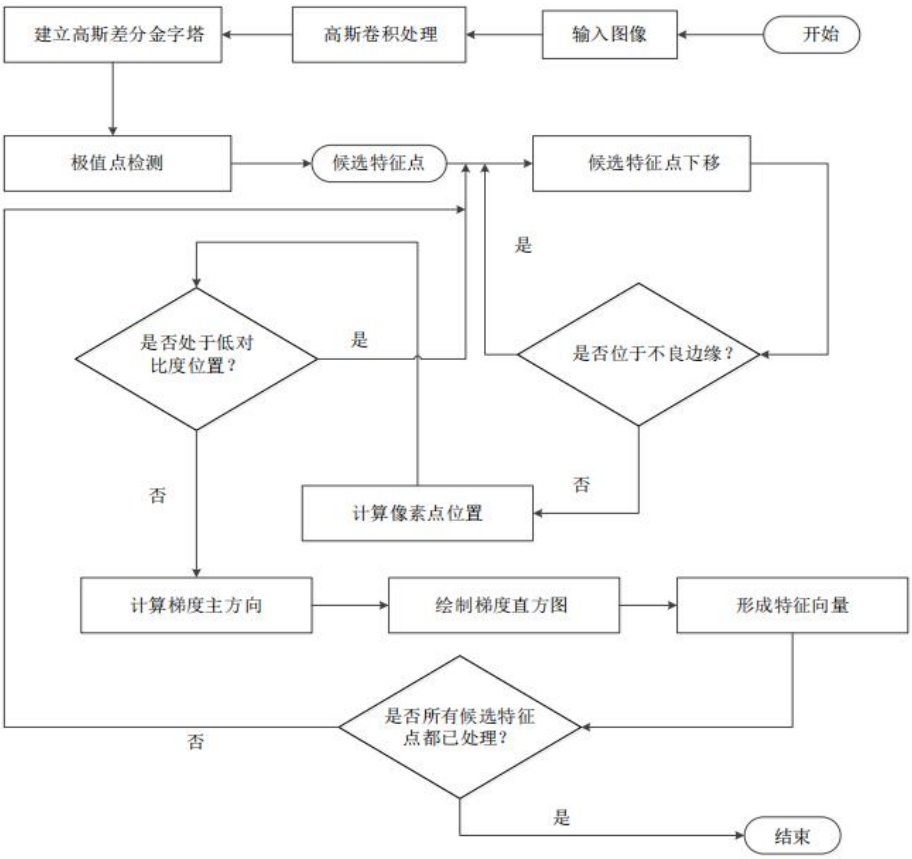


图 2.6 SIFT 原理流程图

2.5 关键点匹配

2.5.1 k-d 树算法

在 SIFT 算法中利用 k-维树 k-d（dimensional）树进行搜索匹配关键点。k-d 树算法由 2 部分组成，一是建立 k-d 树数据结构，二是在 k-d 树这种数据结构上进行数据查找的算法。

- **构建 k-d 树**

k-d 树是一个二叉树的数据结构。k-d 树上每一个节点代表着一个空间范围，通过分割平面将整个空间分割成左右 2 个子空间。1）确定分割域，对于所有描述子数据（特征向量），统计它们在每个维上的数据方差，选取出最大值，而对应的维数 k 即为分割域的值；2）确定节点数据域，数据点集按照前面的分割域的值进行排序，位于中间的那个数据点即为节点数据；3）确定左子空间和右子空间，分割平面（k 等于节点数据值）将整个空间分割成 2 个子间。k-d 树的构建是一个递归的过程，对左子空间和右子空间内的数据重复根节点的过程就可以得到下一级子节点，如此反复到空间只包含一个数据点。构建 k-d 树的流程如下图 2.7 所示。

