
基于 SIFT 的图像拼接

姓名：陈浩 学号：123106222839 学院：计算机科学与工程学院

1 实验目标

- 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
- 使用 python 实现图像拼接算法

2 实现说明

2.1 方法原理

关键点检测算法 DOG。DOG (Difference of Gaussians) 主要基于图像中不同尺度下的高斯滤波，通过在不同尺度下对图像进行高斯平滑和差分操作，从而在不同尺度和位置上检测出图像中的关键点，该方法通常分为以下几个步骤。

①尺度空间构建，首先利用不同尺度下的高斯函数对原始图像进行多次平滑处理，得到一系列尺度空间的图像。

②高斯差分，对于每个尺度空间中的图像，利用相邻尺度的高斯平滑版本进行差分操作，得到一组高斯差分图像。

③关键点检测，通常通过对像素值和相邻像素值进行比较来确定是否为极值点，并且对比周围像素进行非极大值抑制，以精确地确定关键点的位置和尺度。

④关键点定位：在检测到的极值点中，通过一些特定的筛选规则（如梯度、对比度和曲率等）来确定最终的关键点位置和尺度。

⑤关键点描述：对于每个检测到的关键点，通常会利用其周围像素的信息来生成一个具有描述性的特征向量，常用的方法包括 SIFT、SURF、ORB 等。

尺度变化不变特征 SIFT 通过在不同尺度空间中检测关键点，并利用关键点周围的局部图像信息生成描述子，实现了图像特征的尺度不变性和一定程度的旋转不变性。SIFT 算法的核心步骤如下。

①方向赋值，在实现关键点检测和定位后，对于每个关键点 SIFT 进行方向赋值，即确定关键点的主导方向。这一步是为了提高关键点的旋转不变性，使得关键点描述子在旋转变换下具有稳定性。通常采用图像局部梯度信息来确定关键点的主导方向。

②关键点描述，SIFT 利用关键点周围的局部图像区域来生成描述子，通常采用的方法是在关键点周围的图像区域中构建特征向量，该向量能够描述关键点周围的图像结构和纹理特征。这些描述子具有一定的局部不变性，能够在一定程度上抵抗图像的缩放、旋转和亮度变化等影响。

③匹配与验证，最后利用关键点的描述子进行特征匹配，通常采用的方法是计算描述子之间的相似性度量，如欧氏距离或余弦相似度。

2.2 实验流程

①检测关键点和计算关键点描述，先将两图片转换为灰度图，并建立 SIFT 生成器，然后检测特征点并计算描述子。

②匹配两张图片的特征点，我采用 KNN 检测来自两图的 SIFT 特征匹配，过滤掉不合格匹配对，完成满足条件的匹配对点坐标的提取。

③对右图进行透视变换并拼接左图。

④实现拼缝混合渐变。

2.3 代码实现

2.3.1 检测关键点和计算关键点描述

由函数 `detectAndDescribe(image)` 实现，具体流程如下：

①使用 `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` 将输入的彩色图像转换为灰度图像，以便进行后续的特征检测和描述子计算。

②使用 `cv2.SIFT_create()` 创建一个 SIFT 特征检测器，方便使用 SIFT 算法。

③使用 `descriptor.detectAndCompute(gray, None)` 对灰度图像进行特征点检测和描述子计算，得到关键点列表 `kps` 和对应的特征描述子 `features`。

④将关键点列表中的关键点坐标转换为 `numpy` 数组格式，以便后续的处理。

2.3.2 匹配两张图片的特征点

由函数 `matchKeypoints(kpsA, kpsB, featureA, featureB, ratio, reprojThresh)` 实现，具体流程如下：

①创建暴力匹配器对象 `matcher = cv2.BFMatcher()`，用于在特征空间中寻找最接近的匹配点。

②使用 K 最近邻 (KNN) 方法对两图像的 SIFT 特征进行匹配，具体为 `matcher.knnMatch(featureA, featureB, 2)`。这里的 `featureA` 和 `featureB` 分别是两张图像的特征描述子，而 2 表示每个特征点要找到最接近的两个匹配点。

③对原始匹配结果进行筛选，保留满足一定比例关系的匹配对。具体操作为遍历 `rawMatches` 中的每对匹配点，然后判断是否满足以下条件：

`len(m) == 2`：确保每个特征点都有两个最接近的匹配点。

`m[0].distance < m[1].distance * ratio`：第一个最近邻匹配点的距离小于第二个最近邻匹配点的距离乘以 `ratio`。如果满足这个条件，则认为这对匹配是合格的，应该保留下来。

④如果满足条件的匹配对数量大于 4，则继续进行下一步处理；否则返回空值。对于满足条件的匹配对，提取它们的点坐标。

⑤利用 RANSAC 算法估计单应性矩阵 `H`，以及状态 `status`。这里使用 `cv2.findHomography(ptsB, ptsA, cv2.RANSAC, reprojThresh)` 进行计算。

⑥最后将经过筛选后的匹配对 `matches`、单应性矩阵 `H` 和状态 `status` 返回。

2.3.3 对右图进行透视变换并拼接左图

`cv2.warpPerspective(imageB, H, (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))`：这一行代码对图像 B 进行透视变换，`H` 是透视变换矩阵，指定了变换的方式。变换后的图像大小是 `(imageA.shape[1] + imageB.shape[1], imageA.shape[0])`，即图像 A 和图像 B 横向拼接后的大小。

`cv2.cvtColor(result, cv2.COLOR_BGR2RGB)`：这一行代码将处理后的图像转

换为 RGB 格式，因为 `plt.imshow()` 函数需要 RGB 格式的图像来显示。

`result[0:imageA.shape[0], 0:imageA.shape[1]] = imageA`: 这一行代码将图像 A 插入到透视变换后的图像的左上角，形成了拼接后的图像。

2.3.4 混合拼接

`mask = np.zeros((imageA.shape[0], imageA.shape[1]), dtype=np.uint8)`: 首先创建了一个与图像 A 大小相同的 mask，数据类型为 `uint8`。

`mask[:, imageA.shape[1] - 40:] = 255`: 然后将 mask 的右侧 40 个像素列设置为白色（像素值为 255），这个操作会在图像 A 和图像 B 的接缝处创建一个渐变混合区域。

`blend=cv2.seamlessClone(imageA,result,mask,(imageA.shape[1],imageA.shape[0] // 2), cv2.NORMAL_CLONE)`: 使用 `cv2.seamlessClone()` 函数将图像 A 无缝融合到之前拼接得到的 `result` 图像中，融合时使用了之前创建的 mask，并指定了融合的位置和方式。

3 实验结果

3.1 运行说明

在 `imgStitch.ipynb` 文件中，运行方法如下：

①顺序执行步骤 1-4，会详细地展示每一步骤结果。

②完整拼接函数在步骤 5，仅展示匹配结果（`matches.png`）、直接拼接结果（`direct.png`）和混合拼接结果（`blend.png`）。

③输入图像存放于 `img` 文件夹，结果输出在与 `img` 同级文件夹下。

3.2 结果与说明

3.2.1 关键点匹配



图中的绿线连接了两张图片的匹配的关键点。

3.2.2 直接拼接



可以看出直接拼接结果的接缝明显。

3.2.2 混合拼接



可以看出两张图片相接部分有一定的相融，通过渐变融合弱化了接缝。