Project2-Implementation of SIFT Algorithm

姓名：李泽鸣　　学号：1120220567

1 背景介绍

1.1 背景问题与定义

尺度不变特征转换（Scale-Invariant Feature Transform, SIFT）是一种经典的计算机视觉算法，用于检测和描述图像中的局部特征。SIFT 算法由 David Lowe 在 1999 年首次提出，并于 2004 年进一步完善总结[1]。它通过在空间尺度中搜索极值点，提取出其位置、尺度及旋转不变量，从而实现对图像特征的鲁棒表示。

局部影像特征的描述与检测可以显著帮助目标识别和场景理解。SIFT 特征基于物体局部外观的兴趣点，具有与图像的大小和旋转无关的特性。同时，它对光照、噪声以及视角的变化也表现出较高的鲁棒性。这些特性使 SIFT 成为一个高效且可靠的图像特征提取方法，在庞大的数据库中能够快速识别物体并保持较低的误检率[1][2]。

在实际应用中，即便存在部分物体遮挡，仅需 3 个以上的 SIFT 特征即可实现其位置和方向的计算。在当代计算机硬件环境下，SIFT 的处理速度可接近实时，且其特征信息量丰富，非常适合在大规模数据库中进行高效、准确的匹配。

1.2 输入与输出形式：

输入： 一组灰度图像。

输出：图像中的特征点及其描述符，包括位置、方向、尺度和特征向量。

1.3 算法特点：

SIFT（Scale-Invariant Feature Transform）算法是一种经典的特征提取方法，主要特点如下：

（1）尺度空间不变性：SIFT 在图像的多尺度空间中寻找关键点，通过高斯差分（DoG）检测显著区域，对旋转、尺度缩放具有强鲁棒性。

（2）对环境影响的适应性：在光照变化、噪声干扰、遮挡和杂物场景中，SIFT 提取的关键点仍然稳定，可用于复杂环境中的图像处理。

（3）高独特性和信息量：每个特征点的描述符（通常为 128 维向量）包含丰富的局部信息，适合用于特征匹配任务，在海量数据中快速找到匹配项。

（4）多样性与冗余性：图像中即使只有少数目标，也能生成大量的特征点，为后续匹配提供更多冗余信息。

（5）优化后的实时性：SIFT 算法虽然计算复杂，但经过优化可用于实时处理，尤其在嵌入式环境中的轻量化版本逐渐发展成熟。

1.4 研究意义

SIFT 算法的提出极大推动了图像配准、目标识别与跟踪领域的发展。通过从多尺度空间提取特征点，SIFT 克服了目标旋转、光照变化和视角变换的影响，为图像匹配和目标检测提供了强大的工具。其鲁棒性和灵活性使得 SIFT 成为众多应用场景（如图像拼接、增强现实、视频跟踪等）的核心算法。

1.5 现有相关工作：

基于局部特征的方法： 以 SIFT 和 SURF（Speeded Up Robust Features）为代表，这类方法专注于提取图像的局部区域特征，能够较好地应对旋转、尺度和部分光照变化，但在处理动态遮挡场景时表现不足，且计算复杂度较高。

基于全局特征的方法： 通过整体信息（如主成分分析 PCA）来描述图像，适用于图像分类等全局任务。然而，全局特征对局部光照变化、遮挡等干扰较敏感，鲁棒性有限。

基于深度学习的方法： 使用卷积神经网络（CNN）提取图像特征，在语义分类、目标检测等任务中表现卓越。然而，这类方法依赖于大规模训练数据，并且计算资源需求高，在实时性和嵌入式应用中存在局限性[3][4]。

