Project EEN020 Computer Vision

Erik Norlin

21 January 2024

Algorithm

The algorithm consists of four parts; rotation averaging to estimate the absolute rotations of the cameras, translation registration to estimate the translations of the cameras, camera refinement to bundle adjust the cameras, and triangulation of the final 3D-reconstruction.

Rotation averaging

All absolute rotations are obtained by chaining the relative rotations of each adjacent camera pair as $R_j = R_{i,j}R_i$. The relative rotations are extracted from robustly estimated essential matrices and normalized by taking the singular value decomposition of $R_{i,j}$ and setting $R_{i,j} = UV^T$.

Essential matrices for each adjacent camera pair are robustly estimated using image correspondences obtained from SIFT-points in a RANSAC loop. `estimate_E_robust` from assignment 4 is extended by also extracting essential matrices from an estimated homography, checking the validity of every estimated essential matrix, and dynamically changing the number of RANSAC iterations T_E and T_H by computing

$$T = \left\lceil \frac{\ln(1-\alpha)}{\ln(1-\epsilon^s)} \right\rceil \tag{1}$$

Since this formula can be aggressive in the sense that the number of RANSAC iterations can be very large, and sometimes be very small, they are therefore capped at a maximum and minimum number of iterations. The number of RANSAC iterations can also be scaled by a factor.

In each iteration both an essential matrix and a homogropahy are estimated for minimal samples using an 8-point DLT method and a 4-point DLT method respectively. From the homography, two other essential matrices can be extracted. The reason for extracting essential matrices from a homography is that as the scene becomes more planar the risk of degenerate solutions obtained by the 8-point method increases, whereas a homography becomes better at mapping the image points of two cameras facing the scene. Inliers for an essential matrix are measured with point-to-line distance (point to epipolar line), and inliers for a homography are measured with point-to-point distance (point to projected

point). The pixel-threshold for computing inliers of the homography is set to three times larger than for computing inliers of the essential matrices, and the pixel-threshold for computing inliers of the essential matrices is the threshold of recommendation in each data set. A valid essential matrix has rank 2 and is therefore always tested for this. If a new best essential matrix is estimated but is not rank 2 then it is discarded. T_E and T_H are adjusted every time a better valid essential matrix is found.

When RANSAC has terminated, four cameras are extracted from the best essential matrix *outside* the RANSAC loop to speed up the process. The cheirality test is perfomed on these cameras, and the camera that yields the most inliers in front of both cameras is selected as the correct solution. For the vast majority of the time this works wonderful. However in rare cases, triangulating 3D-points using the correct extracted camera from the returned valid essential matrix results in non-sensical "sprays" of point clouds.

Translation registration

An initial 3D-reconstruction is triangulated with relative cameras and image correspondences (SIFT-matches) from an initial camera pair with a sufficiently large baseline. Outliers are removed as well as 10% of the 3D-points being the furthest away from the center of gravity of the point cloud. The initial 3D-reconstruction is not rotated nor centered because the canonical camera in the common coordinate frame is set to be camera P_i of the initial pair. Hence, rotation and centering of the 3D-points are not needed. The pixel-threshold for computing inliers of the initial essential matrix is set to three times larger than the threshold of recommendation.

2D-3D correspondences are established for each camera by matching descriptors corresponding to the 3D-points and the descriptors for the image points obtained from SIFT.

Camera translations are robustly estimated in `estimate_T_robust` in a RANSAC loop one by one for a minimal sample of two 2D-3D correspondences by solving $\mathbf{T}_i$ in eq. 2. Here, x and X is a 2D-3D correspondence. Stacking these equations for two 2D-3D correspondences yields four equations and three unknowns. $\mathbf{T}_i$ is then solved using least squares.

$$\begin{pmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -x_2 \end{pmatrix} \mathbf{T} = \begin{pmatrix} \tilde{X}_3 x_1 - \tilde{X}_1 \\ \tilde{X}_3 x_2 - \tilde{X}_2 \end{pmatrix}, \text{ where } \tilde{X} = RX \tag{2}$$

3D points are projected onto the image plane using the absolute rotation and an estimate of the translation vector, and the 2D-3D inliers are measured using point-to-point distance between the projected points and the corresponding image points. The pixel-threshold for computing inliers here is set to three times larger than the threshold of recommendation.

