基于 SIFT 的图片拼接方法

《计算机视觉》2023 秋

魏少杭
学号：20373594
人工智能研究院
完成日期：2023 年 12 月

目录

备注：本文中所有代码均来自于 *sift_match.py*。

1 项目背景简介

1.1 任务简介

2023 年秋季《计算机视觉》第一项大作业为图片拼接任务。给定两张原始的图片图 1a和1b，需要将这两张图片通过特征匹配方法拼接起来。

图片拼接任务可以通过多种计算机视觉与图像处理技术来解决，其中最常用、简单的图像处理技术之一是 SIFT（Scale-invariant Feature Transform）。

(a) 左图　　(b) 右图

图 1: 原始图

1.2 SIFT 原理描述

SIFT 全称为尺度不变特征变换，拥有尺度不变性、角度旋转不变性、光照不变性以及视角变化、仿射变换、噪声稳定性等优良性质。其具体应用包括了影像缝合、手势辨别、影像追踪、动作比对等。

SIFT 算法总体本质是在不同尺度空间上找关键点，并计算出关键点的方向特征。SIFT 找到的关键点所含特征十分突出，如角点、边缘点等，这些关键点的稳定性保证了后续图像特征匹配和融合。

首先，为了实现尺度不变性，我们利用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。具体分为构造高斯空间、构造高斯差分空间，最后进行局部极值选择。

其次，由于极值选择得到的结果是离散空间的极值点而非精确的极值点，为了精确定位关键点，可以首先进行子像素插值，然后进行对比度检测、消除边缘响应。以上步骤能够提升图像特征点对噪声的鲁棒性。

第三，为了使得图像具有旋转不变性，我们需要先确定出关键点的主方向。在确定主方向的过程中需要对关键点周围的邻域计算梯度方向和模值，并作出关键点周围邻域梯度方向直方图，选择出主方向和辅方向。

第四，为了得到具有旋转不变性和亮度变化不变性的旋转矢量，我们需要首先对图像中所有匹配特征点的邻域点校正旋转主方向，使得梯度方向统一；然后生成描述子，得到 128 维特征向量；最后归一化处理特征向量，以去除光照影响。

2 核心步骤

2.1 SIFT 特征检测、提取

本文使用了 opencv-python 库中的函数实现了 SIFT 特征描述子对象创建，并使用该特征提取器，提取特征关键点位置和特征向量。

```
1
2       # 创建SIFT特征提取对象
3       sift = cv2.SIFT_create()
4
5       # 获取关键点坐标、特征描述
6       keyPointer_left, descriptor_left = detect_sift(sift,
            image_left)
7       keyPointer_right, descriptor_right = detect_sift(sift,
            image_right)
8
9   # 通过cv2封装好的SIFT特征提取器，提取特征关键点位置和特征向量
10  def detect_sift(sift, img):
11      # gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)    # 转为
            灰度图
12      keyPointer, descriptor = sift.detectAndCompute(img, None)
13      keyPointer = np.float32([kp.pt for kp in keyPointer])
14      return keyPointer, descriptor
```

2.2 特征点匹配

```
1       # 特征点匹配
2       (M, matches, status) = match(keyPointer_right,
            keyPointer_left, descriptor_right, descriptor_left,  args.
            k, args.ratio, args.reprojThresh)
3
4   # 获得匹配出来的点坐标、坐标变换矩阵等
5   def match(kps1, kps2, des1, des2, k=2, ratio=0.75, reprojThresh
        =4.0):
6       bf = cv2.BFMatcher() # crossCheck为True时匹配条件更严格
7       raw_matches = bf.knnMatch(des1, des2, k)
8       matches = []
9       for m in raw_matches:
10          # 找到符合最近邻匹配要求的所有匹配对，并把他们的索引分别
                提取出来
11          # 遍历raw_matches，如果第一对点的欧氏距离小于ratio倍第二
                对点的欧氏距离则说明第一对点匹配可靠，将这对点的索引号
                追加到matches
```

```
12          if len(m) == 2 and m[0].distance < ratio * m[1].distance:
13              matches.append((m[0].queryIdx, m[0].trainIdx))
14      kps1 = np.float32([kps1[m[0]] for m in matches])     # 提取出
            对应点的keypointer（图片坐标）
15      kps2 = np.float32([kps2[m[1]] for m in matches])     # 同上
16      # 求解转换矩阵
17      # 求解方法：cv2.RANSAC
18      # reprojThresh是将点对视为内点的最大允许重投影错误阈值（仅用
            于RANSAC和RHO方法）
19      # 返回的M为变换矩阵
20      # status能够指示是否该点有匹配解
21      (M, status) = cv2.findHomography(kps1, kps2, cv2.RANSAC,
            reprojThresh)
22      return (M, matches, status)
```

2.3 拼接图片

```
1       def stich(img1, img2, M):
2       result = cv2.warpPerspective(img1, M, (img1.shape[1] + img2.
            shape[1], img1.shape[0]))
3       result[0: img1.shape[0], 0: img2.shape[1]] = img2
4       return result
```

3 效果展示

拼接后的结果为 stiched_picture.png，如下图 2所示：

可以从这张图看到，通过 SIFT 特征匹配并拼接后，这两张图基本一致匹配地拼接成功。

在 SIFT 之外，还可以使用改进版的 SURF 算法，来改进 SIFT 计算速度。

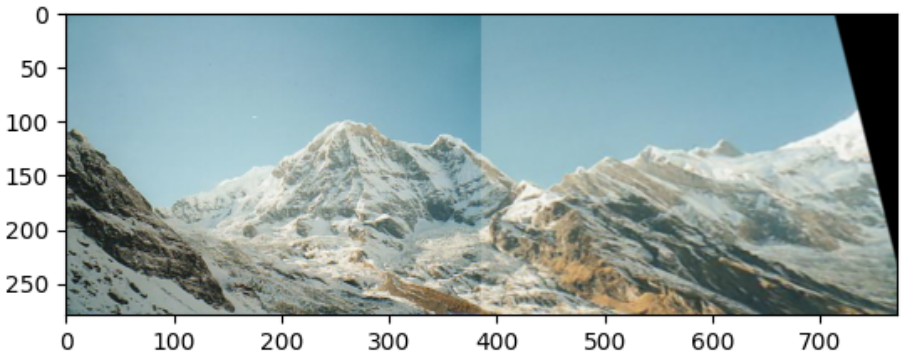

图 2: 拼接后的图片