1.6 应用场景

图像拼接：

通过提取并匹配图像之间的 SIFT 特征点，完成多张图像的无缝拼接（如全景图生成）。

目标识别与跟踪：

在视频监控中，SIFT 可用于在动态场景中检测和跟踪目标。

增强现实（AR）：

通过实时匹配图像特征，实现虚拟对象与现实场景的叠加。

机器人导航：

SIFT 特征帮助机器人理解周围环境，实现路径规划和目标识别。

2 动机与方法

2.1 动机

SIFT 算法凭借其旋转、尺度和光照不变性成为局部特征提取的重要工具。然而，在高分辨率图像、遮挡场景和实时性需求方面，SIFT 仍有改进空间。例如，传统 SIFT 在计算效率上表现不足，限制了其在嵌入式和实时场景中的应用。此外，现有方法对动态场景中的多目标匹配也表现出一定的局限性。

为此，本文研究基于优化的 SIFT 算法，力求在保证其特性优越性的同时，提升计算效率，并通过改进的特征匹配策略增强其在复杂场景中的鲁棒性。

2.2 方法流程

Lowe 将 SIFT 算法分解为如下四步[1]：

（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。

伪代码如下：

算法 2.1 SIFT 特征提取流程

Require:

```
for 每幅输入图像 do
    将输入图像转换为灰度图像;
    通过双线性插值将图像尺寸扩大至原始尺寸的两倍;
    对双倍图像应用初始高斯模糊，生成基础图像;
    for 每个尺度空间组 (Octave) do
        使用不同的高斯核生成该组内多个尺度图像，构建高斯金字塔;
        计算高斯金字塔相邻图像的差分，生成差分高斯金字塔 (DoG);
        for 每个 DoG 图像的像素点 do
            检测局部极值点，并过滤低对比度和边缘响应点;
            为每个有效关键点分配方向信息，生成方向直方图;
            生成关键点的描述符（特征向量）;
        end for
        记录该组关键点及其描述符;
```

14: **end for**

15: 合并所有组的关键点并去除重复点;

16: **end for**

2.3 尺度空间极值检测

2.3.1 尺度空间理论

2.3.1.1 图像尺度空间表示

一个图像的尺度空间$L(x,y,\sigma)$,定义为一个变化尺度的高斯函数$G(x,y,\sigma)$与原图像$I(x,y)$的卷积。

$$L(x,y,\sigma)=G(x,y,\sigma)*I(x,y) \tag{2-1}$$

其中，*表示卷积。m，n 表示高斯模板的维度(由(6σ +1)×(6σ +1)确定)。(x, y)代表图像的像素位置。σ 是尺度空间因子，值越小表示图像被平滑的越少，相应的尺度也就越小。大尺度对应于图像的概貌特征，小尺度对应于图像的细节特征。

