preprocess.py

detect keypoints()：使用 cv2.SIFT_create()创建一个 SIFT 对象，并使用 sift.detectAndCompute()计算关键点和描述符。

create feature matches()：使用 cv2.BFMatcher()创建一个 BFMatcher 对象，并使用 bf.knnMatch()找到最近的两个邻点。然后应用 Lowe 比率测试来过滤无效匹配。Lowe 比率测试的思想是，真实的匹配点应该与第二近的点不同。

create ransac matches()：使用 cv2.findEssentialMat()获取基本矩阵和选定的 2D 点的内点掩码。将用于 sfm.py 的内点保存到本地存储。

create scene graph()：首先枚举一对图像，并使用 create ransac matches()获取它们的匹配。如果匹配数量超过最小内点数，就为这对图像建立一条边。可以通过调用 create ransac matches()从本地存储中获取匹配数量。

sfm.py

get init image ids()：这个函数简单地找到一对图像，它们具有最多的匹配特征。具体来说，匹配特征的数量可以从本地存储中获取。然后枚举场景图中的每条边，并更新具有最多匹配特征的值。然后返回对应于最大值的初始图像对。

get init extrinsics()：使用 np.linalg.svd()获取分解矩阵。然后按照公式获取可能的旋转矩阵 R 和平移向量 t。为了保持 R 在右手坐标系中，如果 det(R) < 0，让 R = -R。然后等待测试四种组合的(R, t)。然后使用 cv2.triangulatePoints 获取 3D 点，并进一步使用深度公式的符号来确定点是否在两个视图的前方。为所有四种(R, t)计算这个数量，由于噪声，具有最多点的组合对应于正确的答案。

get reprojection residuals：首先使用内在矩阵和外在矩阵计算投影矩阵。然后通过 x = PX 计算重投影的 2D 点。然后计算并返回同一平面上的点之间的欧几里得距离。

get next pair()：枚举一对图像，其中一个来自已注册集合，一个来自未注册集合。然后返回具有最多匹配特征的对。

solve pnp()：调用 cv2.solvePnP()获取旋转向量和平移向量。然后使用 cv2.Rodrigues()将旋转向量转换为旋转矩阵。然后调用 get reprojection residuals 获取与旋转矩阵和平移向量对应的残差。然后这个函数维护并返回具有最多内点的(R, t)对。

add points3d()：在这个函数中，简单地调用 triangulate 进行三角测量，使用相应的匹配、外在矩阵和内在矩阵获取 3D 点。

bundle_adjustment.py

compute ba residuals()：首先使用内在和外在参数计算张量 P，其形状是 C×3×3。然后应用 points3d idx 到 points3d 以重新组织点以匹配 point2d，其形状是 N×4，并将 camera idxs 应用到 P 以重新组织点以匹配 point2d，其形状是 N×3×4。然后执行 np.einsum()以获取 P 和 point3d 的乘积，以获取每个 point2d 的重投影。最后，可以通过计算 point2d 和 point2d 重投影之间的欧几里得距离来获得残差。