MCEN5228 Project 3 - Structure From Motion

Anh Thy Thi Nguyen
University of Colorado Boulder
Email: anng8974@colorado.edu

Thanushraam Suresh Kumar
University of Colorado Boulder
Email: thsu9525@colorado.edu

Using 6 Late Days

***Abstract*—This project focuses on the implementation of Structure from Motion (SfM), a technique widely used in computer vision to reconstruct 3D structures from a sequence of 2D images. The methodology involves key steps such as feature detection, feature matching across images, and camera pose estimation. By leveraging these techniques, the project generates a cohesive 3D structure that accurately represents the captured scene. The results demonstrate the effectiveness of SfM in reconstructing 3D geometry from minimal image inputs, highlighting its potential applications in fields such as robotics, photogrammetry, and augmented reality.**

I. Introduction

Structure from Motion (SfM) is a powerful technique in computer vision that allows for the reconstruction of three-dimensional (3D) structures from a series of two-dimensional (2D) images. By analyzing the motion of objects or the movement of a camera through a scene, SfM provides a computational framework to derive spatial geometry, camera poses, and dense 3D point clouds. This capability has revolutionized fields such as robotics, augmented reality, cultural heritage preservation, and geospatial mapping.

Dataset

The dataset provided consists of six images of a building in front of Levine Hall at the University of Pennsylvania. These images were captured using a GoPro Hero 3 camera, with fisheye lens distortion corrected. Additionally, keypoint matching data, derived from SIFT keypoints and descriptors, is included in the same folder for pairs of images. This data is organized into five files named matching*.txt, where * represents numbers from 1 to 5. For instance, matching3.txt contains the matching data for the third image with the fourth, fifth, and sixth images. Note that there is no matching6.txt file since it would correspond to matches of an image with itself.

The matching files are formatted to represent the matches for each corresponding image. Each file begins with the number of features detected in the image, denoted as nFeatures. Following this, each subsequent row provides details for individual features, including the number of matches for that feature, its color values (Red, Green, and Blue), its location within the current image, and the corresponding matching points in other images. Each row specifies the feature information in the format: (Number of matches for the feature) (Red value) (Green value) (Blue value) (x, y location in the current image), followed by (Image ID) and (x, y location in the matching image) for each match.

Part 1

II. Feature Matching

The first step in this project involves performing Feature Matching across the images to establish correspondences necessary for constructing the 3D structure. For this, we utilize SIFT for feature detection and matching, combined with RANSAC to refine the matches. RANSAC is specifically employed to eliminate outlier correspondences, ensuring robust and accurate matching results.

III. Fundamental Matrix and Essential Matrix

A. Fundamental Matrix

The Fundamental Matrix is a key concept in computer vision and epipolar geometry, representing the intrinsic geometric relationship between two views of the same scene captured by different cameras or camera positions. It is a 3x3 matrix that maps points in one image to their corresponding epipolar lines in the other image. Mathematically, if $\mathbf{x}$ and $\mathbf{x}'$ are the homogeneous coordinates of corresponding points in the two images, the Fundamental Matrix $\mathbf{F}$ satisfies the equation:

$$\mathbf{x}'^{\top}\mathbf{F}\mathbf{x} = 0$$

This equation ensures that corresponding points lie on their respective epipolar lines, enforcing the epipolar constraint. The Fundamental Matrix is crucial for tasks such as stereo vision, camera pose estimation, and 3D reconstruction, as it encapsulates the relative orientation and position of the cameras, independent of intrinsic camera parameters. It is typically estimated using matched feature points across the images, often refined with robust techniques like RANSAC to handle outliers.

What we did

We implemented the eight-point algorithm with normalization to compute the Fundamental Matrix F. The algorithm involves normalizing the input points to improve numerical stability, constructing a linear system A based on the epipolar constraint, solving for F using Singular Value Decomposition (SVD), and enforcing the rank-2 constraint to make F valid for epipolar geometry. For outlier rejection, we implemented the RANSAC algorithm. RANSAC iteratively selects 8 random correspondences, computes a candidate F, and evaluates its quality by counting inliers based on a geometric threshold. This approach ensures robustness against outliers in feature matches.
To refine the results, we applied a stricter geometric threshold for the inlier selection step. Finally, we compared the results of our custom implementation with OpenCV's built-in functions to validate accuracy and performance. The comparison included both the computed Fundamental Matrix and the number of inliers detected.