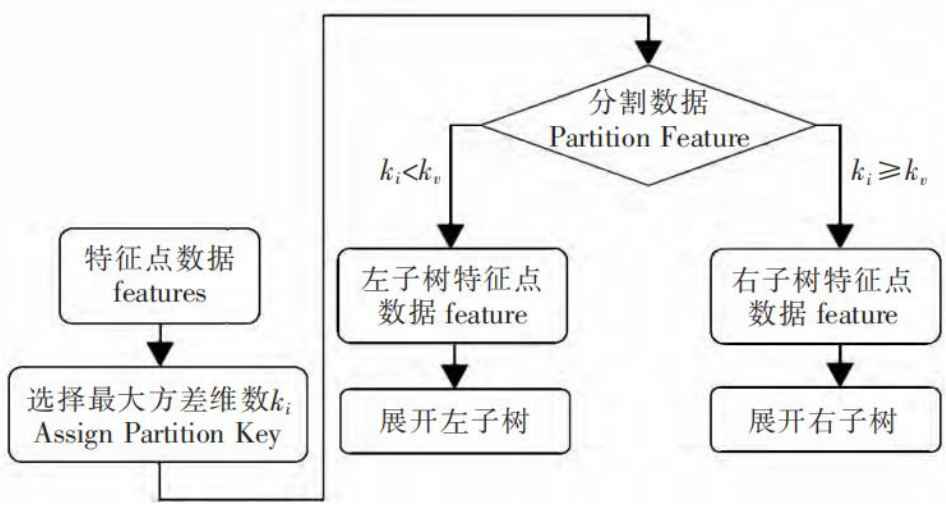


图 2.7 k-d 树构建流程

- **k-d 树最近邻查找算法**

首先，通过二叉树搜索（比较待查询节点和 k-d 树节点最大方差维数的值，小于时进入左子树分支，大于等于时则进入右子树分支直到叶子结点），进行二叉树查找后可以得到最近邻的相似点，即在二叉树的叶子节点。然后,这个叶子节点返回到父节点查找是否有距离查询节点更近的节点,如果没有则说明该子节点就是最近邻点，否则需要跳到其他子结点空间中去搜索（将其他子结点加入到搜索路径）。重复该过程直至搜索路径为空。

2.5.2 RANSAC 算法

随机抽样一致性 RANSAC (Random Sample Consensus)算法是一种鲁棒的变换估计算法，利用特征集合的内在集合约束关系进一步剔除错误的匹配，提高匹配正确率。该算法的主要思想是先随机选取 2 个点来确定一条直线，将这条直线一定距离范围内的点称为这条直线的支撑。随机选择重复多次，具有最大支撑特征集的直线被确认为是样本点集的拟合。在拟合的误差距离范围内的点称为内点，反之则为外点。将外点剔除掉，增强图像配准算法的稳健型。利用这种方法能减少误匹配的点，使配准效果更佳，提高了正确率。

RANSAC 算法假设给定一组数据，存在可以计算出符合这些数据的模型参数的方法。估计模型参数的过程如下：

步骤 1 从一个有 N 个数据的集合 P 中随机抽取模型求解要求最少的 n 个数据，根据这些抽取的 h 个数据计算出一个估计模型 M;

步骤 2 然后对其余的 N-n 个数据，计算它们与估计模型 M 之间的距离，保存这个集合中在估计模型 M 的误差允许范围内的数据个数 c;

步骤 3 步骤 1 和步骤不断迭代 k 次，对应最大 c 值的估计模型即为所求的模型，这 c 个数据即为内点;

实际应用中需要确定一个适当的迭代次数 k，迭代次数是保证模型估计需要的 n 个数据都是内点的概率 p 所需要的迭代次数，迭代次数 k 需要满足的条件为

$$1-p=(1-w^n)^k \Rightarrow k=\frac{log(1-p)}{log(1-w^n)}$$

式中：w 为该数据是数学模型内点的概率；n 为确定模型参数的最少点数。

2.6 图像拼接

图像拼接也称为图像融合算法，根据融合策略的不同可分为两大类：第一种方法是对重叠区域进行简单的拼接，一般有加权平均法、像素加权融合法等；第二种方法是找到图像的最佳分割线如图像分割法等。第一种图像融合法可以有效地消除拼接处的拼接缝，但是无法消除重影等现象，第二种图像融合技术耗时会相对长，需要花费时间代价。

