南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

“计算机视觉与应用实践”

课程作业

姓 名：余 涛　　学 号：122106222833

专 业：计算机科学与工程

项目名称：Python 实现图像拼接算法

时间：2023 年 4 月 11 日

1、实验目标

（1）、理解关键点检测算法 DOG 原理。

（2）、理解尺度变化不变特征 SIFT。

（3）、采集一系列局部图像，自行设计拼接算法。

（4）、使用 Python 实现图像拼接算法。

2、具体要求

（1）、不允许使用现成的图像拼接程序 ；

（2）、在采图过程中可尽可能减少相机在垂直方向的运动，但不能假设图像只存在水平方向平移；

（3）、需包含图像融合部分，从而减少拼接图像中局部图像的“接缝”。

3、总体思路

（1）、首先读取两张图片，确保两张图片的宽度相同，这样才能拼接；

（2）、然后使 sift 对两种图片都做特征点检测，并对特征进行筛选，保留特征点相似程度高的；

（3）、根据上述得到的特征点，进行计算单应性矩阵 H（这个矩阵本质上是做位置上的变换的）；

（4）、根据矩阵 H 对读取的后一张图片,进行仿射变换(相当于位置变换)；

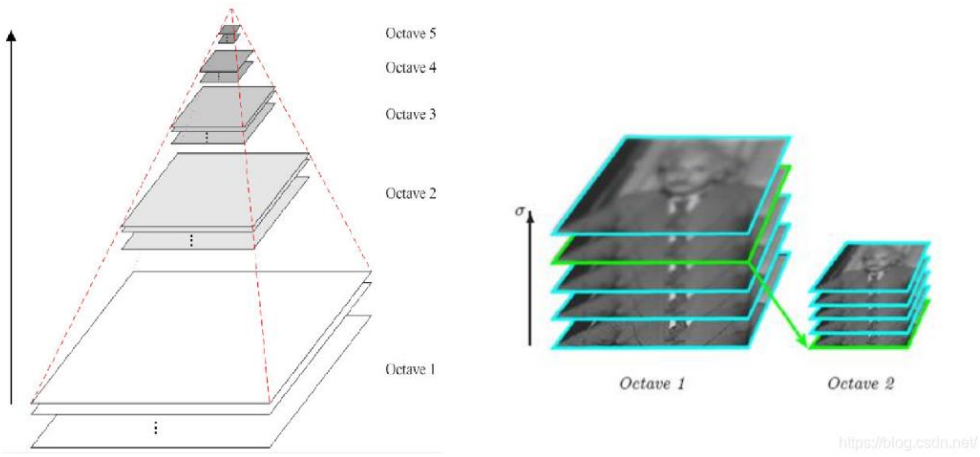
（5）、融合图片重叠部分，也就是把另一张图片拼接到合适的位置上。

4、项目实现

一、原理与实现

1、多分辨率高斯图像金字塔

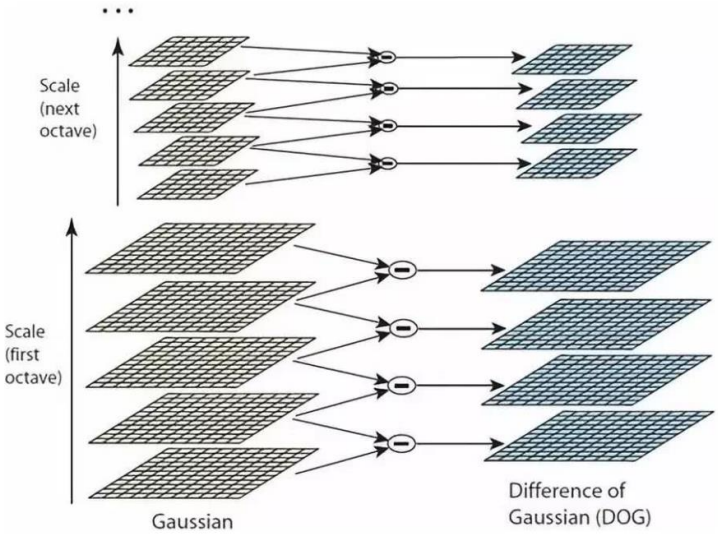
高斯图像金字塔构建过程中，首先将图像扩大一倍，在扩大的图像的基础之上构建高斯金字塔，然后对该尺寸下图像进行高斯模糊，几幅模糊之后的图像集合构成了一个 Octave，然后对该 Octave 下选择一幅图像进行下采样，长和宽分别缩短一倍，图像面积变为原来四分之一。这幅图像就是下一个 Octave 的初始图像，在初始图像的基础上完成属于这个 Octave 的高斯模糊处理，以此类推完成整个算法所需要的所有图像金字塔。



