

**计算机视觉工程实践实验报告**

|  |  |
| --- | --- |
| **姓 名:** | 高杨 |
| **学 号:** | 123106222810 |
| **学 院:** | 计算机科学与工程学院 |

**2024年 4月 23日**

**1.SIFT概述**

SIFT（尺度不变特征转换）是一种视觉算法，用于侦测与描述影像中的局部性特征。该算法通过在空间尺度中寻找极值点，并提取其位置、尺度、旋转不变性，能够在不受影像大小和旋转影响的情况下辨识物体。

SIFT特征对于光线、噪声以及轻微视角改变的容忍度极高，因此在物体辨识上具有重要作用。它基于物体上的局部外观兴趣点，相对容易获取，并且在庞大的特征数据库中能够准确识别物体而几乎不会出现误认。即使在部分物体被遮蔽的情况下，SIFT特征的侦测率也很高，只需要3个以上的SIFT特征就足以计算出位置与方位。

**1.1 SIFT算法的特点**

（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。

**1.2 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或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

（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。

**1.3 SIFT算法步骤**

SIFT算法大致上有如下四个步骤：

（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。

**2.图片拼接**

**2.1 关键点匹配**

再利用sift筛选出特征点后，为了进一步筛选匹配点，来获取优秀的匹配点，一般会采用Lowe’s算法来进一步获取优秀匹配点。

SIFT算法的作者Lowe提出了一种比较最近邻距离与次近邻距离的匹配方式，以排除由于图像遮挡和背景混乱而导致的无效匹配。该方法是这样的：对于一幅图像中的一个SIFT关键点，在另一幅图像中找出与之欧式距离最近的前两个关键点。如果最近距离与次近距离的比值（即ratio）小于某个阈值T，则将这一对匹配点视为有效匹配。这是因为对于错误匹配，由于特征空间的高维性，相似的距离可能有很多其他错误匹配，从而使得ratio值较高。

Lowe推荐的ratio阈值为0.8，但后续研究表明，在任意存在尺度、旋转和亮度变化的两幅图片进行匹配时，ratio取值在0.4~0.6之间效果最佳。小于0.4的很少有有效匹配点，而大于0.6的则存在大量错误匹配点。因此，建议根据实际需求选取ratio的取值。

**2.2 估计变换参数**

之后需要根据匹配的关键点对，通过估计变换参数，可以将一个图像中的特征点或对象映射到另一个图像中，从而实现图像的对齐和匹配。常见的方法包括通过特征点匹配来估计变换参数，例如利用 RANSAC 算法估计透视变换参数，或者利用光流法估计运动场景中的平移和旋转参数等。

RANSAC算法（随机抽样一致算法），采用迭代的方式从一组包含离群的被观测数据中估算出数学模型的参数。RANSAC算法假设数据中包含正确数据和异常数据（或称为噪声）。正确数据记为内点（inliers），异常数据记为外点（outliers）。同时RANSAC也假设，给定一组正确的数据，存在可以计算出符合这些数据的模型参数的方法。该算法核心思想就是随机性和假设性，随机性是根据正确数据出现概率去随机选取抽样数据，根据大数定律，随机性模拟可以近似得到正确结果。假设性是假设选取出的抽样数据都是正确数据，然后用这些正确数据通过问题满足的模型，去计算其他点，然后对这次结果进行一个评分。

**2.3 图像配准和拼接**

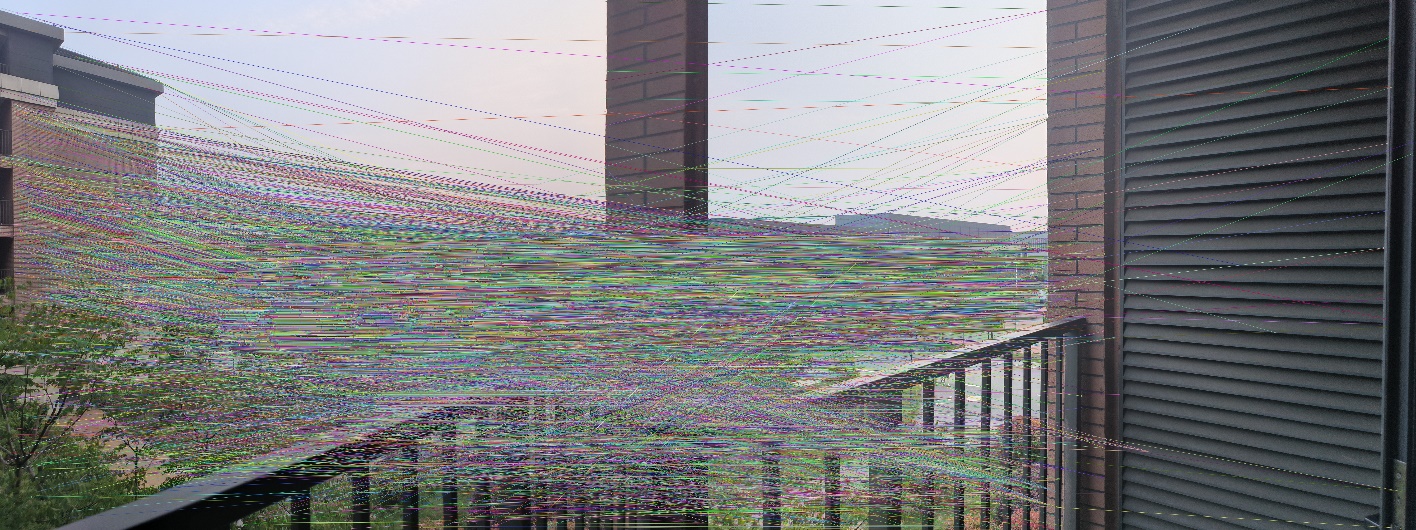
之后根据估计的变换参数，对图像进行配准，即将图像校正到同一坐标系下，使得它们可以正确地拼接在一起。最后，将配准后的图像进行拼接，通常是通过叠加或者混合的方式将图像拼接在一起，形成全景图像。