Initially, the translations were estimated using a 2-point method obtained by simplifying `estimate_P_DLT` from assignment 2, which involved solving $\mathbf{T}_i$ using SVD. Unfortunately, this method did not result in inliers for all cameras even with extremely large values for the pixel-threshold. At times, some cameras only got one inlier, and these initial estimates turned out to never be good enough for the bundle adjustment to yield satisfactory final 3D-reconstructions. Instead, solving $\mathbf{T}_i$ from eq. 2 for a minimal sample of two using least squares resulted in significantly more inliers for all cameras and satisfying 3D-reconstructions.

Camera refinement

The absolute rotations and translations are refined by minimizing the squared reprojection error using Levenberg-Marquardt (LM) where the rotations are parametrized as quaternions when being optimized.

A modified version of LM from assignment 4 was first implemented for this where the rotations and the translations are optimized and the 3D-points assumed fixed. The rotations were first parametrized to axis-angle representation but doing this was infeasible. The rotations of the cameras are so small for some data sets that the angle representation is practically zero resulting in singularity in the parametrization. Instead, the rotations are parametrized as quaternions which avoids this issue.

The updates $\delta\mathbf{T}_i$ and $\delta\mathbf{q}_i$ are computed using the jacobian of the residuals. The jacobian of the residual with respect to the translation parameters are analytically calculated as

$$\frac{\partial \mathbf{r}}{\partial \mathbf{T}_i^1} = \begin{bmatrix} \frac{-1}{R_i^3\mathbf{X}_j+\mathbf{T}_i^3} \\ \mathbf{0} \end{bmatrix}, \frac{\partial \mathbf{r}}{\partial \mathbf{T}_i^2} = \begin{bmatrix} \mathbf{0} \\ \frac{-1}{R_i^3\mathbf{X}_j+\mathbf{T}_i^3} \end{bmatrix}, \frac{\partial \mathbf{r}}{\partial \mathbf{T}_i^3} = \begin{bmatrix} \frac{R_i^1\mathbf{X}_j+\mathbf{T}_i^1}{(R_i^3\mathbf{X}_j+\mathbf{T}_i^3)^2} \\ \\ \frac{R_i^2\mathbf{X}_j+\mathbf{T}_i^2}{(R_i^3\mathbf{X}_j+\mathbf{T}_i^3)^2} \end{bmatrix} \tag{3}$$

The jacobian of the residual with respect to the quaternion parameters are tedious to write here but are implemented in `compute_jacobian_of_residual_wrt_q` in `computer_vision.py`.

The cameras are optimized one by one and LA is terminated when the reprojection error is "too close" to the reprojection error of the previous iteration. However, the cameras that are returned from the optimization are essentially the same as before. One fix that could potentially improve the implementation is to optimize over all parameters of all cameras simultaneously in order to increase the reprojection error and give LA more to work with, so to speak. It still remains unclear why this implementation does not work as expected, though. For further details about this implementation see `optimize_T_and_R` in `computer_vision.py`.

Resorting to other options, LM is instead implemented using Scipy's least squares module where LM is an available method. Refining all cameras at once using this module is convenient and it turns out that this yields better 3D-reconstructions. The 3D-reconstructions from each adjacent camera pair aligns closely with each other with slight offset, which is reasonable since this is not a full scale bundle adjustment.

Final 3D-reconstruction

Two final 3D-reconstructions are obtained by accumulating triangulated 3D-points using cameras and image correspondences for each adjacent camera pair. The first showing the reconstruction of the un-refined camera pairs, and the other showing the reconstruction of the refined camera pairs.

Running the software

The modules that needs to be installed to run this software are

- `argparse`
- `matplotlib`
- `numpy`
- `opencv-python`
- `scipy`
- `tqdm`

The files of the actual implementation are

- `computer_vision.py`
- `get_dataset_info.py`
- `main.py`
- `pipeline.py`

which all need to be in the working directory together with a `data` folder containing numerated sub-folders with data sets. `computer_vision.py` contains more general functions for computer vision tasks such as robust estimations and LM, and `pipeline.py` contains larger functions for the specific steps of the project such as rotation averaging, translation registration, etc.

Run the software with `main.py -dataset=<dataset>`, where `-dataset` is required and takes an integer and searches for the specified data set in the `data` folder.

Reconstruction of the data sets

Figures 1-14 show reconstructions of the data sets before and after LM optimization. The 3D-point clouds obtained from triangulation from all adjacent camera pairs have different colors to differentiate them. The reconstructions look pretty good before optimization, and after optimization we can see that LM has improved the solutions. The pixel-thresholds for all data sets were set to what was mentioned previously.