图像加权平均法是最简单、直接的图像融合方法，其公式如下所示，其中，它具有简单易实现、运算速度快的优点，并能提高融合图像的信噪比。

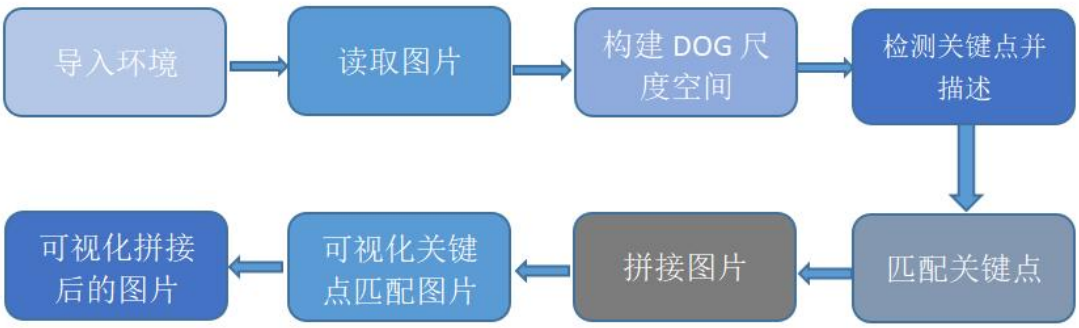
$$f(x,y) = \omega_1 f_1(x,y) + \omega_2 f_2(x,y)$$

简单的加权平均法是通过预设 ω_1 和 ω_2 的值且 ω_1和 ω_2 的值是固定不变的，且满足条件ω_1+ω_2= 1。

像素加权融合法也叫做像素加权平均法(Weighted Averaging, WA)，它是对加权平均法的一种改进方法。其表达形式也如上述公式所示，且 ω_1 和 ω_2的值也需满足条件 $\omega_1 + \omega_2 = 1$，不同的是 ω_1 和 ω_2的值是固定不变的，而是根据像素与两个图像重叠区域边界间的距离来决定的，其取值变换可以通过用如下表达式表示，左边 fi 为图像 1 的像素，右边 f 为图像 2 的像素，中间 f∩f2 为两个图像的重叠区域的像素变换。

$$\begin{cases} f(x,y) = f_1(x,y) & (x,y) \in f_1 \\ f(x,y) = \omega_1 f_1(x,y) + \omega_2 f_2(x,y) & (x,y) \in f_1 \cap f_2 \\ f(x,y) = f_2(x,y) & (x,y) \in f_2 \end{cases}$$

三、实验步骤



四、数据集

如下图所示，数据是两张图片，图(a)是国平图书馆南门中间和左边区域，图(b)是国平图书馆南门中间和右边区域。其中，为了更好的体现 SIFT 算法的鲁棒性，图(a)的拍摄角度和图(b)略有偏差，具体体现为图(a)拍摄角度更接近水平，图(b)有一点向上的角度；图(a)在图片中的位置偏高，而图(b)偏低。

(a) (b)

五、程序代码

- 首先，导入所需第三方包 numpy 和 cv2，并将收集的两张图片依次读入。

```
import numpy as np
import cv2
```

```
img1 = cv2.imread('image2.jpg')
img2 = cv2.imread('image1.jpg')
```

- 接着，将导入的图片转换为灰度图，然后建立 SIFT 生成器，并检测关键点、确定关键点主方向、对关键点进行描述。

```
def detectAndDescribe(self, image):
    # 转换为灰度图
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 建立SIFT生成器
    descriptor = cv2.SIFT_create()
    # 检测特征点并计算描述子
    kps, features = descriptor.detectAndCompute(gray, None)

    kps = np.float32([kp.pt for kp in kps])

    return kps, features
```

- 下面是匹配检测到的关键点。先建立暴力匹配器，然后试用 KNN 检测来自两

张图片的 SIFT 关键点匹配。匹配过后需要过滤掉误匹配的关键点。

```
def matchKeypoints(self, kpsA, kpsB, featureA, featureB, ratio, reprojThresh):
    # 建立暴力匹配器
    matcher = cv2.BFMatcher()

    # 使用KNN检测来自AB图的SIFT特征匹配
    rawMatches = matcher.knnMatch(featureA, featureB, 2)

    # 过滤
    matches = []
    for m in rawMatches:
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            matches.append((m[0].trainIdx, m[0].queryIdx))

    if len(matches) > 4:
        # 获取匹配对的点坐标
        ptsA = np.float32([kpsA[i] for (_, i) in matches])
        ptsB = np.float32([kpsB[i] for (i, _) in matches])

        # 计算H矩阵
        H, status = cv2.findHomography(ptsB, ptsA, cv2.RANSAC, reprojThresh)

        return matches, H, status
```