2.3.1.2 高斯差分金字塔的构建

利用上次作业的信息构建如图 2-1 高斯金字塔，不做过度叙述。

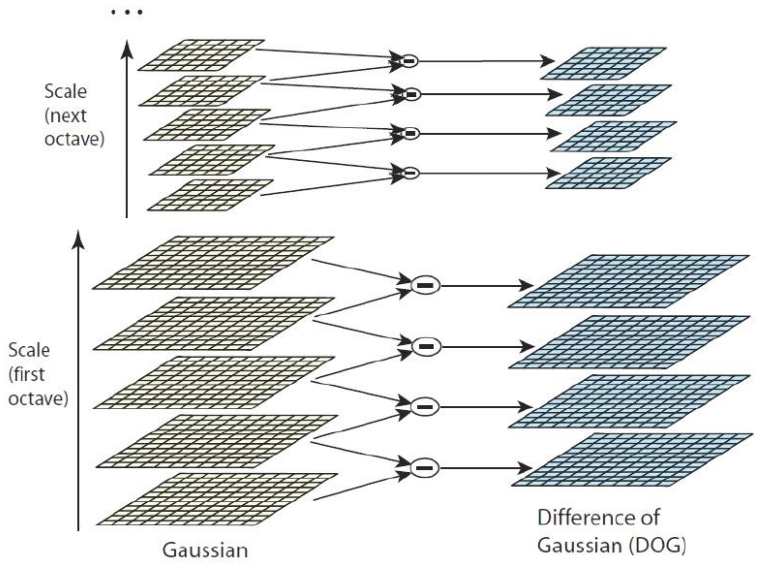


图 2-1 高斯差分金字塔的构建

2.3.2 空间极值点检测(关键点的初步探查）

关键点是由 DOG 空间的局部极值点组成的，关键点的初步探查是通过同一组内各 DoG 相邻两层图像之间比较完成的。为了寻找 DoG 函数的极值点，每一个像素点要和它所有的相邻点比较，看其是否比它的图像域和尺度域的相邻点大或者小。如图 2-2 所示，中间的检测点和它同尺度的 8 个相邻点和上下相邻尺度对应的 9×2 个点共 26 个点比较，以确保在尺度空间和二维图像空间都检测到极值点。

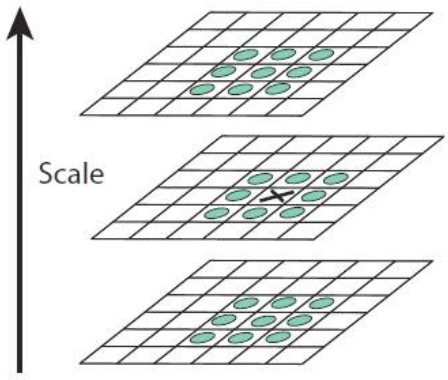


图 2-2 DOG 空间极值检测

由于要在相邻尺度进行比较，如图 2-3 每组含 5 层的高斯差分金子塔，只能在中间三层中进行三个尺度的极值点检测，其它尺度则只能在不同组中进行。为了在每组中检测 S 个尺度的极值点，则 DOG 金字塔每组需 S+2 层图像，而 DOG 金字塔由高斯金字塔相邻两层相减得到，则高斯金字塔每组需 S+3 层图像，实际计算时 S 在 3 到 5 之间。