Difficulty

We were able to obtain reasonable set of inliers and fundamental matrix after running GetInlieresRANSAC (See Comparison box and Inlier Image), which performed seemingly well but not as refined as OpenCV results in comparison to the expected results. Our RANSAC results can sometimes be unstable, which sometimes works very well for downstream functions, but sometimes would cause error or infinite loop. The randomness nature of RANSAC can sometimes cause issues and choosing the right threshold can be tricky. We could have been able to make a more complex and dynamic threshold to suit with variety of image pairs but for the scope of the project, we decided to test downstream function with the fundamental matrix result from build in OpenCV so that we can examine later step in the SfM algorithm better without being affected by the GetInliersRANSAC Step

Comparison of Fundamental Matrices and Inlier Counts for Image Pair (1-2)

Custom Implementation (FBest):

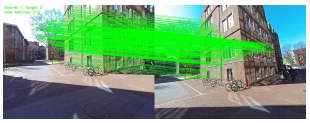
$$\begin{bmatrix} 5.1539 \times 10^{-8} & -1.1826 \times 10^{-5} & 3.6597 \times 10^{-3} \\ 1.0686 \times 10^{-5} & 3.5238 \times 10^{-7} & -5.9900 \times 10^{-3} \\ -4.6506 \times 10^{-3} & 4.9994 \times 10^{-3} & 1.0000 \end{bmatrix}$$

OpenCV Implementation (F_cv2):

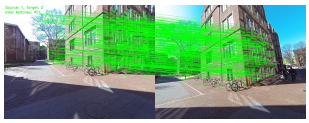
$$\begin{bmatrix} -4.7389 \times 10^{-7} & -1.1817 \times 10^{-5} & 2.9515 \times 10^{-3} \\ 1.5310 \times 10^{-5} & -1.3051 \times 10^{-6} & -2.9537 \times 10^{-3} \\ -5.0818 \times 10^{-3} & 3.5190 \times 10^{-4} & 1.0000 \end{bmatrix}$$

Comparison of Inlier Counts:

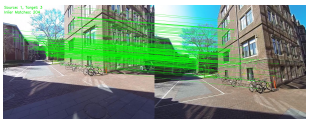
- Custom Implementation Inliers: 572
- OpenCV Implementation Inliers: 451



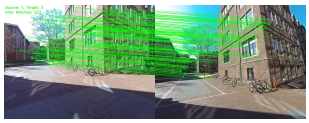
(a) Our Implementation for 1-2 Inliers



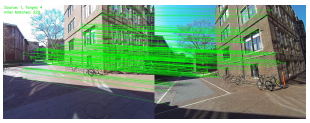
(b) OpenCV Implementation for 1-2 Inliers



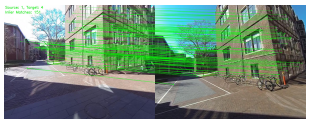
(a) Our Implementation for 1-3 Inliers



(b) OpenCV Implementation for 1-3 Inliers



(a) Our Implementation for 1-4 Inliers



(b) OpenCV Implementation for 1-4 Inliers

B. Essential Matrix

The Essential Matrix is a fundamental concept in computer vision, particularly in epipolar geometry, that describes the relationship between two calibrated camera views of the same scene. It is a 3×3 matrix that encodes the relative rotation and translation between the two cameras, assuming the cameras are calibrated (i.e., their intrinsic parameters are known).

Mathematically, if $\mathbf{x}$ and $\mathbf{x}'$ are the normalized homogeneous coordinates of corresponding points in the two images, the Essential Matrix $\mathbf{E}$ satisfies the equation:

$$\mathbf{x}'^{\top} \mathbf{E} \mathbf{x} = 0$$

This equation ensures that corresponding points in the two images satisfy the epipolar constraint. The Essential Matrix can be decomposed into the relative rotation $\mathbf{R}$ and translation $\mathbf{t}$ between the cameras:

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}$$

where $[\mathbf{t}]_\times$ is the skew-symmetric matrix representation of the translation vector $\mathbf{t}$.