2、高斯差分图像金字塔（DOG）

高斯图像金字塔处理图像，让 6 个不同高斯模糊程度不同的图片都可以缩小成其他不同的尺寸，而高斯差分图像金字塔（DOG）就是将图像金字塔每一层的图片（6 张不同模糊程度，相同尺寸）进行差分运算。这些差分运算后得出了的特征图，不就是代表这两张图片的差异性结果吗？ 就跟上面举的例子一样。

跟下图一样，每一层的第一幅图片与第二幅图片进行差运算、第二幅与第三幅进行查运算。以此类推，逐组逐层生成每一个差分图像，所有差分图像构成差分金字塔。 概括为 DOG 金字塔的第 o 组第 1 层图像是有高斯金字塔的第 o 组第 1+1 层减第 o 组第 1 层得到的。后续 Sift 特征点的提取都是在 DOG 金字塔上进行的。

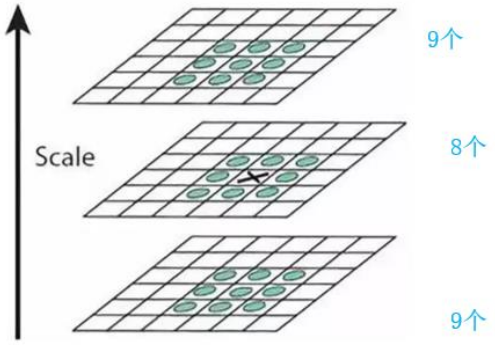


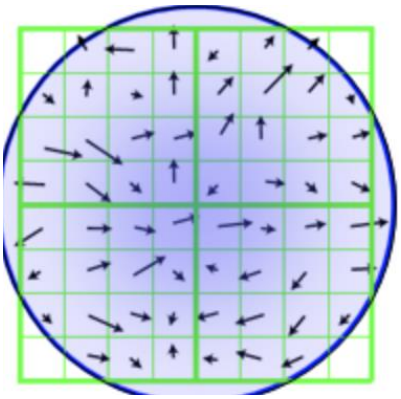
3、DOG 空间关键点检测

在 DoG 搞定之后，就可以在不同的尺度空间中搜索局部最大值了。对于图像中的一个像素点而言，每个像素点要和其图像域（同一尺度空间的 8 个点）和尺度域（相邻的尺度空间——上下各 9 个点，一共 18 个）的所有相邻点（一共 26 个）进行比较，当其大于或者小于所有相邻点时，就会被认为是关键点(局部极值点)。基本上来说关键点是图像在相应尺度空间中的最好代表。

如下图所示，中间的检测点要和其所在图像的 3 * 3 邻域 8 个像素点，以及

其相邻的上下两层 3 * 3，邻域 18 个像素点，共 26 个像素点进行比较。





总的来说，SIFT 在图像的不变特征提取方面拥有无与伦比的优势，但并不完美，仍然存在实时性不高，有时特征点较少，对边缘光滑的目标无法准确提取特征点等缺陷，自 SIFT 算法问世以来，人们就一直对其进行优化和改进，其中最著名的就是 SURF 算法。

二、具体步骤及过程

1. 首先需要找到图像的特征点，也就是使用 SIFT 进行特征点检测，得到特征点以及特征向量：

查阅资料发现，如果要实现图像之间的特征点匹配，要通过特征描述子集之间比对完成。在 DOG 金字塔中，通常用来找到确定的图像的特征点，而 SHIFT 也是运用了 DOG 金字塔。

常见的匹配器有暴力匹配器和快速近似最邻近算法匹配器。暴力匹配器就是将两幅图像中的特征描述子全都匹配一遍，得到最优的结果，它的优点是精度高，但是缺点也是显而易见的，在大量的匹配时，匹配时间会很长。快速近似最邻近算法匹配器，故名思意，它只搜索邻近的点，找到邻近的最优匹配，它的匹配准确度会比暴力匹配器低，但是它的匹配时间大大的缩减了。

```
def image_catdescribe(self, image):
        # 首先将图片转换为灰度图
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # 然后再建立一个 SIFT 生成器
        descriptor = cv2.xfeatures2d.SIFT_create()
        # SHIFT 会检测特征点并计算描述子，其中 kps 就是得到的特征点
        kps, features = descriptor.detectAndCompute(gray, None)
        kps = np.float32([kp.pt for kp in kps]) #这一步是逐个遍历算子，将其化成
flost32 浮点型
        return kps, features
```