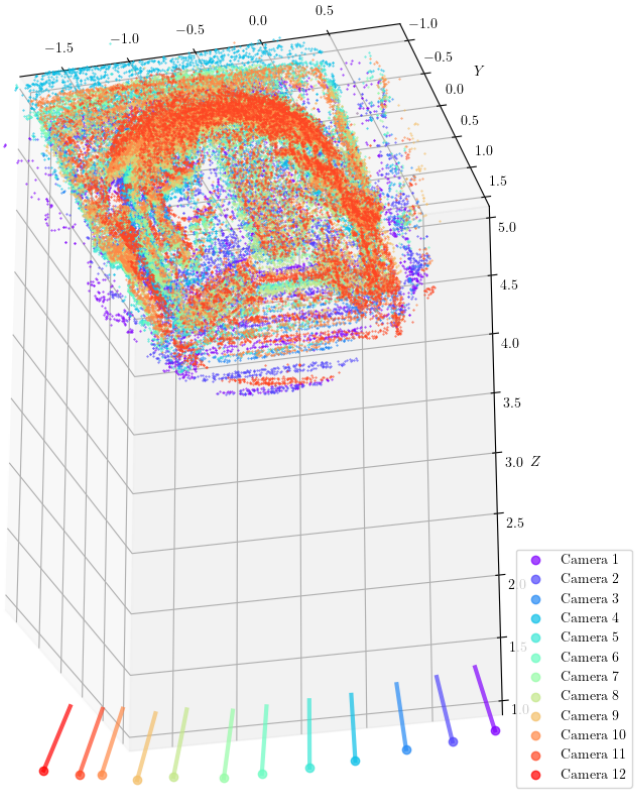

Figure 1: Reconstruction of data set 3 before LM.

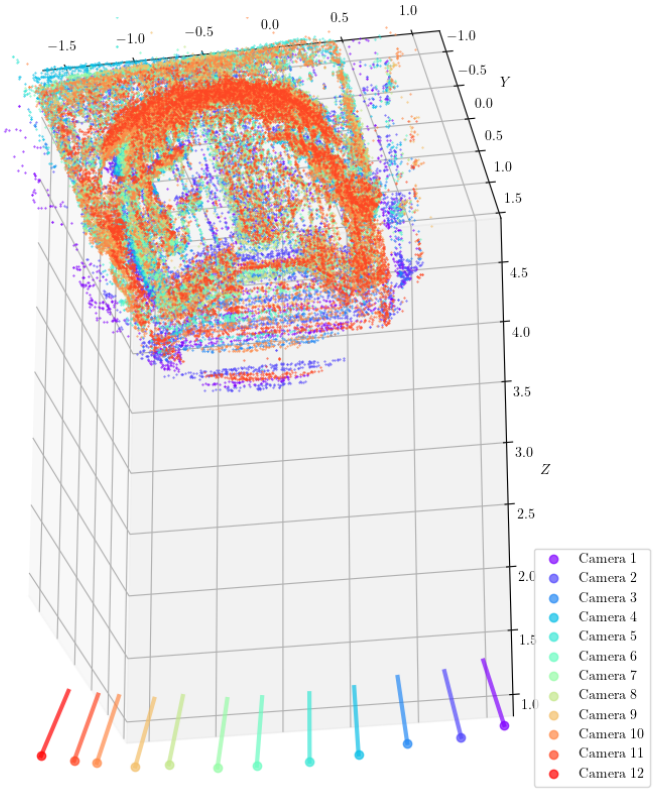

Figure 2: Reconstruction of data set 3 after LM.