What we did

We implemented a function to compute the Essential Matrix (E) from the Fundamental Matrix (F) using the camera intrinsic matrix (K). The Essential Matrix was calculated as:

$$\mathbf{E} = \mathbf{K}^\top \mathbf{F} \mathbf{K}$$

We then applied Singular Value Decomposition (SVD) to $\mathbf{E}$ to enforce the constraints of a valid Essential Matrix, ensuring it has two singular values of 1 and one singular value of 0. This correction ensures $\mathbf{E}$ is consistent with the epipolar geometry of calibrated cameras. The corrected Essential Matrix was returned for further camera pose estimation.

Difficulty

There were no major difficulties in implementing the Essential Matrix computation. The process was straightforward, as it primarily involved matrix multiplication and applying the SVD-based correction to enforce the necessary constraints.

Comparison of Essential Matrices for Image Pair (1-2)

Essential Matrix (With Our F):

$$\begin{bmatrix} 0.02174057 & -0.90281856 & -0.26005648 \\ 0.98397646 & 0.00495183 & 0.1621886 \\ 0.08554323 & -0.32897191 & -0.08207996 \end{bmatrix}$$

Essential Matrix (With Open CV F):

$$\begin{bmatrix} -0.02024298 & -0.61320295 & -0.27045225 \\ 0.79948729 & -0.07664602 & 0.57190476 \\ 0.18240537 & -0.72027465 & -0.16196006 \end{bmatrix}$$

IV. Linear Triangulation and Camera Pose Estimation

Linear triangulation and camera pose estimation are crucial steps in structure-from-motion pipelines, enabling the recovery of 3D points and camera poses from two views of the same scene.

A. Camera Pose Estimation

Camera pose estimation involves determining the relative rotation ($\mathbf{R}$) and translation ($\mathbf{t}$) between two cameras from the Essential Matrix ($\mathbf{E}$). The Essential Matrix encodes the geometric relationship between corresponding points in two calibrated views.

Using Singular Value Decomposition (SVD), the Essential Matrix is decomposed as:

$$\mathbf{E} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

where $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}^T$ are the components of the decomposition. From this, two possible rotation matrices ($\mathbf{R}_1, \mathbf{R}_2$) and two possible translations ($\mathbf{t}, -\mathbf{t}$) can be derived. This results in four potential camera poses, each representing a candidate for the relative orientation and position of the cameras.

B. Linear Triangulation

Linear triangulation reconstructs 3D points from corresponding 2D points in two images. Given the camera projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$, the relationship between a 3D point $\mathbf{X}$ and its image points $\mathbf{x}_1$ and $\mathbf{x}_2$ is expressed as:

$$\mathbf{x}_i = \mathbf{P}_i \mathbf{X}, \quad i = 1, 2$$

where $\mathbf{x}_i$ represents the image coordinates in homogeneous form.

The triangulation process involves constructing a linear system from these equations:

$$\mathbf{A}\mathbf{X} = 0$$

where $\mathbf{A}$ is derived from the projection matrices and the image points. Solving this system using SVD yields $\mathbf{X}$, the homogeneous coordinates of the 3D point.

Linear triangulation is a critical step in verifying the camera poses and reconstructing the scene geometry, providing a foundation for further refinement steps like non-linear optimization.

What we did

We implemented camera pose extraction and linear triangulation for 3D reconstruction as follows:

Camera Pose Estimation

1) **Singular Value Decomposition (SVD):** Decomposed the Essential Matrix (E) to obtain U, S, and $V^\top$.
2) **Constructed Rotation Matrices:** Used the predefined rotation matrix W to compute two possible rotations:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Resulting in:

$$R_1 = UWV^\top, \quad R_2 = UW^\top V^\top$$

3) **Computed Translations:** Extracted two possible translations from the third column of U:

$$C_1 = U[:,2], \quad C_2 = -U[:,2]$$

4) **Validated Rotation Matrices:** Ensured that the determinant of each rotation matrix is positive ($\det(R) = +1$) by inverting R and C if necessary.
5) **Generated Possible Camera Poses:** Combined two rotations (R_1, R_2) with two translations (C_1, C_2) to produce four candidate poses.