- 将两张图片拼接的拼接函数。

```
# 拼接函数
def stitch(self, images, ratio = 0.75, reprojThresh = 4.0, showMatches = True):
    # 读取图像
    imageB, imageA = images
    # 计算特征点和特征向量
    kpsA, featureA = self.detectAndDescribe(imageA)
    kpsB, featureB = self.detectAndDescribe(imageB)

    # 匹配两张图片的特征点
    M = self.matchKeypoints(kpsA, kpsB, featureA, featureB, ratio, reprojThresh)

    # 没有匹配点，退出
    if not M:
        return None

    matches, H, status = M
    # 将图片B进行视角变换 中间结果
    result = cv2.warpPerspective\
        (imageB, H, (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
    # cv2.imshow('h',result)
    # 将图片B传入]
    result[0:imageA.shape[0], 0:imageA.shape[1]] = imageA

    # 检测是否需要显示图片匹配
    if showMatches:
        # 生成匹配图片
        vis = self.drawMatches(imageA, imageB, kpsA, kpsB, matches, status)
        # 返回结果
        return result, vis

    # 返回匹配结果
    return result
```

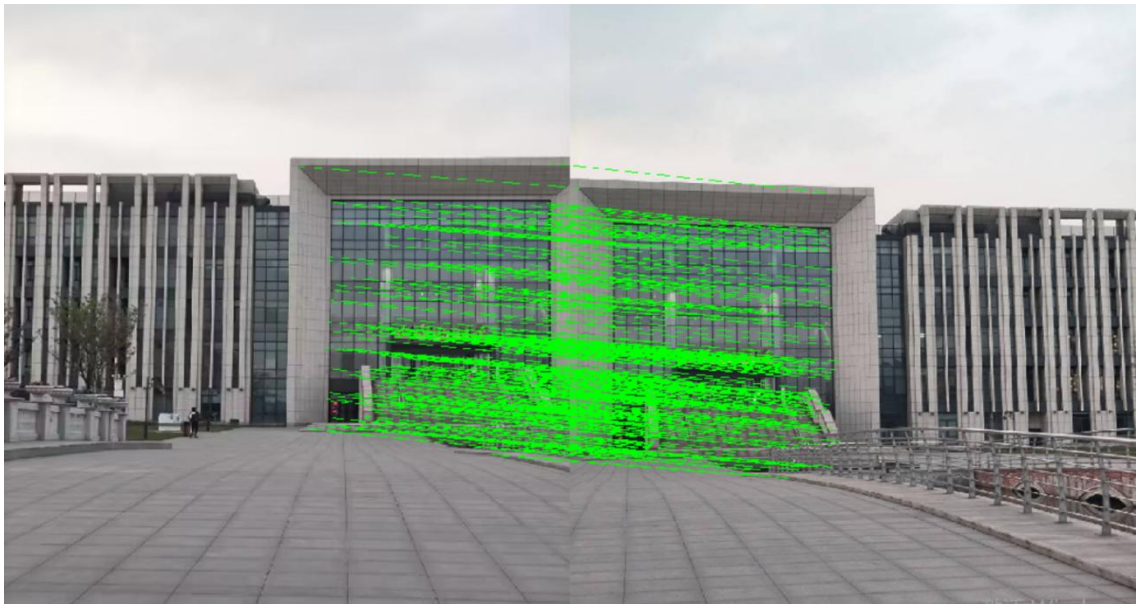
- 画出匹配了关键点的图片并将图片拼接起来。

```
def drawMatches(self, imageA, imageB, kpsA, kpsB, matches, status):
    # 初始化可视化图片，将A、B图左右连接到一起
    hA, wA = imageA.shape[:2]
    hB, wB = imageB.shape[:2]
    vis = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")
    vis[0:hA, 0:wA] = imageA
    vis[0:hB, wA:(wA+wB)] = imageB

    # 联合遍历，画出匹配对
    for ((trainIdx, queryIdx), s) in zip(matches, status):
        # 当点对匹配成功时，画到可视化图上
        if s == 1:
            # 画出匹配对
            ptA = (int(kpsA[queryIdx][0]), int(kpsA[queryIdx][1]))
            ptB = (int(kpsB[trainIdx][0]) + wA, int(kpsB[trainIdx][1]))
            cv2.line(vis, ptA, ptB, (0, 255, 0), 1)

    # 返回可视化结果
    return vis
```