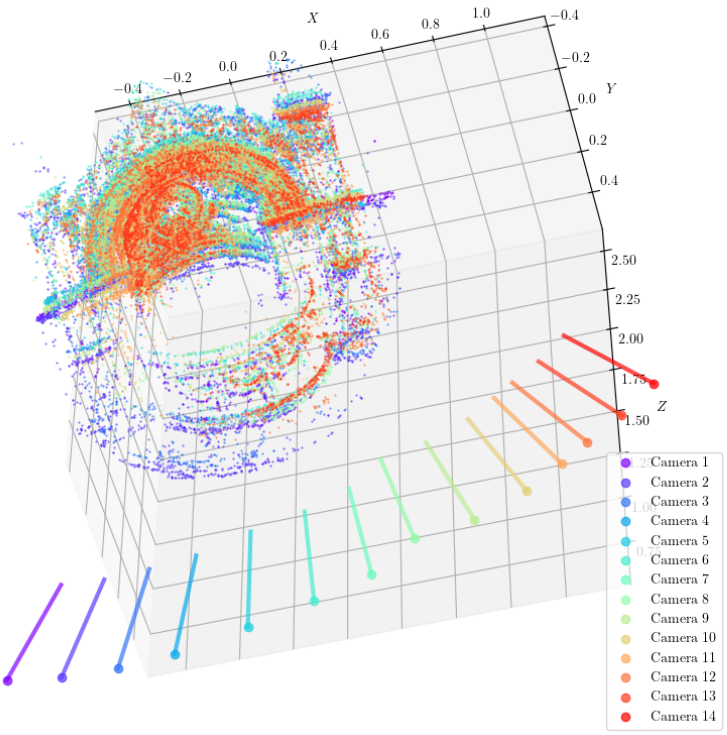

Figure 3: Reconstruction of data set 4 before LM.

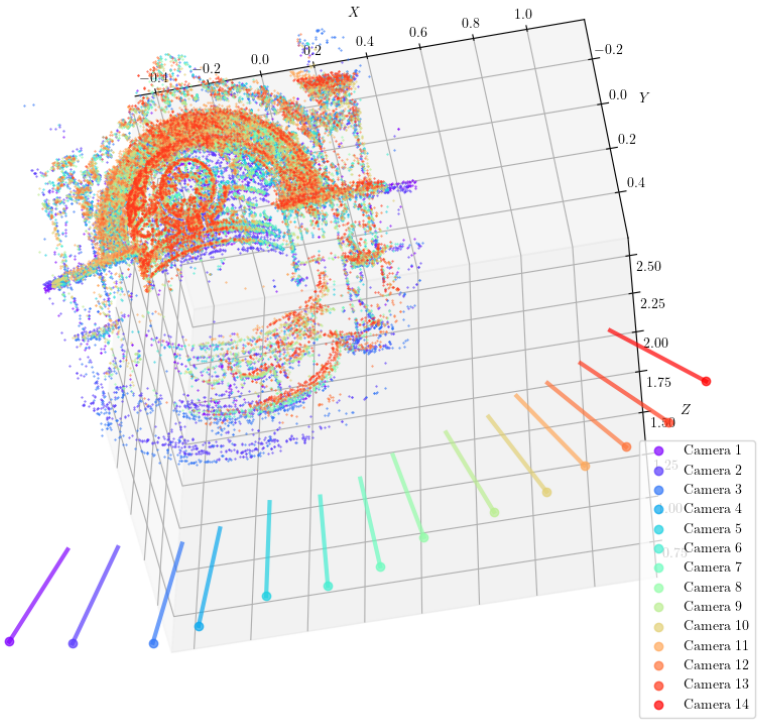

Figure 4: Reconstruction of data set 4 after LM.

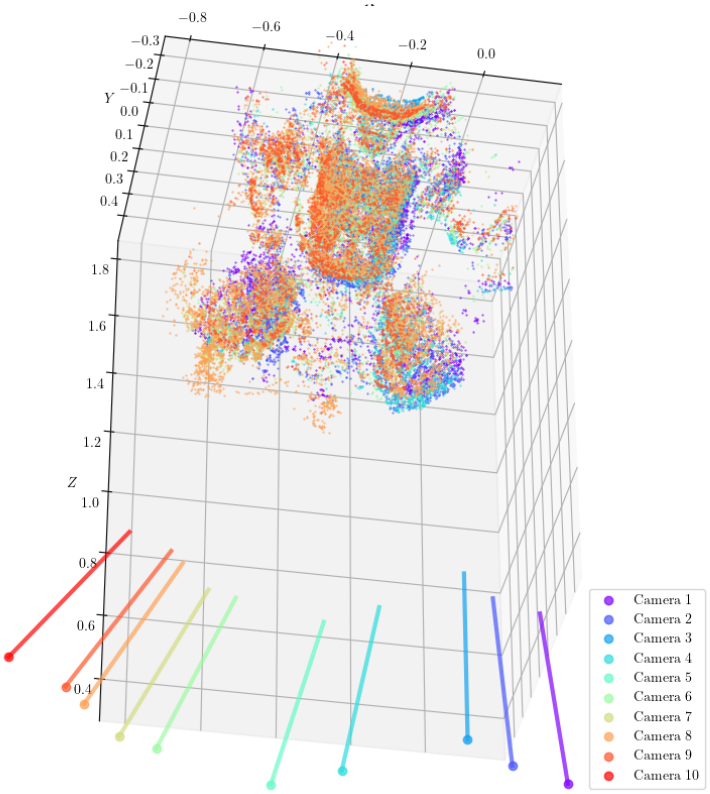

Figure 5: Reconstruction of data set 5 before LM.