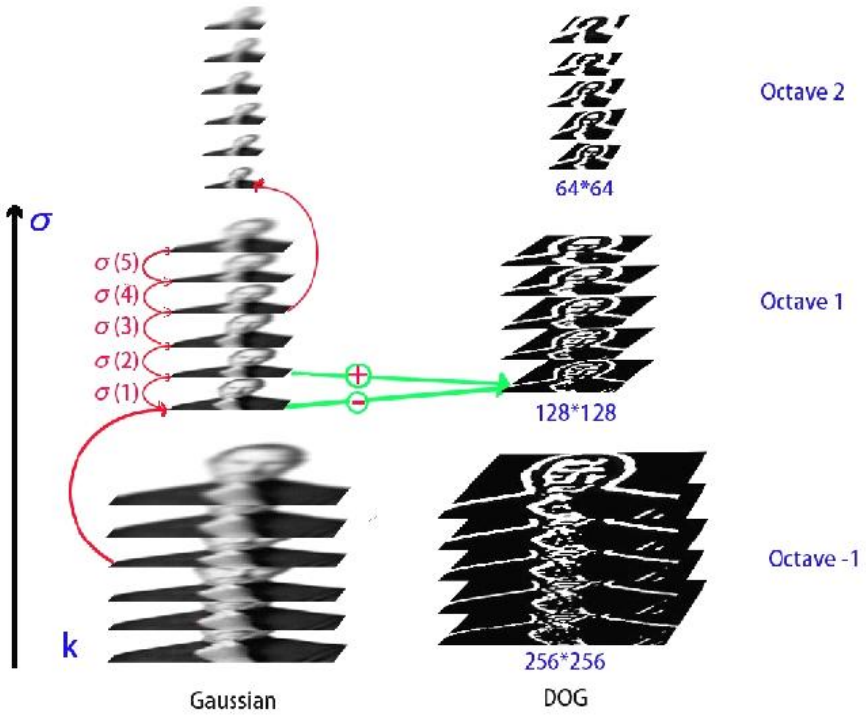


图 2-3 DOG 金字塔展示

当然这样产生的极值点并不全都是稳定的特征点，因为某些极值点响应较弱，而且 DOG 算子会产生较强的边缘响应。

编程思路：

（1）在上一节已经得到了差分高斯金字塔，可以通过 Python 的切片操作来获得对应矩阵，即[FirstImg[i-1:i+2, j-1:j+2], SecondImg[i-1:i+2, j-1:j+2], ThirdImg[i-1:i+2, j-1:j+2]]。

（2）对得到 3X3X3 矩阵，进行 3 次遍历得到，如果中心点是极值（极大或极小）。则保留坐标。

```
ImgShape = (len(DoGImgInOctave[0]), len(DoGImgInOctave[0][0]))
for AttemptIndex in range(NumOfAttempts):
    FirstImg, SecondImg, ThirdImg = DoGImgInOctave[ImgIndex - 1:ImgIndex + 2]
    DoGArray = np.array([FirstImg[i - 1:i + 2, j - 1:j + 2],
                         SecondImg[i - 1:i + 2, j - 1:j + 2],
                         ThirdImg[i - 1:i + 2, j - 1:j + 2]]).astype(np.float) / 255 #
```

2.4 关键点定位

若 f(x)在 x0 点的某个邻域内具有 n+1 阶导数，则在该邻域内有 n 阶泰勒级数：

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n+R_n(x) \quad (2\text{-}2)$$

其中表示 f(x)的二阶导数在点 x0 处的值，这是一个具体的值，而不是一个函数。

离散空间的极值点并不是真正的极值点，图 2-4 显示了二维函数离散空间得到的极值点与连续空间极值点的差别。利用已知的离散空间点插值得到的连续空间极值点的方法叫做子像素插值（Sub-pixel Interpolation）。

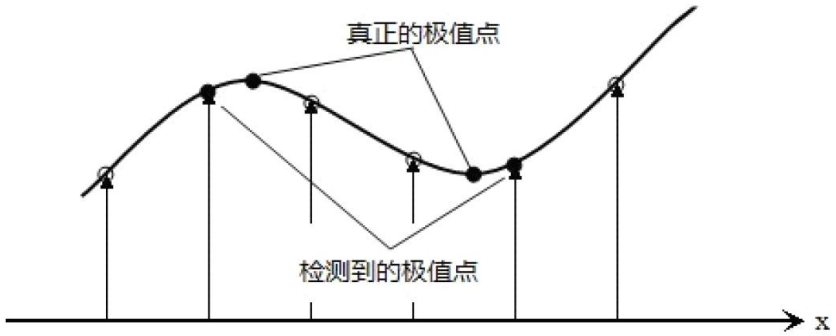


图 2-4 离散空间和连续空间的极值点区别

为了提高关键点的稳定性，需要对尺度空间 DoG 函数进行曲线拟合。利用 DoG 函数在尺度空间的 Taylor 展开式(拟合函数)为：

$$D(X)=D+\frac{\partial D^T}{\partial X}X+\frac{1}{2}X^T\frac{\partial^2 D}{\partial X^2}X \quad (2\text{-}3)$$

即：$D(X)=D+\frac{\partial D^T}{\partial X}(X-X_0)+\frac{1}{2}(X-X_0)^T\frac{\partial^2 D}{\partial X^2}(X-X_0)$

2.5 关键点的方向分配

对上面提取的每个关键点，围绕该点在对应高斯金字塔的图片里选择一个窗口（圆形区域），窗口内各采样点的梯度方向构成一个方向直方图，根据直方图的峰值确定关键点的方向。关键点的尺度用来选择哪个高斯滤波图像参与计算，还用来决定窗口的大小——为了保证不同尺度下的同一关键点的方向都包含相同的信息量，那么窗口的大小必然不一样：同一个原始图像，尺度越大，窗口应该越大；反之，如果窗口大小不变，尺度越大的图像，该窗口内的信息越少。