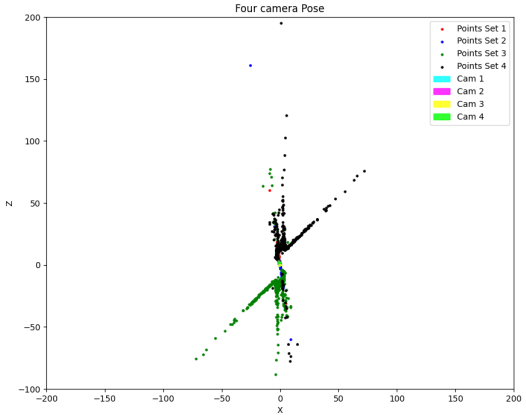
Linear Triangulation

1) **Projection Matrices:** Constructed the camera projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ for the two views:

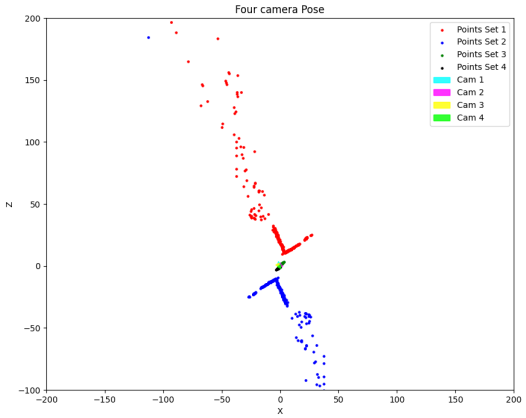
$$P_1 = KR_0[I|-C_0], \quad P_2 = KR[I|-C]$$

where K is the intrinsic camera matrix, R and C represent rotation and translation for each view, and I is the identity matrix.
2) **Linear System for Triangulation:** For each pair of corresponding 2D points ($\mathbf{x}_1, \mathbf{x}_2$), formulated the linear system $\mathbf{AX} = 0$ based on the projection matrices and image points.
3) **Solved for 3D Points:** Solved the system using eigen decomposition of $\mathbf{A}^T\mathbf{A}$ to find the 3D point $\mathbf{X}$ in homogeneous coordinates.
4) **Normalized Results:** Converted the 3D points to inhomogeneous coordinates by dividing by the last entry of $\mathbf{X}$.

This implementation provided us with four possible camera poses and a set of triangulated 3D points for further verification and refinement.



(a) Plot of 4 Camera Poses with Our F and Inliers



(b) Plot of 4 Camera Poses with OpenCV F and Inliers

Difficulty

Linear Triangulation is a bit problematic because sometimes we have extreme outliers. Maybe we can refine the output more by removing points that are too far from others. Camera Pose implementation was relatively straightforward. Ensuring that the determinant of R was positive required an additional step to validate R as a valid rotation matrix. Otherwise, no significant challenges were encountered.

Evaluation

The resulting plot from 4 Camera Pose with Open CV F looks exactly like the results given in the project Write up so we are confident that our implementation is correct.

V. Chierality Condition and Non-Linear Triangulation

Cheirality Condition in Triangulation

The **cheirality condition** is a fundamental check in 3D reconstruction to ensure that triangulated points lie in front of both cameras. Since real-world objects cannot exist behind a camera, this condition is used to select the correct camera pose configuration during triangulation, especially when multiple poses are possible (e.g., from the decomposition of the Essential Matrix).

For a triangulated 3D point $\mathbf{X}$, the depth in the camera's frame must satisfy the condition:

$$(\mathbf{R}_i\mathbf{X} + \mathbf{t}_i)_z > 0$$

for both cameras, where $\mathbf{R}_i$ and $\mathbf{t}_i$ are the rotation and translation of the i-th camera. This ensures that the reconstructed 3D point is physically plausible and visible in both views. Any configuration that fails this condition is discarded.

Non-Linear Triangulation

Non-linear triangulation is an optimization-based method used to refine the 3D coordinates of points by minimizing reprojection errors. Unlike linear triangulation, which provides a direct but potentially less accurate solution, non-linear triangulation iteratively adjusts the 3D points to achieve the best fit in image space.

The reprojection error is defined as the difference between the observed image points and the projected 2D points derived from the triangulated 3D coordinates. The below is reprojection error calcuation formula which we used in this project

$$\min_{\mathbf{x}} \sum_{j=1,2} \left(w^j - \frac{\mathbf{P}_1^{j\top}\tilde{\mathbf{X}}}{\mathbf{P}_3^{j\top}\tilde{\mathbf{X}}} \right)^2 + \left(v^j - \frac{\mathbf{P}_2^{j\top}\tilde{\mathbf{X}}}{\mathbf{P}_3^{j\top}\tilde{\mathbf{X}}} \right)^2$$

where j is the index of each camera, $\tilde{\mathbf{X}}$ is the homogeneous representation of the 3D point, and $\mathbf{P}_1^j$, $\mathbf{P}_2^j$, and $\mathbf{P}_3^j$ represent the rows of the camera projection matrix $\mathbf{P}^j$. Non-linear triangulation is particularly effective in handling noisy data and when precise accuracy is required, such as in applications like visual SLAM, 3D mapping, and augmented reality.

