华东师范大学计算机科学技术系上机实践报告

课程名称：计算机视觉	**年级**：2015	**上机实践成绩**：
指导教师：文颖	**姓名**：朱勇赤	**上机实践日期**：2018/6/19
实践编号：实验4特征匹配的图像拼接	**学号**：10152130131	**上机实践时间**：10:00~11:40

一、 实验名称

特征匹配的图像拼接

二、 实验目的

输入一系列待拼接图像，输出拼接好的图像

三、 实验内容

尝试采用SIFT特征描述子提取特征；

尝试特征匹配；

找到变换矩阵，变换图像

拼接融合图像

四、 实验原理

一幅图像中总存在着其独特的像素点，这些点我们可以认为就是这幅图像的特征，成为特征点。计算机视觉领域中的很重要的图像特征匹配就是一特征点为基础而进行的，所以，如何定义和找出一幅图像中的特征点就非常重要。 从图像中选取某些特征点并对图像进行局部分析，而非观察整幅图像。 只要图像中有足够多可检测的兴趣点，并且这些兴趣点各不相同且特征稳定， 能被精确地定位，从而对分割的图像进行拼接。

尺度不变特征转换(Scale-invariant feature transform或SIFT)是一种电脑视觉的算法用来侦测与描述影像中的局部性特征，它在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量，此算法由 David Lowe在1999年所发表，2004年完善总结。

其应用范围包含物体辨识、机器人地图感知与导航、影像缝合、3D模型建立、手势辨识、影像追踪和动作比对。

局部影像特征的描述与侦测可以帮助辨识物体，SIFT 特征是基于物体上的一些局部外观的兴趣点而与影像的大小和旋转无关。对于光线、噪声、些微视角改变的容忍度也相当高。基于这些特性，它们是高度显著而且相对容易撷取，在母数庞大的特征数据库中，很容易辨识物体而且鲜有误认。使用 SIFT特征描述对于部分物体遮蔽的侦测率也相当高，甚至只需要3个以上的SIFT物体特征就足以计算出位置与方位。在现今的电脑硬件速度下和小型的特征数据库条件下，辨识速度可接近即时运算。SIFT特征的信息量大，适合在海量数据库中快速准确匹配。

SIFT算法的特点有：

1. SIFT特征是图像的局部特征，其对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性；

2. 独特性（Distinctiveness）好，信息量丰富，适用于在海量特征数据库中进行快速、准确的匹配；

3. 多量性，即使少数的几个物体也可以产生大量的SIFT特征向量；

4. 高速性，经优化的SIFT匹配算法甚至可以达到实时的要求；

5. 可扩展性，可以很方便的与其他形式的特征向量进行联合。

SIFT算法可以解决的问题：

目标的自身状态、场景所处的环境和成像器材的成像特性等因素影响图像配准/目标识别跟踪的性能。而SIFT算法在一定程度上可解决：

1. 目标的旋转、缩放、平移（RST）

2. 图像仿射/投影变换（视点viewpoint）

3. 光照影响（illumination）

4. 目标遮挡（occlusion）

5. 杂物场景（clutter）

6. 噪声

SIFT算法的实质是在不同的尺度空间上查找关键点(特征点)，并计算出关键点的方向。SIFT所查找到的关键点是一些十分突出，不会因光照，仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

Lowe将SIFT算法分解为如下四步：

1. 尺度空间极值检测：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。

2. 关键点定位：在每个候选的位置上，通过一个拟合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。

3. 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。

4. 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。

特征检测的视觉不变性是一个非常重要的概念。 但是要解决尺度不变性问题，难度相当大。 为解决这一问题，计算机视觉界引入了尺度不变特征的概念。 它的理念是， 不仅在任何尺度下拍摄的物体都能检测到一致的关键点，而且每个被检测的特征点都对应一个尺度因子。 理想情况下，对于两幅图像中不同尺度的的同一个物体点， 计算得到的两个尺度因子之间的比率应该等于图像尺度的比率。其中SIFT（尺度不变特征转换， ScaleInvariant Feature Transform）因为其巨大的特征计算量而使得特征点提取的过程异常花费时间，所以在一些注重速度的场合难有应用场景。但是SIFT相对于SURF的优点就是，由于SIFT基于浮点内核计算特征点，因此通常认为， SIFT算法检测的特征在空间和尺度上定位更加精确，所以在要求匹配极度精准且不考虑匹配速度的场合可以考虑使用SIFT算法。