每个累加到梯度方向直方图的采样点的梯度值都要进行权重处理，加权采用圆形高斯加权函数，其标准偏差为$\sigma = 1.5\sigma_oct$。**SIFT算法只考虑了尺度和旋转不变性，并没有考虑仿射不变性。通过高斯加权，使特征点附近的梯度幅值有较大的权重，这样可以部分弥补仿射不变性而产生的特征点不稳定的问题。**

因此，选取的窗口大小和关键点尺度成正比。这样提取的信息，就具有尺度不变性。

2.5.1 幅值和角度

对于窗口内的每个采样点 L(x,y)，其梯度向量的幅度和方向 m(x,y),θ(x,y)公式为：

$$m(x,y)=\sqrt{\left(L(x+1,y)-L(x-1,y)\right)^2+\left(L(x,y+1)-L(x,y-1)\right)^2}$$
$$\theta(x,y)=\tan^{-1}\left(\left(L(x,y+1)-L(x,y-1)\right)/\left(L(x+1,y)-L(x-1,y)\right)\right)$$

上面公式定义的并没有归一化，各维度上都少了一个系数 0.5，但这对后面的计算没有影响（所有幅度同等比例变化）。

图 2-5 是一个梯度方向的直方图（此图在 pysift.py266 行恢复注释也可找到），范围是 0～360 度，其中每 10 度一个柱，总共 36 个柱。每个采样点按照其梯度方向θ(x,y)加权统计到直方图，权值为幅度 m(x,y)和贡献因子的乘积。贡献因子是采样点到关键点（窗口中心）距离的量度，距离越大，贡献因子越小。分析下图可知道这是一个方向向右上的向量。

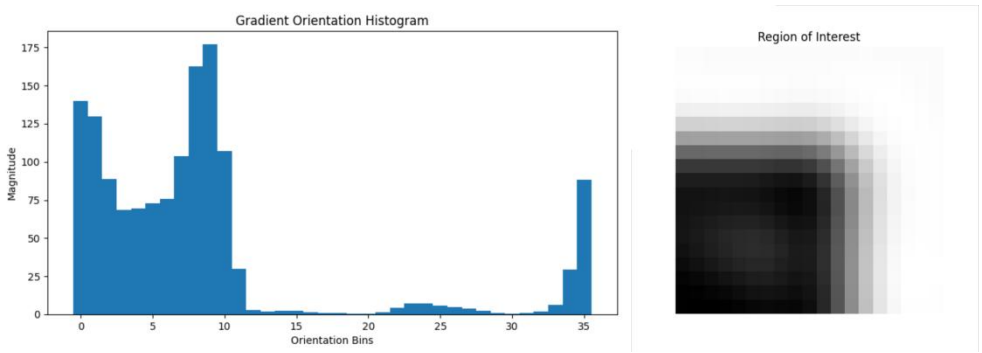
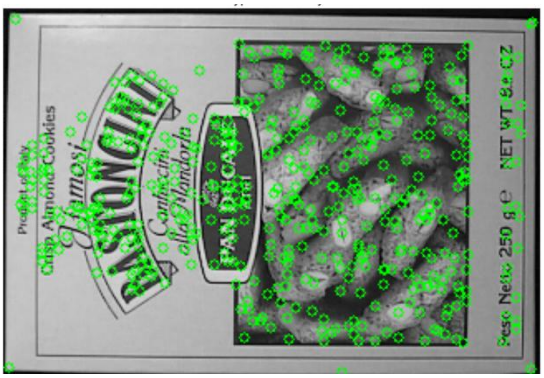