What we did

We implemented the disambiguation of camera poses and refined the 3D points using non-linear triangulation as follows:

Disambiguation of Camera Pose

1) **Input:** Four candidate camera poses derived from the Essential Matrix.
2) **Cheirality Condition:** Evaluated each candidate pose by checking the positive depth of triangulated 3D points in both the candidate and reference camera frames.
3) **Selection of Pose:** Counted the number of points satisfying the cheirality condition for each pose and selected the pose with the maximum count as the correct pose.
4) **Validation:** Further refined the selected pose by filtering out 3D points that failed the cheirality condition.

Non-Linear Triangulation

1) **Initial 3D Points:** Used linearly triangulated 3D points as the starting estimates.
2) **Reprojection Error Minimization:** Optimized the 3D points to minimize reprojection error using `scipy.optimize.least_squares`.
3) **Camera Projection Matrices:** Constructed projection matrices for the reference and selected cameras.
4) **Optimization:** Refined the 3D points by iteratively reducing the reprojection error across both views, ensuring higher accuracy.

Difficulty

The implementation of the disambiguation step was straightforward, with the primary challenge being the efficient evaluation of the cheirality condition for large datasets. Non-linear triangulation required careful setup of the optimization problem, including proper construction of projection matrices and handling outliers. However, both steps were manageable and yielded reliable results.

Comparison of Results and Evaluation

Selected Camera Pose and Parameters:

- **With OpenCV Data:**
  - Selected Pose Index: 0
  - Camera Center (C):

$$\begin{bmatrix} -0.7419 \\ -0.1669 \\ 0.6494 \end{bmatrix}$$

  - Rotation Matrix (R):

$$\begin{bmatrix} 0.974 & 0.064 & -0.218 \\ -0.027 & 0.985 & 0.169 \\ 0.225 & -0.159 & 0.961 \end{bmatrix}$$

- **With Our Data:**
  - Selected Pose Index: 3
  - Camera Center (C):

$$\begin{bmatrix} -0.3418 \\ -0.0739 \\ 0.9369 \end{bmatrix}$$

  - Rotation Matrix (R):

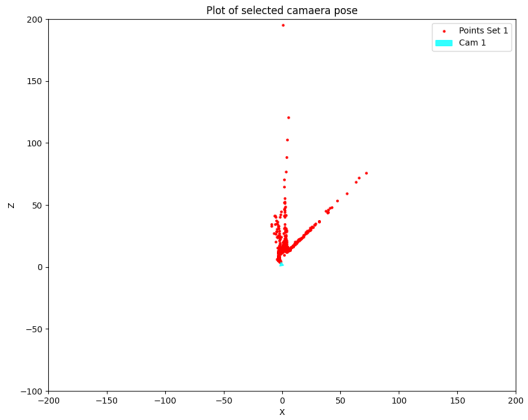
$$\begin{bmatrix} 0.981 & 0.075 & -0.178 \\ -0.038 & 0.979 & 0.202 \\ 0.190 & -0.191 & 0.963 \end{bmatrix}$$

Reprojection Error Comparison:

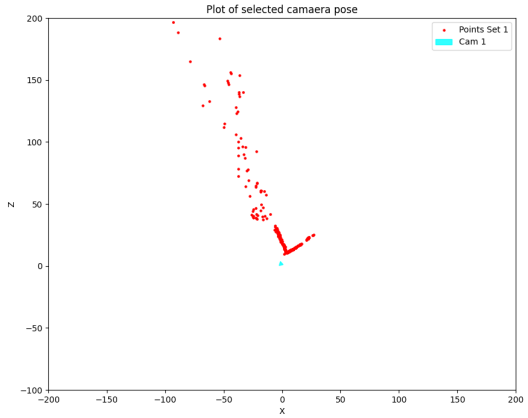
- **With Our F:**
  - Linear Triangulation Error: 5.4613
  - Non-linear Triangulation Error: 5.3851
- **With OpenCV F:**
  - Linear Triangulation Error: 4.0059
  - Non-linear Triangulation Error: 3.9843

Evaluation:

- The selected pose with OpenCV data matched the expected results provided in the project description, indicating correctness.
- The pose selected using our implementation also aligned well with the scene geometry, demonstrating the robustness of our approach.
- Non-linear triangulation consistently reduced reprojection error compared to linear triangulation, confirming its effectiveness in refining 3D point estimates.
- Reprojection errors with OpenCV's Fundamental Matrix were lower compared to our custom implementation, highlighting the benefits of using a more refined input.



