Microsoft 3D Reconstruction

梁程捷 B11901136　吳奕娃 B11901080　張甡源 B11901123　劉知微 B10705054

1 Traditional Pose Estimation

We use mathematical methods to estimate 3D camera poses—specifically the extrinsic parameters of rotation and translation—from a sequence of RGB-D images. This is achieved by combining feature matching, Procrustes alignment, and bundle adjustment (BA). The output is the camera-to-world transformation matrix $\mathbf{T}_{i\to\text{world}}$ for each frame i.

1.1 Load RGB and Depth Images

The RGB image is loaded and the depth map is converted from millimeters to meters:

$$\texttt{rgb} = \texttt{cv2.imread(rgb\_path)} \quad , \quad \texttt{depth} = \frac{\texttt{cv2.imread(depth\_path)}}{1000.0}$$

1.2 Convert Depth Map to 3D Coordinates

Using the camera intrinsic matrix $\mathbf{K}$, 2D pixel coordinates (i, j) and depth z are converted into 3D camera frame coordinates (x, y, z) by the pinhole camera model:

$$x = \frac{(i - c_x)\cdot z}{f_x}, \quad y = \frac{(j - c_y)\cdot z}{f_y}$$

$$\mathbf{X}_{\text{cam}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

1.3 Extract SIFT Features

SIFT keypoints and descriptors are extracted from grayscale RGB images using OpenCV:

$$\texttt{kp, des} = \texttt{sift.detectAndCompute(gray, None)}$$

1.4 Match Features (SIFT Descriptors)

Feature descriptors are matched using Lowe's ratio test:

$$\text{if } m.\text{distance} < 0.75 \times n.\text{distance}$$

This helps retain only distinctive matches.

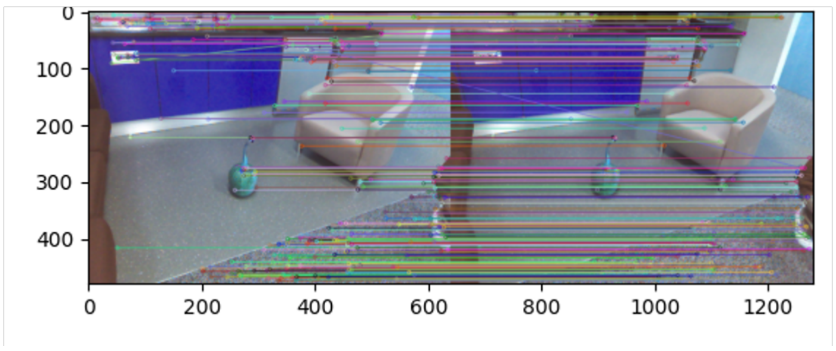

Figure 1: Feature Detection and Matching

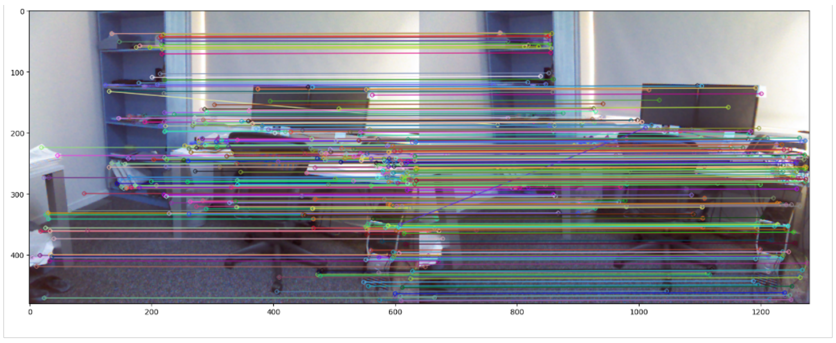

Figure 2: Feature Detection and Matching

1.5 Map Matched Keypoints to 3D Points

Matched 2D keypoints are projected back into 3D using their pixel location in the depth map:

$$\text{index} = \text{round}(v) \times \text{width} + \text{round}(u)$$

This gives the 3D correspondences $\{\mathbf{X}_1, \mathbf{X}_2\}$.