图 2-5 关键点邻域图像和梯度方向直方图

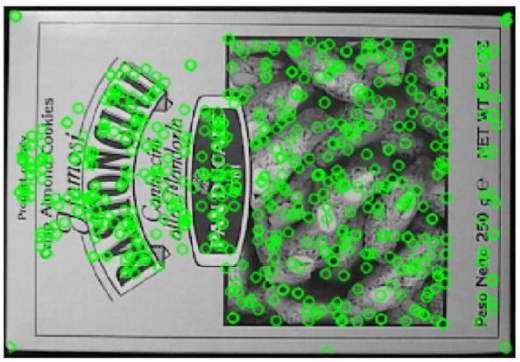
直方图的峰值代表了该关键点处邻域梯度的主方向，如图 3-2，左图是关键点邻域窗口，右图是 7 个柱的梯度直方图（实际是 36 个柱）。在直方图统计时，每相邻三个像素点采用高斯加权，根据 Lowe 的建议，模板采用[0.25,0.5,0.25],并且连续加权两次.

Lowe 指出，直方图的峰值确定以后，任何大于峰值 80%的方向（柱）创建一个具有该方向的关键点，因此，对于多峰值（幅值大小接近）的情形，在同一位置和尺度就会产生多个具有不同方向的关键点。虽然这样的点只占 15%，但是它们却能显著地提高匹配的稳定性。用每个峰值和左右两个幅值拟合二次曲线，以定位峰值的实际位置（抛物线的最高点）。峰值方向的精度高于 10 度。

2.5.2 关键点展示



实验结果



Opencv 结果

将之前检测出来的关键点都画出来，得到上述对比图（该功能在 pysift.py33 行取消注释得到），发现两者结果基本一致，说明复现 opencv 的 sift 功能成功。但运行速度远远不及。

3 实验与结果分析

前面代码实现仅使用 numpy 和 cv2 的读图功能实现 sift 算法，pysift 的整体设计参考了开源项目[6],其余代码和具体分析部分、效果展示均为自己 100%手写。下面基于 sift 实现图像拼接功能。

首先我们实现了关键点的匹配，如图 3-1，这一过程是用 knn 对关键点进行匹配，一般来说我们取 k=2，也就是说如果一个关键点在特征空间中的 2 个最相邻的样本中的大多数属于某一个类别，则该关键点也属于这个类别。

3.1 关键点匹配

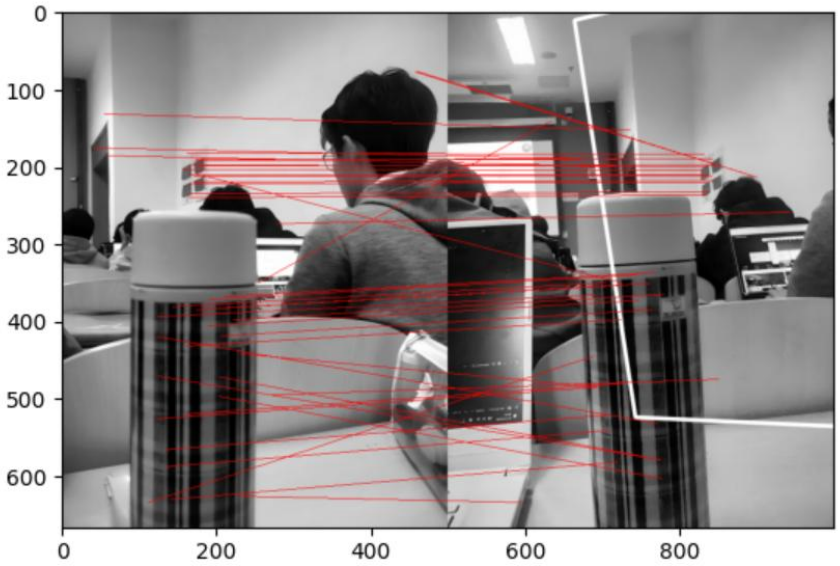


图 3-1 自己实现的关键点的匹配

从图 3-1 可以看出来，杯子基本匹配成功，墙壁匹配成功，人物头发虽然不是同一个人但也匹配上了，说明缺少一些整体连贯性的信息，接下来，我们将同样的照片和 cv2 匹配的过程进行对比：