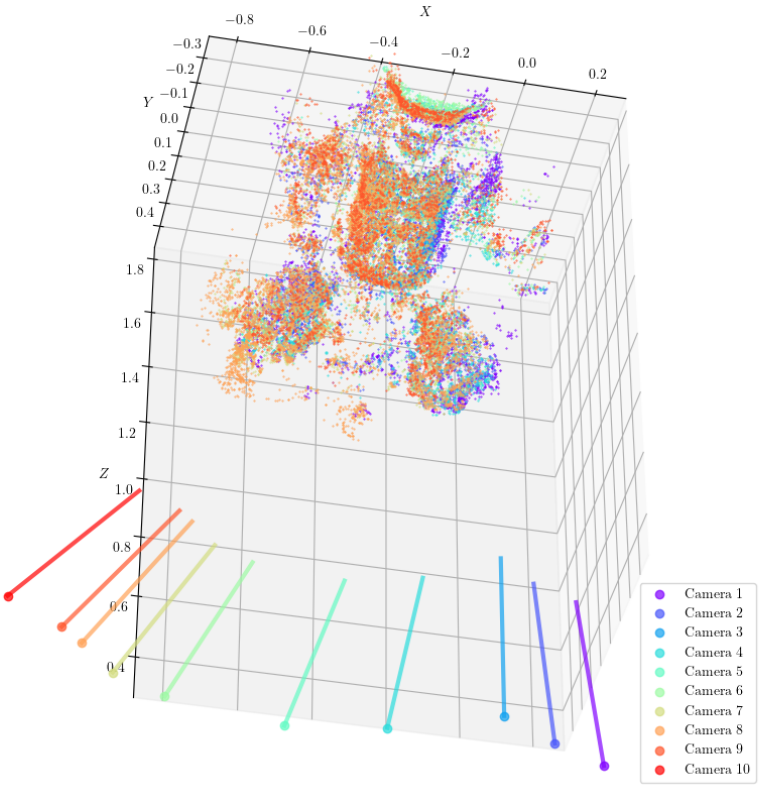

Figure 6: Reconstruction of data set 5 after LM.

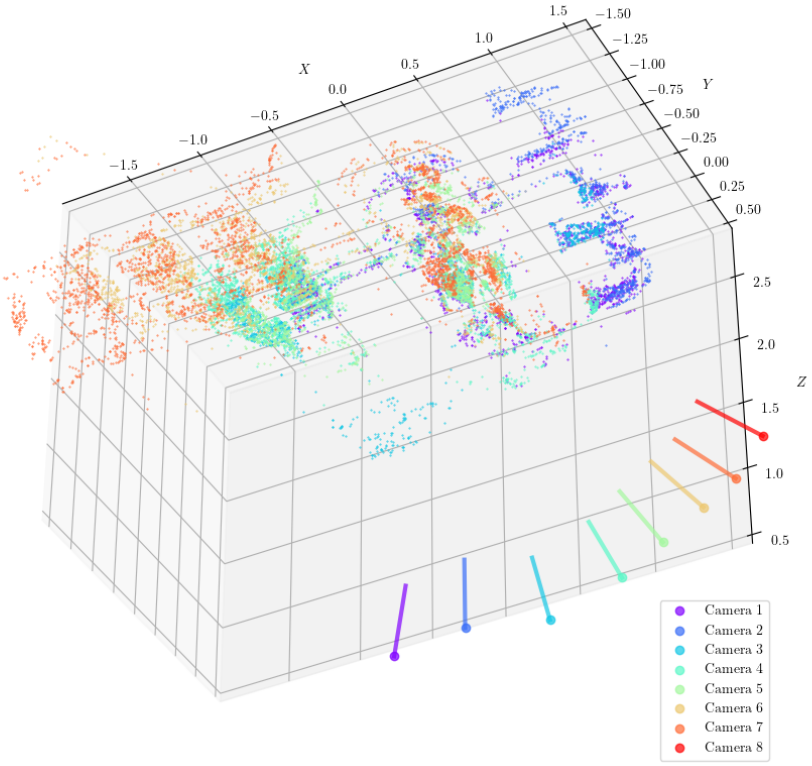

Figure 7: Reconstruction of data set 6 before LM.

