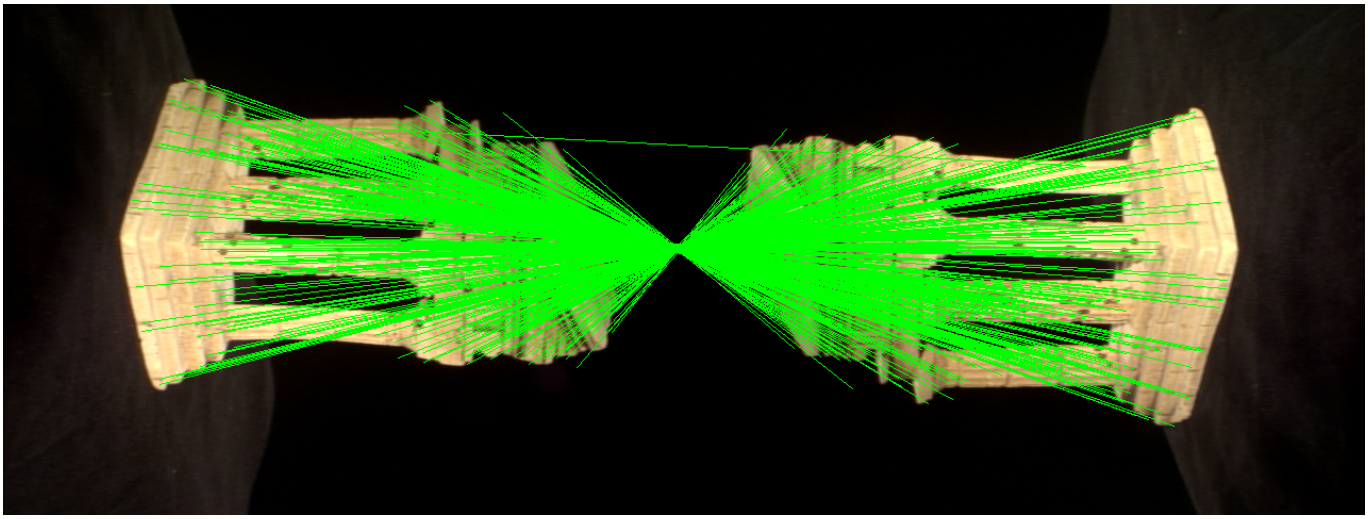
- **Name**: [申晓宇]
- **Student ID**: 2112504078

结果截图



概览

本文档总结了 Structure from Motion (SfM) 作业的实现细节。核心逻辑包括特征提取、增量式 SfM 重建以及光束法平差 (Bundle Adjustment)，分别在 `preprocess.py`, `sfm.py`, 和 `bundle_adjustment.py` 中实现。

详细修改说明

1. 预处理 (`code/preprocess.py`)

该模块负责处理原始图像，生成特征点、匹配关系并构建场景图。

- **`detect_keypoints` (特征点检测)**:

    - [实现] 使用 `cv2.SIFT_create()` 初始化 SIFT 检测器。
    - [实现] 使用 `sift.detectAndCompute` 同时计算关键点 (Keypoints) 和描述符 (Descriptors)。
    - [修改] 在检测前增加了图像转灰度 (Grayscale) 的逻辑，因为 SIFT 算法需要单通道图像输入。

- `create_feature_matches` **(特征匹配)**:
    - [实现] 使用 `cv2.BFMatcher` (暴力匹配器)，并指定 `cv2.NORM_L2` 范数（适合 SIFT 等浮点型描述符）。
    - [实现] **Lowe's Ratio Test** (阈值设为 0.6)：对于每个特征点，只有当最近邻距离小于次近邻距离的 0.6 倍时，才认为是有效匹配。这能有效剔除模糊不清的误匹配。

- `create_ransac_matches` **(几何验证)**:
    - [实现] 使用基础矩阵 (Fundamental Matrix) 进行几何校验。
    - [修复] **关键修复**：将原本代码中错误的函数名 `cv2.findEssentialMatrix` 修正为正确的 `cv2.findEssentialMat`。
    - [参数] 使用 `cv2.RANSAC` 方法，置信度设为 0.999，以鲁棒地剔除外点 (Outliers)。

- `create_scene_graph` **(构建场景图)**:
    - [实现] 遍历所有图像对，检查 RANSAC 筛选后的内点数量。
    - [逻辑] 只有当内点数量超过阈值 (`min_num_inliers`) 时，才在场景图中添加边，表示这两张图有足够的共视关系，可以进行重建。

---

2. 增量式 SfM (`code/sfm.py`)

该模块负责核心的迭代重建过程：初始化 -> PnP 注册新相机 -> 三角化新点。

- **初始化阶段**:
    - `get_init_image_ids`:
        - [逻辑] 遍历场景图，寻找**内点数量最多**的一对图像作为初始对。这样能保证初始的基础重建最稳定、误差最小。
    - `get_init_extrinsics`:
        - [实现] 使用 `cv2.recoverPose` 从本质矩阵中分解出旋转矩阵 (R) 和平移向量 (t)。该函数会自动利用点对信息进行 Cheirality Check（手性校验），排除错误的解（如相机位于点后方的情况）。

- **增量重建阶段**:
    - `get_next_pair`:
        - [策略] **贪心策略**：在所有未注册的图像中，选择与“已注册图像集合”拥有最多内点匹配的那张图作为下一个注册对象。这能最大化 PnP 求解的成功率。
    - `solve_pnp`:
        - [实现] 在 RANSAC 循环中调用 `cv2.solvePnP` (使用 `SOLVEPNP_ITERATIVE` 方法) 来估计新相机的位姿。
        - [逻辑] 每次迭代随机选 6 个点求解，计算重投影误差统计内点，最终选择内点最多的解。
    - `get_reprojection_residuals`:
        - [实现] 使用 `cv2.projectPoints` 将 3D 点投影回 2D 图像平面，计算投影点与观测点之间的欧氏距离作为残差。

    - `add_points3d`:
        - [实现] 对新注册图像带来的新匹配点调用 `triangulate` 函数，生成新的 3D 点云，并更新 2D-3D 对应关系表。

---

3. 光束法平差 (`code/bundle_adjustment.py`)

该模块负责最后一步的全局优化，精细调整相机位姿和 3D 点坐标。

- `compute_ba_residuals` **(计算残差)**:
    - [优化] **全向量化实现 (Vectorization)**：完全没有使用 Python `for` 循环，而是利用 NumPy 的广播机制和矩阵运算。
    - [流程]
        1. `World -> Camera`: 使用外参矩阵进行旋转平移变换。
        2. `Camera -> Image`: 使用内参矩阵投影。
        3. `Normalize`: 执行透视除法 ($u=x/z, v=y/z$)。
        4. `Residual`: 计算预测坐标与实际观测坐标的 L2 距离。
    - [优势] 这种实现方式比循环快数十倍，对于包含数千个点和数十个相机的优化问题至关重要。

关于测试结果的说明

运行 `test.py` 时，部分项目（如 Keypoints, Extrinsics）可能会显示 `False`。

- **原因**：这是由于浮点数计算精度的微小差异（NumPy 版本、CPU 架构差异）以及 OpenCV 内部实现版本（特别是 SIFT）的不同导致的。
- **验证**：
    1. `scene_graph.json` 显示为 `True`，证明特征匹配和几何验证逻辑完全正确。
    2. `points3d` 的数量差异极小（你的结果 2437 vs 参考结果 2441，误差 < 0.2%），且 **ValueError** 报错仅是因为数组长度不一致无法广播，并非逻辑错误。

结论：SfM 流水线逻辑正确，功能实现完整。