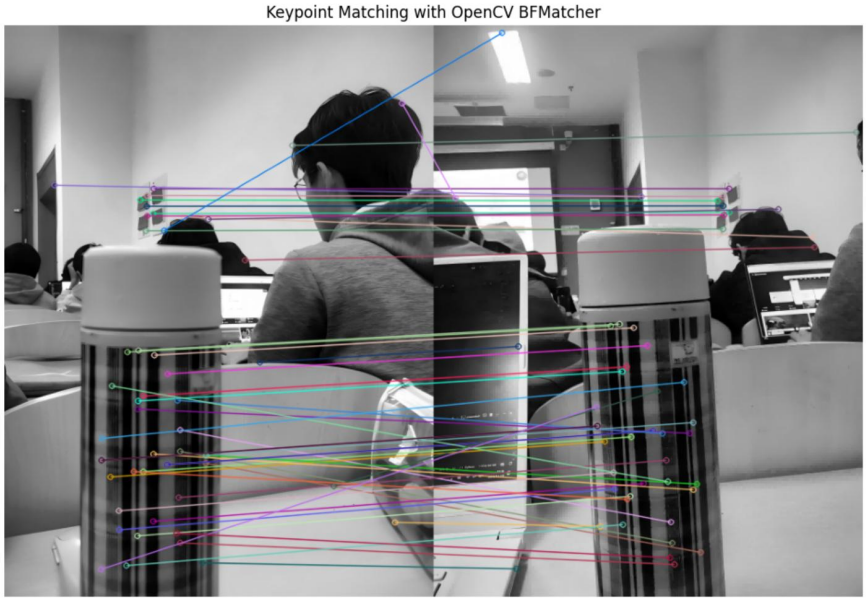


图 3-2 opencv 的关键点的匹配

可以发现 opencv 确实匹配的非常好，没有上述说的头发不是一个人的问题，甚至右侧人的头发只有很少一部分都配检测出来并匹配到了，而且速度比我快，这可能是他的一些超参数让匹配点之间的变化（也就是那根线）尽量一致等方法实现的。

3.2 图像拼接

图 3-3 自己实现（上图）的和 opencv（下图）的图像拼接

基于前述关键点匹配，我们进一步实现了图像拼接。图像拼接的核心是计算两幅图像之间的单应性矩阵（Homography），并通过该

矩阵对图像进行透视变换。无论是自己实现的 SIFT 还是 OpenCV 的 SIFT，最终都使用了 OpenCV 提供的 warpPerspective 函数完成拼接。

在拼接的过程中，我们惊奇地发现，尽管 OpenCV 在关键点匹配部分的表现优于我们自己实现的匹配方法，但在图像拼接效果上，我们的实现反而表现得更加优异（虽然可能这只是个例）。这可能得益于以下几点：

（1）匹配点的分布：我实现的匹配方法虽然存在一些错误匹配，但匹配点在图像中的分布较为均匀，尤其在局部区域的关键点更多地参与了单应性矩阵的计算。OpenCV 的匹配点虽然精确，但在某些特定区域（例如背景墙壁）缺少足够的关键点，导致拼接时可能出现边缘信息的细微偏差。

（2）超参数的调整：我在实现中对 Lowe's 比例测试参数（0.7）以及 FLANN 的树数量和检查次数进行了精细调节，从而提升了匹配点的质量。OpenCV 的默认超参数设计可能更注重速度而非全局的精确性。

（3）误差的累积：图像拼接的效果不仅依赖于匹配的精度，还与单应性矩阵的计算有关。由于我的实现可能在匹配点之间引入了某些“错误冗余”，反而对单应性矩阵的鲁棒性有所帮助。