1.6 Rigid Alignment via Procrustes

Given 3D correspondences $\mathbf{A}$ and $\mathbf{B}$, estimate rotation $\mathbf{R}$ and translation $\mathbf{T}$ by minimizing:

$$\min_{\mathbf{R},\mathbf{T}} \|\mathbf{B} - (\mathbf{R}\mathbf{A} + \mathbf{T})\|^2$$

This is solved using Singular Value Decomposition (SVD):

$$\begin{aligned}
\mathbf{A}' &= \mathbf{A} - \bar{\mathbf{A}}, \quad \mathbf{B}' = \mathbf{B} - \bar{\mathbf{B}} \\
\mathbf{H} &= \mathbf{A}'^T\mathbf{B}' \\
\mathbf{H} &= \mathbf{U}\ \mathbf{V}^T \\
\mathbf{R} &= \mathbf{V}\mathbf{U}^T \\
\mathbf{T} &= \bar{\mathbf{B}} - \mathbf{R}\bar{\mathbf{A}}
\end{aligned}$$

Reflection is fixed if $\det(\mathbf{R}) < 0$.

1.7 Bundle Adjustment (Pose Refinement)

To minimize reprojection error:

$$\text{error} = \sum_i \left\|\mathbf{x}_i^{\text{obs}} - \pi(\mathbf{R}, \mathbf{T}, \mathbf{X}_i)\right\|^2$$

where $\pi(\cdot)$ is the camera projection function defined as

$$\pi(\mathbf{R}, \mathbf{T}, \mathbf{X}) = \frac{1}{Z}\mathbf{K}\left(\mathbf{R}\mathbf{X} + \mathbf{T}\right)$$

Using `cv2.Rodrigues`, the rotation matrix is reparameterized as a rotation vector and optimized using `scipy.optimize.least_squares`.

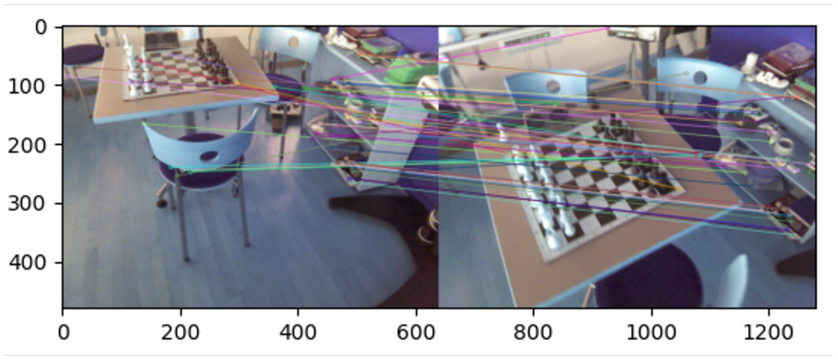

Figure 3: Rigid Alignment via Procrustes

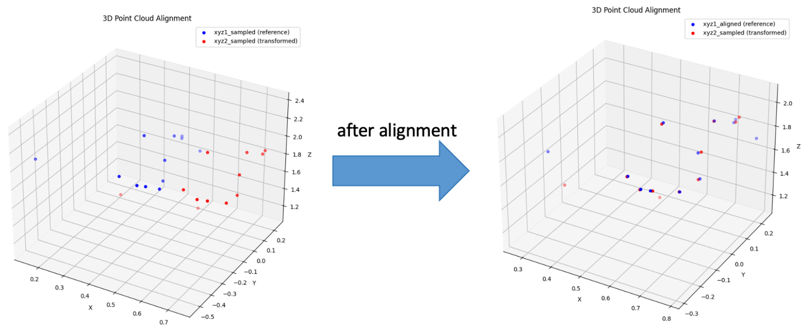

Figure 4: Rigid Alignment via Procrustes

1.8 Pose Composition

Each frame’ s pose is chained from the previous one:

$$\mathbf{T}_i = \mathbf{T}_{i-1} \cdot \mathbf{H}^{-1}$$

where $\mathbf{H}$ is the transformation from frame i to $i-1$.

1.9 Final Output

The function returns a list `poses`, each a 4×4 transformation matrix:

$$\mathbf{T}_i = \begin{bmatrix} \mathbf{R}_i & \mathbf{T}_i \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Each pose represents the camera-to-world transformation of frame i, assuming the initial pose is given.