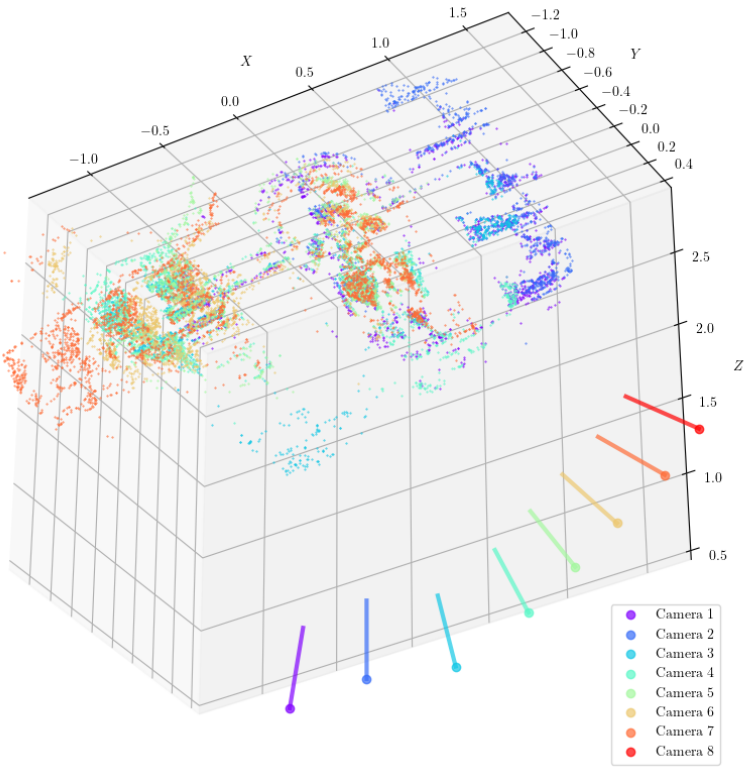

Figure 8: Reconstruction of data set 6 after LM.

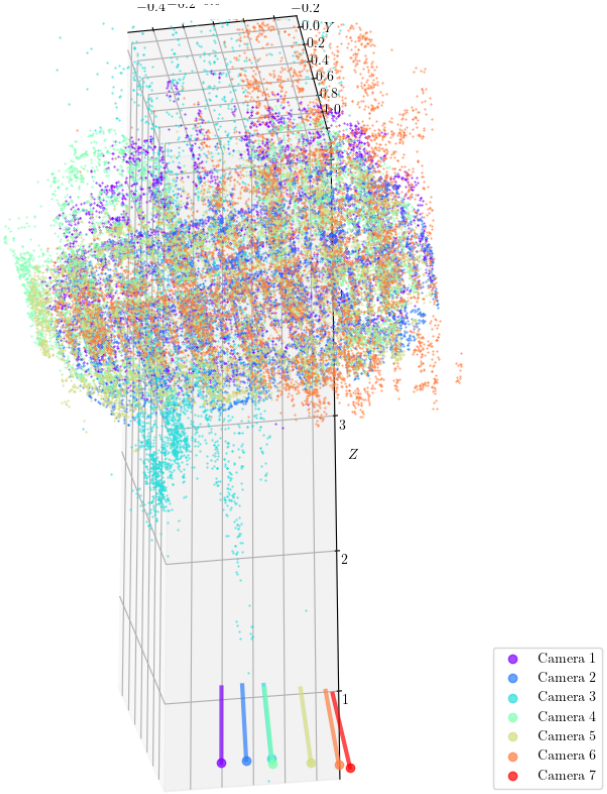

Figure 9: Reconstruction of data set 7 before LM.