五、 实验算法

OPENCV下SIFT特征点提取与匹配的大致流程如下：

读取图片-》特征点检测（位置，角度，层）-》特征点描述的提取（16*8维的特征向量）-》匹配-》拼接-》显示

步骤如下：

1、使用opencv内置的库读取两幅图片

2、生成一个descriptor对象，这个对象顾名思义就是SIFT特征的探测器，用它来探测衣服图片中SIFT点的特征，存到一个KeyPoint类型的vector中。

实现如下：

```
# 使用 SIFT_create()方法实例化 DOG 关键点与 SIFT 特征点
descriptor = cv2.xfeatures2d.SIFT_create()
#将关键点与特征分离
(kps, features) = descriptor.detectAndCompute(image, None)
```

keypoint只是保存了opencv的sift库检测到的特征点的一些基本信息，但sift所提取出来的特征向量其实不是在这个里面，特征向量通过SiftDescriptorExtractor 提取，结果放在一个Mat的数据结构中。这个数据结构才真正保存了该特征点所对应的特征向量。

3、对图像所有KEYPOINT提取其特征向量：

得到keypoint只是达到了关键点的位置，方向等信息，并无该特征点的特征向量，要想提取得到特征向量就还要进行detectAndCompute 的工作，建立了对象后，通过该对象，对之前SIFT产生的特征点进行遍历，找到该特征点所对应的128维特征向量。通过这一步后，所有keypoint关键点的特征向量被保存到了一个MAT的数据结构中，作为特征。

实现如下：

```
#将关键点与特征分离
(kps, features) = descriptor.detectAndCompute(image, None)
```

4、对两幅图的特征向量进行匹配，得到匹配值。

两幅图片的特征向量被提取出来后，我们就可以使用BruteForce方法对两幅图片的descriptor进行匹配，得到匹配的结果到matches中，

实现如下：

```
# 使用 BruteForceMatcher 对象对两幅图片的 descriptor 进行匹配
matcher = cv2.DescriptorMatcher_create("BruteForce")
rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
```

至此，SIFT从特征点的探测到匹配都已经完成。

5、 筛选匹配点：

如果直接使用所有的匹配点无疑会工作量巨大且不一定准确，所以我们需要进一步筛选匹配点，来获取优秀的匹配点，这就是所谓的“去粗取精”。这里我们采用了Lowe’s算法来进一步获取优秀匹配点。

为了排除因为图像遮挡和背景混乱而产生的无匹配关系的关键点，SIFT的作者Lowe提出

了比较最近邻距离与次近邻距离的SIFT匹配方式：取一幅图像中的一个SIFT关键点，并找出其与另一幅图像中欧式距离最近的前两个关键点，在这两个关键点中，如果最近的距离除以次近的距离得到的比率ratio少于某个阈值T，则接受这一对匹配点。因为对于错误匹配，由于特征空间的高维性，相似的距离可能有大量其他的错误匹配，从而它的ratio值比较高。显然降低这个比例阈值T，SIFT匹配点数目会减少，但更加稳定，反之亦然。

Lowe推荐ratio的阈值为0.8，但作者对大量任意存在尺度、旋转和亮度变化的两幅图片进行匹配，结果表明ratio取值在0.4~0.6 之间最佳，小于0.4的很少有匹配点，大于0.6的则存在大量错误匹配点，所以建议ratio的取值原则如下:

ratio=0.4：对于准确度要求高的匹配；

ratio=0.6：对于匹配点数目要求比较多的匹配；

ratio=0.5：一般情况下。

代码实现：

```
matches = []
# 在原始匹配点间循环
for m in rawMatches:
    # 确保彼此之间的距离在比率内
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        matches.append((m[0].trainIdx, m[0].queryIdx))
# 单应性变换需要至少四对匹配
if len(matches) > 4:
    ptsA = np.float32([kpsA[i] for (_, i) in matches])
    ptsB = np.float32([kpsB[i] for (i, _) in matches])

    # 计算出匹配点之间的单应性变换以及状态
    (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,reprojThresh)
    return (matches, H, status)
```

6、 绘制匹配点连线图像

```
def drawMatches(self, imageA, imageB, kpsA, kpsB, matches, status):
    # 对连接图初始化
    (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:] = imageB
    # 在匹配点循环
    for ((trainIdx, queryIdx), s) in zip(matches, status):
        if s == 1:
            # 将两图间匹配点连线
            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)

    # 返回可视化连接图像
    return vis
```