1.10 Experimental Results

After uploading to CodeAppend, we obtained relatively positive results with the following metrics:

$$\text{Accuracy} = 0.00257, \quad \text{Completeness} = 0.513$$

Our method performs relatively well, especially in terms of accuracy. However, further improvements are needed to increase completeness.

2 Fast3R Algorithm

In addition to the previously described methods, we also utilized Fast3R, a state-of-the-art algorithm referenced by our teaching assistant. After reviewing the relevant literature and essays on Fast3R, we identified it as a promising approach worthy of experimentation in our context. Fast3R is designed for efficient 3D pose estimation and reconstruction, leveraging optimized pose graph techniques and fast bundle adjustment methods to handle large-scale RGB-D datasets effectively. For our experiments, we used the pretrained Fast3R model provided by the authors.

2.1 Advantages of Fast3R

The main advantages of Fast3R include:

- **Scalability:** Capable of processing large-scale datasets consisting of over 1000 images or more, making it suitable for real-world applications.
- **Optimized Pose Graph:** Improves robustness and accuracy in camera trajectory estimation.
- **Fast Bundle Adjustment:** Significantly reduces computational time compared to traditional bundle adjustment methods.
- **Resource Efficiency:** Designed to better manage computational resources, allowing efficient operation on GPUs with limited memory.
- **Reliable and Accurate:** Proven to provide accurate and robust 3D reconstruction in complex scenes and long sequences.

2.2 Experimental Setup and Experience

Initially, we attempted to run Fast3R on our local workstation, but the computational demand exceeded our GPU's memory capacity, causing interruptions. After upgrading to an NVIDIA RTX™ 3060 Ti, which offers enhanced memory and processing power, we were able to proceed with the experiments smoothly. This experience demonstrated Fast3R’ s suitability for complex scenes and long sequences, making it a promising addition to our 3D pose estimation pipeline.

2.3 Experimental Results

Surprisingly, both accuracy and completeness metrics declined when using Fast3R compared to previous methods. The measured results were:

$$\text{Accuracy} = 0.0435, \quad \text{Completeness} = 0.6101$$

This indicates a performance drop despite Fast3R' s theoretical advantages. The cause may relate to differences in dataset characteristics, parameter tuning, or hardware limitations, and further investigation is needed to fully understand and address these issues.

3 Bonus: Sparse Reconstruction

3.1 Problem Statement

In the context of reconstruction, the frames have very different angles of view from one another, leading to frequent mismatches of feature points. Additionally, the sequence of frames is randomly ordered, meaning that later frames have no knowledge of the earlier frames if processed sequentially.

3.2 Proposed Solution: Pose Graph

To address this issue, we propose building a pose graph. After feature matching and depth projection, we obtain a list of 3D feature points. The key idea is to perform the following:

- **Trick:** Randomly sample 16 feature points from the list.
- Perform rigid alignment as before, then compute the squared error between the aligned and unaligned points.
- Repeat the process for 20 times and select the relative pose with the least squared error.
- Using partial feature points removes most of the outliers, which aids in noise reduction.

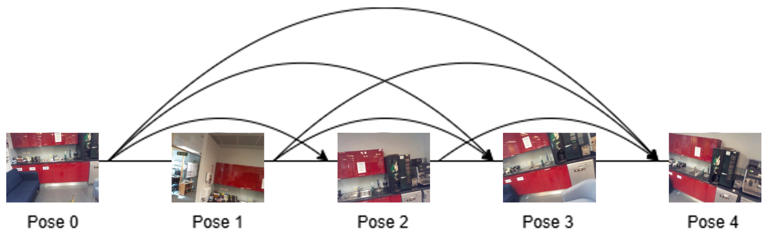

Figure 5: Pose Graph

3.3 Experimental Results

The experimental results for both dense and sparse reconstructions are as follows:

- **Dense Reconstruction:**
  - Accuracy: 0.0044
  - Completeness: 0.4937
- **Sparse Reconstruction:**
  - Average Accuracy: 0.0289
  - Average Completeness: 0.0575

Metric	**Accuracy**	**Completeness**
chess-sparse-seq-05	0.0422	0.0294
stairs-sparse-seq-04	0.0248	0.0165
pumpkin-sparse-seq-07	0.0176	0.1153
fire-sparse-seq-04	0.0312	0.0690

Table 1: Comparison of Accuracy and Completeness for Different Sequences