- 最后，根据构建的 stitcher 类，画出匹配关键点的图片和最后的拼接完成的结果图片。

```
# 图片拼接
stitcher = Stitcher()
result, vis = stitcher.stitch([img1, img2], showMatches=True)
#画出关键点匹配图片
cv2.namedWindow('keypoints matches',0)
cv2.resizeWindow('keypoints matches',(img1.shape[1]+img2.shape[1],img1.shape[0]))
cv2.imshow('keypoints matches',vis)
#画出最终拼接完成的图片
cv2.namedWindow('result',0)
cv2.resizeWindow('result',(img1.shape[1]+img2.shape[1],img1.shape[0]))
cv2.imshow('result', result)
cv2.waitKey(0)
```

六、实验结果



如上图所示，展示的结果为匹配了关键点的拼接图片和最后的拼接图片。从图中可以看出，该拼接程序在拼接图片上取得了很好的结果。

七、实验分析与总结

实验过程中，首先需要准备两张需要拼接的图片。为了保证拼接效果，需要确保两张图片之间存在足够的重叠区域，并且保持相似的比例和角度。接着，使用 SIFT 算法提取图片中的局部特征点，并对这些特征点进行描述符的计算和匹配。

SIFT 算法的核心是局部特征点的提取和描述符的计算。在提取局部特征点时，SIFT 算法首先对图像进行高斯滤波，然后使用尺度空间极值检测来检测特

征点。在描述符的计算中，SIFT 算法使用梯度直方图来描述特征点周围的梯度方向和大小。通过这些描述符的计算，可以将图片中的特征点进行匹配，并计算出两张图片之间的单应性矩阵。

在单应性矩阵的计算中，使用 RANSAC 算法来剔除不良的匹配点。RANSAC 算法是一种基于随机采样和最小二乘法的算法，能够有效地去除不良的匹配点，并提高拼接的准确性和鲁棒性。通过单应性矩阵的估计和匹配点的筛选，最终可以将两张图片拼接在一起。

总的来说，利用 SIFT 算法进行图片拼接的实验取得了良好的效果。通过 SIFT 算法提取的局部特征点，我成功地找到了两张图片中相似的区域，并估计出了两张图片之间的单应性矩阵。最终，我利用单应性矩阵将两张图片拼接在一起，形成了一个更加完整的场景。

在实验过程中，我发现 SIFT 算法具有很强的鲁棒性和准确性，能够在不同光照、旋转和缩放条件下提取出相似的局部特征。同时，RANSAC 算法也能够有效地去除不良的匹配点，提高了拼接的准确性和鲁棒性。在实际应用中，这些算法和技术可以用于全景拍摄、卫星图像处理、医学图像配准等领域，具有广泛的应用前景。

当然，SIFT 算法也存在一些局限性，如计算速度较慢、需要大量的存储空间等问题。同时，当拼接的图片中存在较大的视角变化时，可能会出现拼接不完整或者拼接不准确的情况。因此，在实际应用中需要根据具体情况选择合适的算法和技术，进行针对性的优化和改进。

总之，利用 SIFT 算法进行图片拼接的实验为我们提供了一个很好的学习机会。通过实验，我深入了解了 SIFT 算法及其在图像拼接中的应用，同时也感受到了计算机视觉领域的魅力和挑战。相信在不久的将来，随着技术的不断发展和进步，图像拼接技术将会越来越成熟和完善，为人们的生活和工作带来更多的便利和实用价值。