其中特征点的检测和提取使用 SIFT：

```
cv2.xfeatures2d.SIFT_create(nfeatures=None,nOctaveLayers=None,contrastThreshold=None,
edgeThreshold=None, sigma=None)-->descriptor
#这个函数 SIFT 是用来创建一个用于提取 SIFT 特征的描述符，也就是初始化。
#nfeatures 可以保留的最佳特性的数量。特征按其得分进行排序(以 SIFT 算法作为局部对比度进行测量)；
# nOctavelLayers 高斯金字塔最小层级数，由图像自动计算出；
# constrastThreshold 对比度阈值用于过滤区域中的弱特征。阈值越大，检测器产生的特征越少；
# edgeThreshold 用于过滤掉类似边缘特征的阈值。请注意，其含义与 contrastThreshold 不同，即 edgeThreshold 越大，滤出的特征越少
# sigma，高斯输入层级，如果图像分辨率较低，则可能需要减少数值。
```

2. 其次对于上述特征点进行筛选，从而得到相似度较高的特征点

这里可以用 KNN 算法，但若是特征点过多的话就比较复杂，所有这里使用 RANSAC 方法，并规定来找到和一个点在另一张图片上最相似的两个点，比如：如果第一个点的近似程度远远大于第二个点，那么才认为第一个点是可靠的匹配点。

```
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])
```

另外一个就是使用已有的 KNN 算法，这个函数的原理如下：

```
rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
# featureA 和 featureB 表示两个特征向量集；
# K 表示按 knn 匹配规则输出的最优的 K 个结果。
在该算法中，输出的结果是：featureA（检测图像）中每个点与被匹配对象 featureB（样本图像）中特征向量进行运算的匹配结果。
则，rawMatches 中共有 featureA 条记录，每一条有最优 K 个匹配结果。
每个结果中包含三个非常重要的数据分别是 queryIdx，trainIdx，distance。
-#queryIdx：特征向量的特征点描述符的下标（第几个特征点描述符），同时也是描述符对应特征点的下标。
-#trainIdx：样本特征向量的特征点描述符下标,同时也是描述符对应特征点的下标。
```

-#distance：代表匹配的特征点描述符的欧式距离，数值越小也就说明俩个特征点越相近。

3. 通过上述得到的特征点对，来计算 H 矩阵

实际上所谓的转置矩阵 H，就是用来计算相对位置的。图像拼接的本质其实很简单，就是在一张图像合适的位置上拼接上另一张图片，而这个合适的位置就要去可以把重复的部分，也就是哪一块区域覆盖掉，而转置矩阵 H 实际上就是用来定位的。

```
cv2.findHomography(srcPoints,dstPoints,method=None,ransacReprojThreshold=None, mask=None, maxIters=None, confidence=None) --> (H, status)
计算多个二维点对之间的最优单映射变换矩阵 H（3*3），status 为可选的输出掩码，0/1 表示在变换映射中无效/有效。
#method（0，RANSAC, LMEDS, RHO）。
#ransacReprojThreshold 最大允许冲投影错误阈值（限方法 RANSAC 和 RHO）。
#mask 可选输出掩码矩阵，通常由鲁棒算法（RANSAC 或 LMEDS）设置，是不需要设置的。
#maxIters 为 RANSAC 算法的最大迭代次数，默认值为 2000。
# confidence 可信度值，取值范围为 0 到 1。
```

4. 利用矩阵 H 对另一张图片进行变换，并完成最后的拼接拼接

这最后一步比较就比较简单了，从以上代码可以看，已知矩阵 H，就可以对图片 A 的位置进行相对变换，想象一下一个长是 A 和 B 两个图片长的画布，现在通过 H 把 A 放在合适的位置（刚好可以重叠的区域的边界），那么在把图像 B 也放在这个画布原本图像重叠的边界，如此图像拼接是不是就完成了。

```
# 将图片 A 进行视角变换中间结果
        result = cv2.warpPerspective(imageA, H, (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
        # 将图片 B 传入]
        result[0:imageB.shape[0], 0:imageB.shape[1]] = imageB
        self.cv_show('result', result)
```

三、结果展示

实例一：

原图（source）：

融合后（cat）

实例二：

原图（source）：

融合后（cat）

实例三：

原图（source）：


融合后（cat）

三、代码运行说明

（1)、代码使用 python 编写，运行时请使用 pycharm 等 python 编译器。

（2)、image_stitching 总共包含三个文件:mian、miancat、Stitching。

（3)、Stitching.py 包含了图像拼接所用到的所有方法，miancat 为图像拼接的主函数，使用时只运行 miancat.py（确保图像的路径正确，以及包的完整性)。