7、 拼接图像

使用获得的M矩阵，调用warpPerspective将左右两幅图连接在一起（以左图为基准）

实现如下：

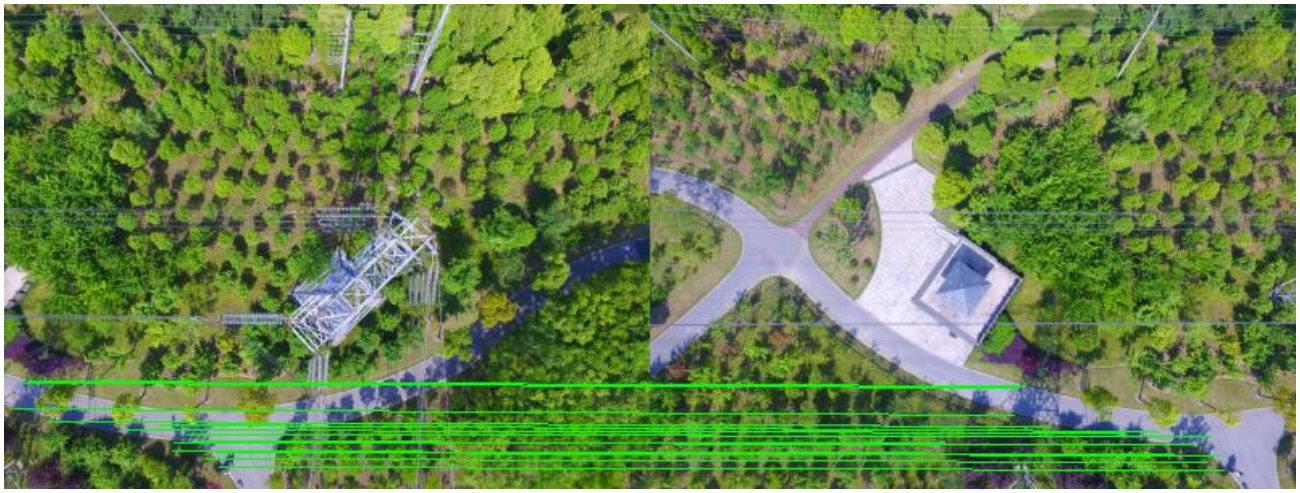
```
#将 M 矩阵划分为匹配点，单应性变换矩阵以及匹配点状态
(matches, H, status) = M
result = cv2.warpPerspective(imageA, H,
                             (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
result[0:imageB.shape[0], 0:imageB.shape[1]] = imageB
```

8、 拼接全景图像

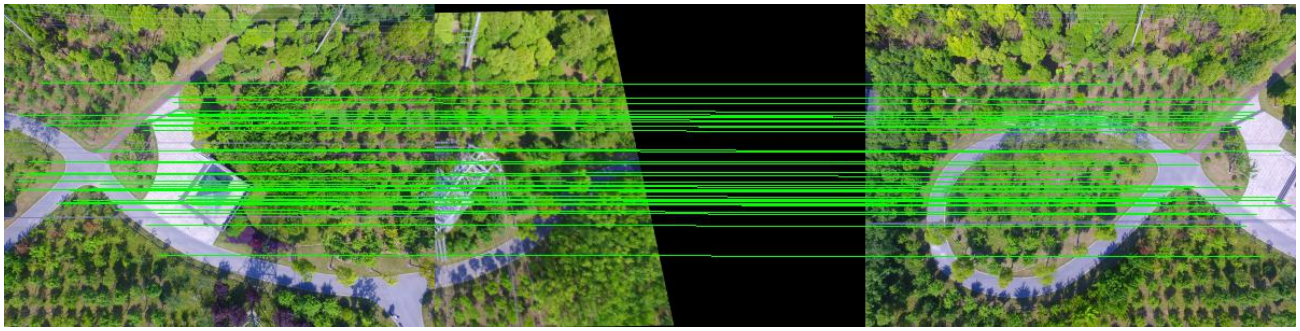
每次调用上方定义好的stitch()函数，将左部图像与右侧连接，总计五次，最终将六幅图像拼接为完整的全景图