**3.实验结果**

手机拍摄的原始图片：



匹配结果以及最后拼接图：





实验中，最关键的是去实现重叠区域的匹配，在最后对图像进行拼接的时候，我尝试了几种版本。首先是对一侧图像进行透视变换，另一侧用原图直接填充，但这样会有明显的缝隙感觉。



第二版中对两侧都进行了透视变换，然后直接拼接在一起，但重叠区域会有明显不对应。



接着，尝试对重叠区域进行加权，即让拼接的接缝更加平滑点。实验中，我让重叠区域离左边近的用左图的权重大，离右边近的右图权重大，但这样依旧存在虚幻感。



最后，尝试将重叠区域划分成三份，左边、右边都使用对应的原图，而中间对距离进行加权。但此方法对于其他图的凭借可能效果并不好。

**4.相关代码**

**4.1 利用SIFT提取关键点和描述符**

    def registration(self,img1,img2):

        '''

        提取图像的关键点和描述子，并进行匹配，即找到两幅图像之间的几何关系

        图像可以乱序

        @param img1

        @param img2

        '''

        # 提取关键点和描述符

        kp1, des1 = self.sift.detectAndCompute(img1, None)

        kp2, des2 = self.sift.detectAndCompute(img2, None)

        # 特征点匹配

        matcher = cv2.BFMatcher()

        raw\_matches = matcher.knnMatch(des1, des2, k=2)

        #筛选较好的匹配点

        good\_points = []

        good\_matches=[]

        for m1, m2 in raw\_matches:

            if m1.distance < self.ratio \* m2.distance:

                good\_points.append((m1.trainIdx, m1.queryIdx))

                good\_matches.append([m1])

        # 绘制匹配结果

        img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good\_matches, None, flags=2)

        cv2.imwrite('matching.jpg', img3)

        # 如果匹配点数量大于最小匹配数量，则估计单应性变换矩阵

        if len(good\_points) > self.min\_match:

            image1\_kp = np.float32(

                [kp1[i].pt for (\_, i) in good\_points])

            image2\_kp = np.float32(

                [kp2[i].pt for (i, \_) in good\_points])

            H, status = cv2.findHomography(image2\_kp, image1\_kp, cv2.RANSAC,5.0)

        return H

**4.2根据配准结果和蒙版进行图像融合**

        '''

            图像融合函数，根据配准结果和蒙版进行图像融合，生成全景图像

        '''

        # 图像配准，获取单应性变换矩阵

        H = self.registration(img1,img2)

        # 获取图像尺寸信息

        height\_img1 = img1.shape[0]

        width\_img1 = img1.shape[1]

        width\_img2 = img2.shape[1]

        height\_panorama = height\_img1

        width\_panorama = width\_img1 +width\_img2

        # 创建全零全景图像

        panorama1 = np.zeros((height\_panorama, width\_panorama, 3))

        # 创建左侧图像的蒙版

        mask1 = self.create\_mask(img1,img2,version='left\_image')

        # 将左侧图像拼接到全景图像上，并根据蒙版进行权重调整

        panorama1[0:img1.shape[0], 0:img1.shape[1], :] = img1

        panorama1 \*= mask1

        cv2.imwrite('panorama1.jpg', panorama1)

        # 创建右侧图像的蒙版

        mask2 = self.create\_mask(img1,img2,version='right\_image')

        # 对右侧图像进行透视变换，并根据蒙版进行权重调整

        panorama2 = cv2.warpPerspective(img2, H, (width\_panorama, height\_panorama))\*mask2

        cv2.imwrite('panorama2.jpg', panorama2)

**4.3 重叠区域加权**

        res = np.zeros([rows, cols, 3], np.uint8)

        for row in range(0, rows):

            for col in range(0, cols):

                if not panorama1[row, col].any():

                    res[row, col] = panorama2[row, col]

                elif not panorama2[row, col].any():

                    res[row, col] = panorama1[row, col]

                else:

                    #if col < img1.shape[1]:

                    left\_new = (width\_panorama)//2-100

                    right\_new = (width\_panorama)//2+100

                    if col < left\_new:

                        res[row, col] = img1[row, col]

                    elif col > right\_new:

                        res[row, col] = img2[row, col]

                    else:

                        # 根据距离计算权重

                        srcImgLen = float(abs(col - left\_new))

                        testImgLen = float(abs(col - right\_new))

                        alpha = srcImgLen / (srcImgLen + testImgLen)

                        res[row, col] = np.clip(panorama1[row, col] \* (1-alpha)\*2 + panorama2[row, col] \*(alpha)\*2, 0, 255)

**5.总结**

通过本次实验，我了解SIFT算法提取特征点和描述符的大致流程，熟悉了特征的比较和匹配，并对相关的几何变换有了一定程度的了解。在完成图像拼接后，我尝试对拼接结果进行了评估和调优，对缝隙的平滑度以及色彩亮度一致性尝试进行改进。但实验最后并不能实现鲁棒性的拼接，对于重叠区域的处理依旧需要改进。