(a) Plot of 4 Camera Poses with Our F and Inliers



(b) Plot of 4 Camera Poses with OpenCV F and Inliers

VI. Perspective-n-Points

A. Perspective-n-Points

The Perspective-n-Points (PnP) problem involves estimating the pose (position and orientation) of a calibrated camera given a set of $\mathbf{n}$ 3D points in the world and their corresponding 2D projections in the image. The goal is to find the camera's rotation and translation that align the 3D points with their 2D counterparts. PnP is widely used in computer vision applications like augmented reality and visual SLAM, as it directly relates 2D-3D correspondences to the camera pose.

B. Linear Camera Pose Estimation

To estimate the camera pose using linear least squares, given 2D-3D correspondences ($\mathbf{X} \leftrightarrow \mathbf{x}$) and the intrinsic parameters ($\mathbf{K}$), the 2D points are first normalized by the intrinsic parameters to isolate the camera parameters. This normalization is expressed as:

$$\mathbf{x}' = \mathbf{K}^{-1}\mathbf{x}$$

The relationship between the 3D points and normalized 2D points can then be formulated and solved using linear least squares to determine the camera center ($\mathbf{C}$) and the rotation matrix ($\mathbf{R}$), where the translation vector $\mathbf{t}$ is computed as:

$$\mathbf{t} = -\mathbf{R}^{\top}\mathbf{C}$$

Since the linear least squares solution does not inherently enforce the orthogonality of the rotation matrix ($\mathbf{R} \in SO(3)$), the rotation matrix must be corrected. This is done using singular value decomposition (SVD), such that:

$$\mathbf{R} = \mathbf{U}\mathbf{V}^\top$$

If the determinant of the corrected rotation matrix is -1, it is adjusted to ensure a proper rotation matrix as:

$$\mathbf{R} = -\mathbf{R}$$

Linear PnP requires at least six 2D-3D correspondences to solve for the camera pose parameters. This minimum number is necessary to provide sufficient constraints for resolving the degrees of freedom in the system.

C. PnP RANSAC

PnP-RANSAC is a robust extension of the standard PnP algorithm. It incorporates the RANSAC (Random Sample Consensus) framework to handle outliers in 2D-3D correspondences. By iteratively selecting subsets of correspondences, estimating the pose, and evaluating the number of inliers, PnP-RANSAC ensures that the final pose is robust to noise and mismatches. This makes it highly effective in real-world applications where data may be noisy or contain incorrect matches.

D. Nonlinear PnP

Given $N \geq 6$ 3D-2D correspondences ($\mathbf{X} \leftrightarrow \mathbf{x}$) and an initial camera pose estimate $(\mathbf{C}, \mathbf{R})$ from a linear PnP method, the camera pose can be refined by minimizing the reprojection error:

$$\min_{\mathbf{C},\mathbf{R}} \sum_{j=1}^{J} \left(\left(w^j - \frac{\mathbf{P}_1^\top \tilde{\mathbf{X}}_j}{\mathbf{P}_3^\top \tilde{\mathbf{X}}_j} \right)^2 + \left(v^j - \frac{\mathbf{P}_2^\top \tilde{\mathbf{X}}_j}{\mathbf{P}_3^\top \tilde{\mathbf{X}}_j} \right)^2 \right)$$

Here, $\tilde{\mathbf{X}}_j$ is the homogeneous representation of $\mathbf{X}_j$, and $\mathbf{P} = \mathbf{K}\mathbf{R}[\mathbf{I}_{3\times 3} \mid -\mathbf{C}]$. To ensure $\mathbf{R} \in SO(3)$, quaternions $\mathbf{q}$ can be used, reformulating the optimization as:

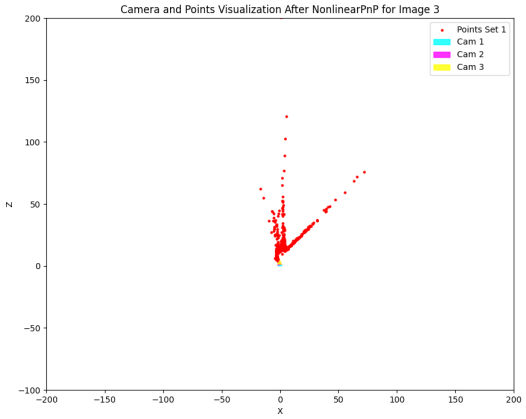
$$\min_{\mathbf{C},\mathbf{q}} \sum_{j=1}^{J} \left(\left(w^j - \frac{\mathbf{P}_1^\top \tilde{\mathbf{X}}_j}{\mathbf{P}_3^\top \tilde{\mathbf{X}}_j} \right)^2 + \left(v^j - \frac{\mathbf{P}_2^\top \tilde{\mathbf{X}}_j}{\mathbf{P}_3^\top \tilde{\mathbf{X}}_j} \right)^2 \right)$$

We have used nonlinear optimization function `scipy.optimize.least_squares` these refines the pose iteratively, starting from the initial guess provided by linear PnP.

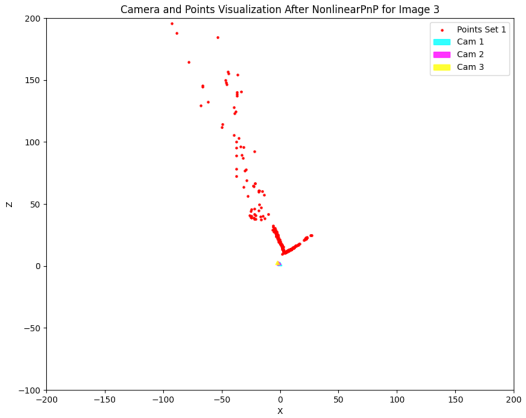
VII. Bundle Adjustment

A. Bundle Adjustment

Bundle Adjustment (BA) is an optimization technique used in computer vision to refine the 3D structure and camera poses simultaneously. It works by minimizing the reprojection error, which is the difference between the observed 2D image points and the projected 2D points obtained from the estimated 3D structure and camera parameters. BA is widely used in applications like 3D reconstruction, structure-from-motion (SfM), and SLAM to improve the accuracy of results.



(a) PnPRANSAC of Image 3



(b) PnPRANSAC of Image 3

B. Visible Matrix

The visible matrix is a binary matrix used in bundle adjustment to indicate whether a particular 3D point is visible in a specific camera view. Each entry in the matrix corresponds to a point-camera pair, with a value of 1 1 if the 3D point is visible in the camera view and 0 0 otherwise. This matrix helps in organizing and constraining the optimization problem by focusing computations only on observed correspondences.

C. Bundle Adjustment optimization

Bundle Adjustment is an optimization process used to refine initialized camera poses and 3D points by minimizing the reprojection error. This ensures alignment between the observed 2D image points and the projected points obtained from the estimated camera parameters and 3D points. The optimization problem can be formulated as follows:

$$\min_{\{\mathbf{C}_i,\mathbf{q}_i\}_{i=1}^{I},\{\mathbf{X}_j\}_{j=1}^{J}} \sum_{i=1}^{I}\sum_{j=1}^{J} V_{ij} \left(\left(w^j - \frac{\mathbf{P}_1^{i\top}\tilde{\mathbf{X}}_j}{\mathbf{P}_3^{i\top}\tilde{\mathbf{X}}_j} \right)^2 + \left(v^j - \frac{\mathbf{P}_2^{i\top}\tilde{\mathbf{X}}}{\mathbf{P}_3^{i\top}\tilde{\mathbf{X}}} \right)^2 \right)$$

where:

- $\mathbf{C}_i$ and $\mathbf{q}_i$ represent the pose (translation and rotation) of the i-th camera.
- $\mathbf{X}_j$ is the j-th 3D point.
- V_{ij} is the visibility matrix, indicating whether the j-th 3D point is visible in the i-th camera.
- w^j and v^j are the observed 2D coordinates of the j-th point in the image captured by the i-th camera.

The optimization problem is computationally complex and can be slow, especially for larger datasets, as it involves refining both camera parameters and 3D points. While non-linear optimization libraries like `scipy.optimize.leastsq` can solve this problem, they are often inefficient due to the high dimensionality of the parameters.