六、 实验结果及分析

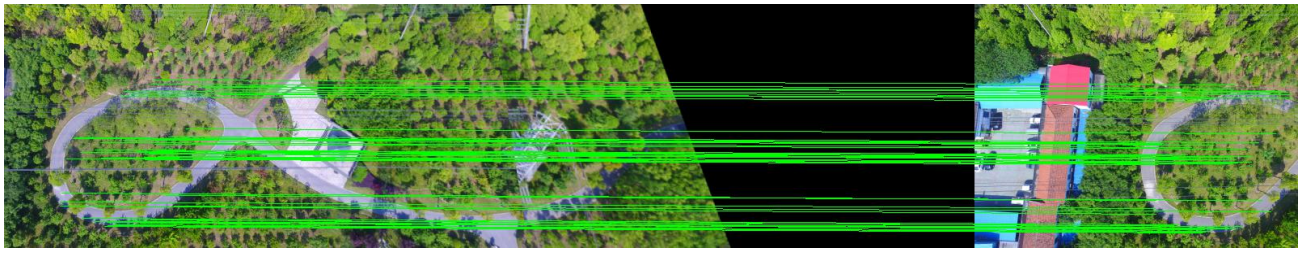
1、第五张与第六张的特征点匹配



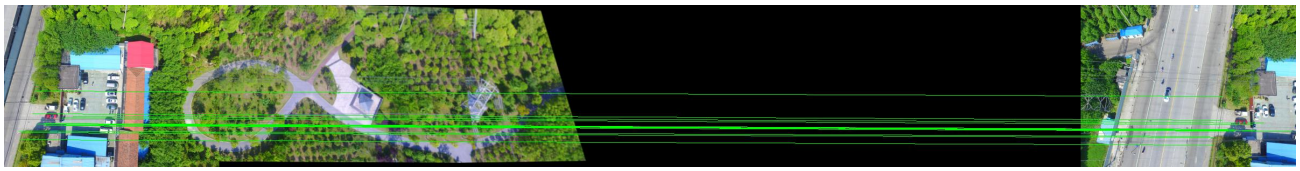
2、五六结合与第四张的特征点匹配



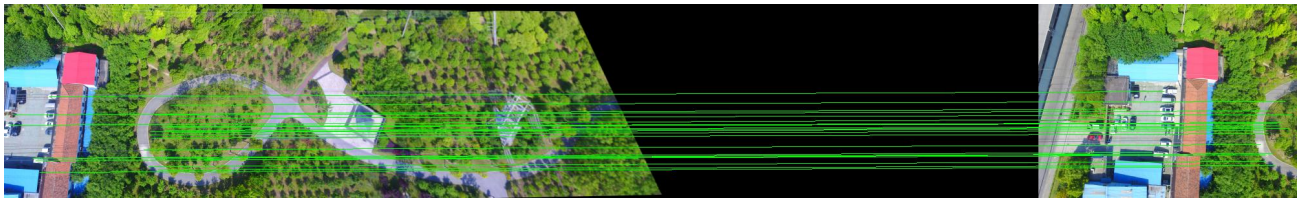
3、四五六结合与第三张的特征点匹配



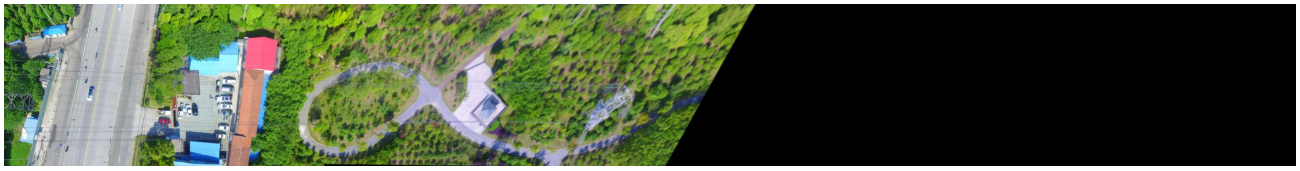
4、三四五六结合与第二张的特征点匹配



5、二三四五六结合与第一张的特征点匹配



6、将六幅图像进行拼接后的结果



可以看出六张图片都被连接在了一起，图片间没有明显的分割与错位，整体上也没有照片之间的独立感。

七、 问题讨论

连续的拼接需要右侧的图像不断被仿射变化来与左侧图像连接，而导致了最右侧的图像在最终的全景图中有些扭曲。

需要一定方法来只修正图片的右端，而不会对左侧造成影响。