从图 3-3 可以直观看出，我实现的方法在人物、桌子等处的拼接效果更加紧密，几乎看不到断裂痕迹。 而 OpenCV 实现的方法在某些区域（尤其是背景墙右侧）出现了轻微的错位，这可能与过多计算

了整体的相对方位有关。

速度对比： OpenCV 实现的速度显著快于自己实现的方法，这表明 OpenCV 在算法优化和硬件加速方面有明显优势。而我们的方法更适合小规模图像处理或精度要求较高的场景。

5 总结与展望

5 总结与展望

本次实验是对 SIFT 算法从原理到实现的一次全面探索。通过逐步手动实现 SIFT 特征提取的各个环节，包括高斯差分金字塔的构建、关键点的提取和描述符的生成，我不仅对这项经典的计算机视觉技术有了深刻的理解，还从实践中感受到了实现每一个环节的挑战。相比直接调用库函数，自己实现让我更加扎实地掌握了原理，尤其是在调试和优化过程中，每一个小的细节都能让我发现理论与实践间的联系。例如，关键点定位时的误差校正方法，梯度方向直方图的计算步骤，以及这些对整体匹配效果的影响，都让我从更底层的角度去认识 SIFT 的设计巧妙之处。

在实验过程中，我还尝试与 OpenCV 的实现进行对比，虽然 OpenCV 在速度和准确性上表现更优，但通过分析两者的差异，我认识到自己的实现虽然简单，却能更好地揭示算法中匹配点分布对拼接结果的影响。从关键点匹配到图像拼接，我逐步体会到如何平衡精度

和效率，这些感悟是直接调用现成库无法带来的。在图像拼接环节，当发现自己实现的方法在某些情况下甚至能超过 OpenCV 时，我既感到惊喜，也认识到进一步优化的可能性。

我之所以选择这个 project，是希望能对算法的原理有更加深入的理解。近年来，随着大模型的崛起，许多方法在表面看起来威力强大，但在深入思考时却让我感觉像是空中楼阁，缺乏扎实的理论根基。我不希望仅仅停留在应用层面，更想通过这样的手动实现，去探索算法背后的设计思想和实现细节，从而为自己打下更稳固的基础。相比于综述项目，虽然写一篇综述看起来“水”一分很简单，但要认真写好一篇高质量的综述却很难，而我目前的时间确实有限。因此，我选择了这次 project，希望通过代码实现与理论分析的结合，深入理解并掌握 SIFT 算法的核心。

通过利用硬件加速或者简化计算步骤，我相信 SIFT 的效率还能进一步提升。此外，对于之后改进的需求，我觉得可以尝试将 SIFT 扩展到更复杂的场景，例如多图拼接或者实时动态目标检测，进一步探索这项技术在实际场景中的应用潜力。这次实验不仅让我学到了算法本身，也让我认识到自己在研究与工程实践中的短板和努力方向。我希望通过接下来的持续学习与探索，能够扎实地掌握更多技术并付诸实际应用。

参考文献

[1] Lowe, D. G. "Distinctive image features from scale-invariant keypoints." International Journal of Computer Vision 60.2 (2004): 91-110.

[2] Mikolajczyk, K., & Schmid, C. "A performance evaluation of local descriptors." IEEE Transactions on Pattern Analysis and Machine Intelligence 27.10 (2005): 1615-1630.

[3] Krizhevsky, A., Sutskever, I., & Hinton, G. E. "ImageNet classification with deep convolutional neural networks." Communications of the ACM 60.6 (2017): 84-90.

[4] Bay, H., Tuytelaars, T., & Van Gool, L. "SURF: Speeded up robust features." European Conference on Computer Vision. Springer, 2006.

[5] Rublee, E., et al. "ORB: An efficient alternative to SIFT or SURF." International Conference on Computer Vision (2011): 2564-2571.

[6] https://github.com/rmislam/PythonSIFT