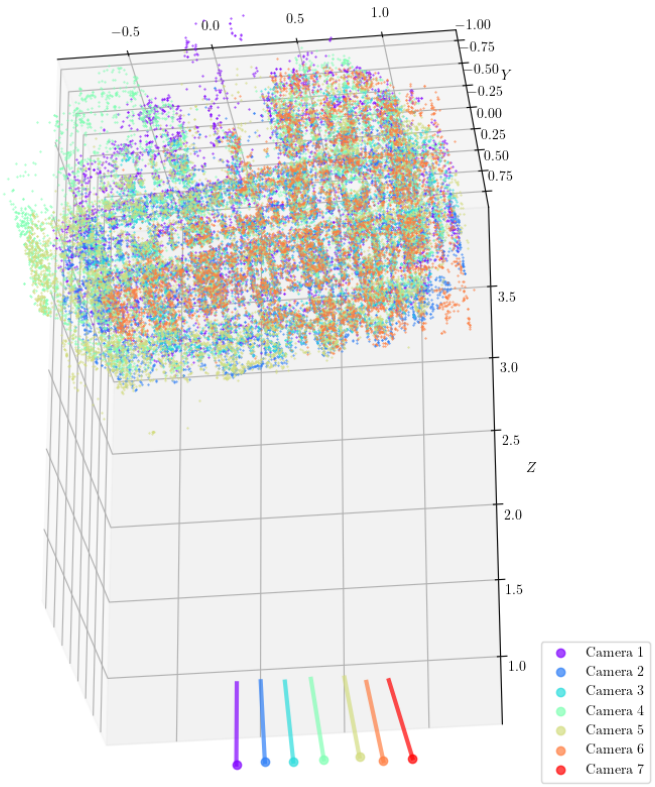

Figure 10: Reconstruction of data set 7 after LM.

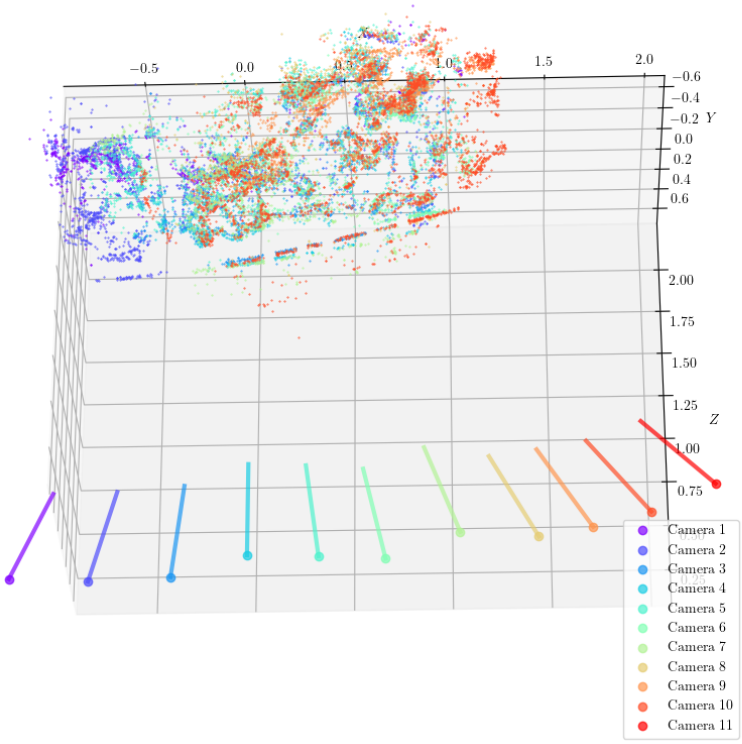

Figure 11: Reconstruction of data set 8 before LM.