To address this, Sparse Bundle Adjustment (SBA) toolboxes, such as `pySBA` or large-scale SBA implementations in `scipy`, are used. These toolboxes exploit the sparsity of the visibility matrix V, where most 3D points are visible in only a small subset of images. By leveraging this sparsity, SBA significantly improves computational efficiency, making it suitable for large-scale datasets. Using sparse optimization techniques is highly recommended for practical implementation.

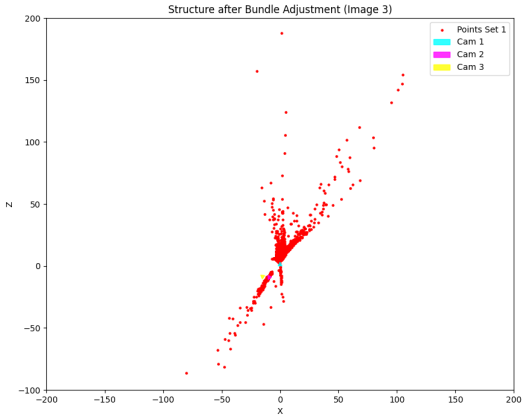
VIII. Results

We will be comparing results from our implementation and along with VisualSfM's output. VisualSfM is a state-of-the-art, graphical tool for performing Structure-from-Motion on sets of unordered images.It integrates robust feature extraction, matching, and other advanced algorithms behind a GUI, making it easy to obtain 3D reconstructions without manually piecing together individual steps. It enables efficient and scalable reconstruction of complex scenes with minimal user intervention.

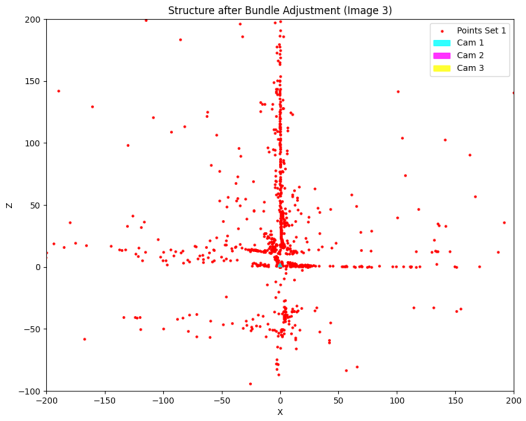
A. Our Results

B. VisualSfM Results

As illustrated in figure 8, the visualization displays both the camera poses and the resulting 3D reconstruction. Additionally, figure 9 highlights the dense 3D point cloud generated from the input images.



(a) Structure after Bundle Adjustment for Image 3



(b) Structure after Bundle Adjustment for Image 3

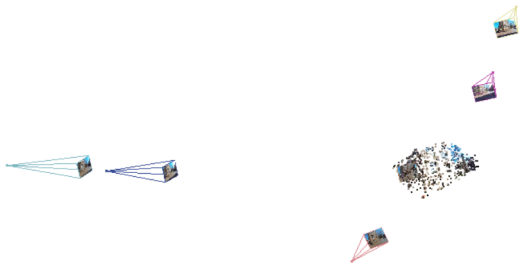


Fig. 8: VisualSfM - 3D Reconstruction Output

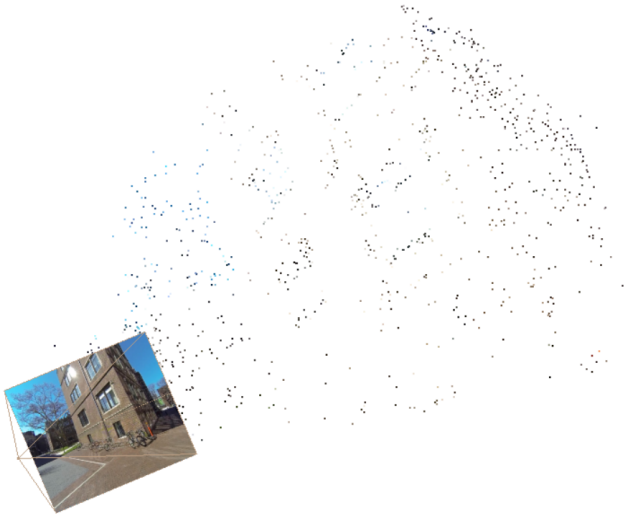


Fig. 9: Another sample output from VisualSfM