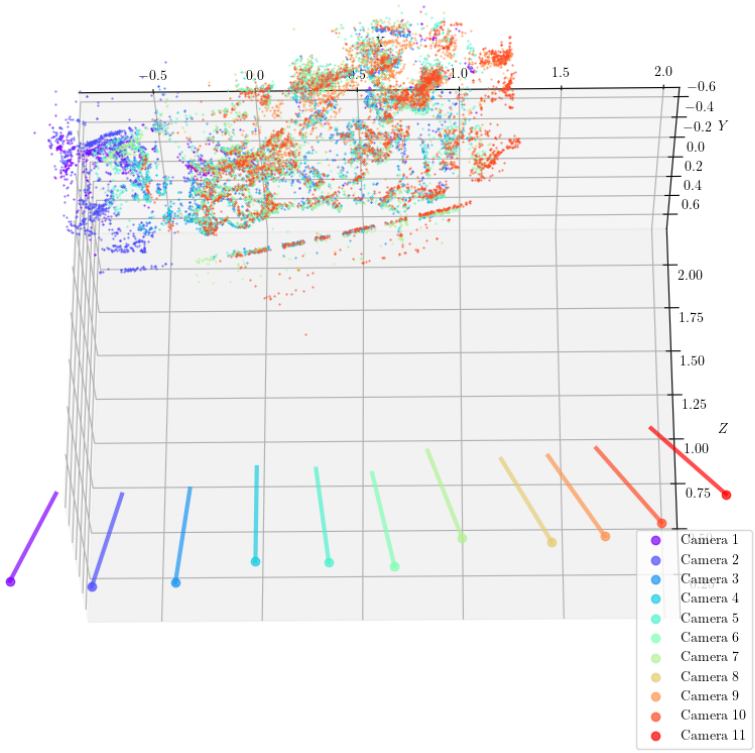

Figure 12: Reconstruction of data set 8 after LM.

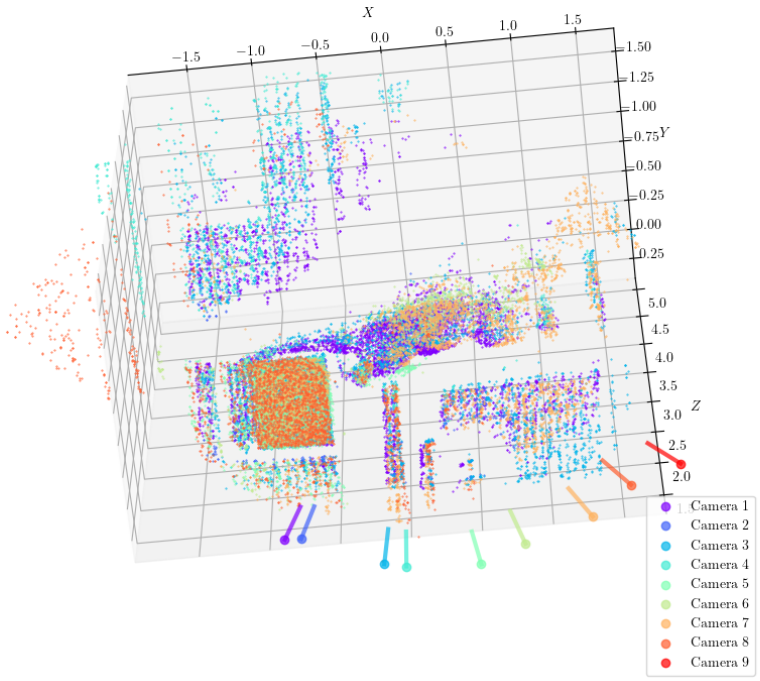

Figure 13: Reconstruction of data set 9 before LM.

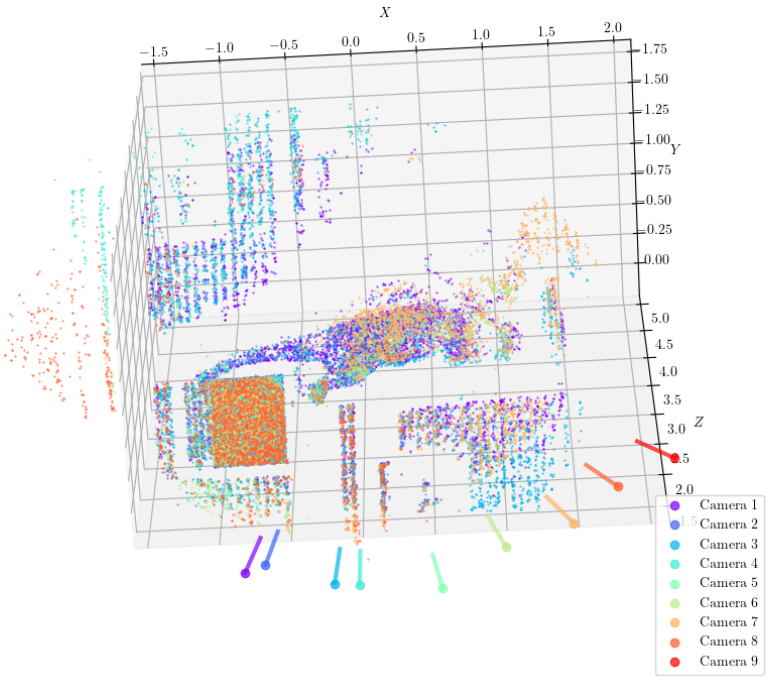

Figure 14: Reconstruction of data set 9 after LM.