Structure From Motion: Reconstructing a 3D Landmark from Images

Muhammad Haseeb
Department of Computer Science
Syed Babar Ali School of Science
Engineering, LUMS
Lahore, Pakistan
26100253@lums.edu.pk

Abstract

This project focuses on implementing a 3D reconstruction pipeline using the Structure from Motion (SfM) technique. SfM is a robust method for reconstructing 3D geometry from 2D images captured from different viewpoints. Key stages include image preprocessing, feature extraction and matching, camera pose estimation, and 3D point triangulation, culminating in the generation of a point cloud and optional mesh reconstruction. OpenCV and Open3D libraries were employed for image processing and 3D visualization. The results demonstrated successful reconstruction, although challenges in feature matching and numerical precision highlighted areas for further improvement. The pipeline is adaptable for integration with advanced photogrammetry and machine learning techniques.

1 Introduction

The objective of this project was to implement a 3D reconstruction pipeline using the Structure from Motion (SfM) technique. SfM is a well-known method for reconstructing the 3D geometry of a scene from a set of 2D images taken from different viewpoints. This project explores key steps in SfM, including image preprocessing, feature extraction, matching, camera pose estimation, and 3D point triangulation, followed by point cloud generation and optional mesh reconstruction. OpenCV and Open3D were used as the primary libraries for image processing and 3D visualization, respectively.

Submitted to Dr. Murtaza Taj

2 Methodology

2.1 Image Preprocessing

Image preprocessing is crucial for ensuring high-quality inputs in the 3D reconstruction pipeline. Two variations were tested to evaluate their impact on feature extraction: one included sharpening and edge detection, while the other excluded these steps. Both pipelines involved resizing images to 600x400 pixels, brightness adjustment, grayscale conversion, and Gaussian blurring to standardize the dataset, reduce computational complexity, and minimize noise.

The first variation enhanced edges and fine details, resulting in a higher number of detected features and improved feature matching. This approach significantly benefited feature detection and the reliability of the reconstruction pipeline, as shown in Figure 1.

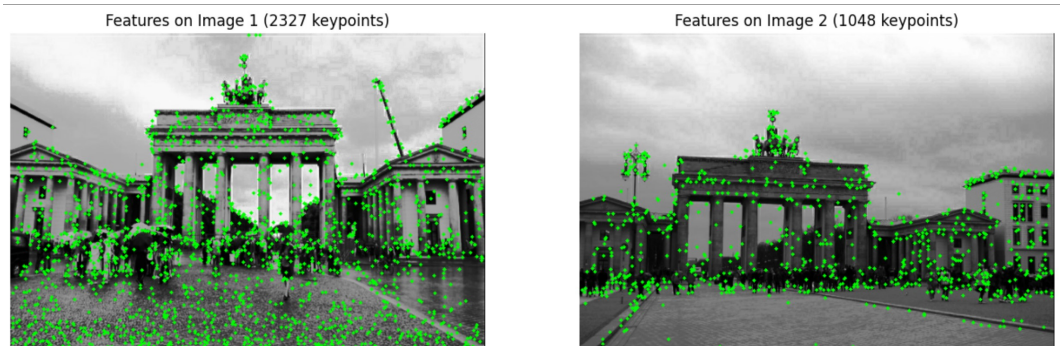

Figure 1: Image with additional preprocessing (sharpening and edge detection).

In contrast, the second variation omitted sharpening and edge detection. Although effective, it detected fewer features, as illustrated in Figure 2. This highlights the value of additional enhancement steps in improving feature extraction and ensuring a more accurate 3D reconstruction process.

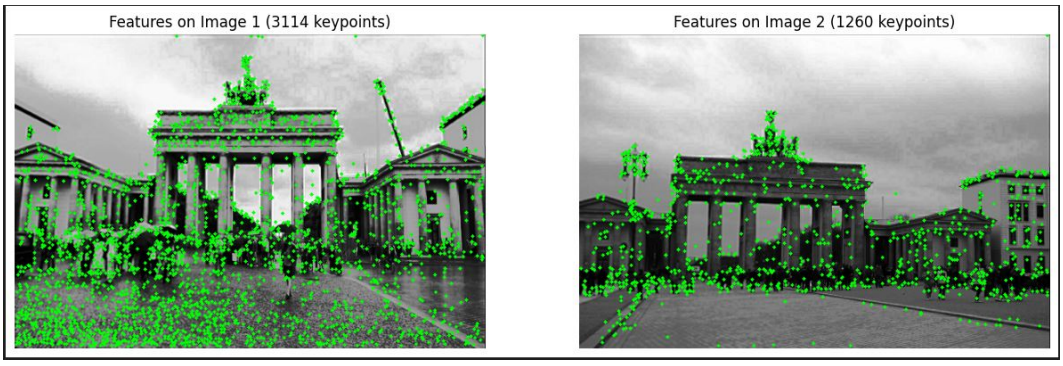

Figure 2: Image without additional preprocessing.

2.2 Feature Extraction and Matching

Feature extraction and matching are essential for establishing correspondences between image pairs in the 3D reconstruction pipeline. The Scale-Invariant Feature Transform (SIFT) algorithm was

employed for feature extraction due to its robustness to scale, rotation, and illumination changes. The extracted keypoints and descriptors formed the basis for matching.

Initially, brute-force matching with a distance threshold was employed, but it yielded inconsistent precision. To improve accuracy, a FLANN-based (Fast Library for Approximate Nearest Neighbors) matcher with $k = 2$ nearest neighbors was utilized. Despite enhancing computational efficiency, this approach introduced false positives, which were mitigated using a ratio test with a threshold of 0.8.

The remaining matches were refined with the RANSAC (Random Sample Consensus) algorithm, which eliminated outliers by estimating a robust homography matrix. This ensured reliable matches adhering to geometric constraints. The combination of SIFT, FLANN, ratio testing, and RANSAC significantly improved match quality, forming a robust feature matching pipeline critical for accurate 3D reconstruction.

Figure 3: An example of feature matching.

2.3 Camera Intrinsics

The camera intrinsics were obtained from a file containing the camera matrices. The relevant data from this file was processed to extract the intrinsic parameters for each camera. These parameters included the focal lengths in the x and y directions, as well as the coordinates of the principal point. Using these values, the intrinsic matrix for each camera was constructed, which describes the relationship between the 3D world coordinates and the 2D image coordinates. This intrinsic matrix was essential for the subsequent triangulation and pose estimation steps.

2.4 Camera Pose Estimation

Camera pose estimation was performed using a combination of intrinsic parameters and essential matrix decomposition. Initially, intrinsic camera matrices K were derived from the camera specifications, which included the focal lengths and principal points. These matrices were used to normalize the image coordinates of feature points, facilitating more accurate pose estimation. The essential matrix E, which encapsulates the relationship between matched points in two views, was computed.

This matrix was then decomposed to obtain the rotation R and translation t matrices that represent the relative position and orientation of the cameras. The decomposition is governed by the equation:

$$E = [t]_\times R,$$

where $[t]_\times$ is the skew-symmetric matrix of the translation vector t, defined as:

$$[t]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

The decomposition process yields four possible solutions for the camera's rotation and translation, and the physically valid solution was selected by verifying which configuration resulted in 3D points that lie in front of both cameras. This was achieved through triangulation of the feature points. An image illustrating this essential matrix decomposition is shown below in Figure 5.

```
Pose estimated for pair (0, 1):
Rotation matrix R =
[[ 0.99797048  0.04351672 -0.04648879]
 [-0.04207552  0.9986164   0.03154261]
 [ 0.0477971  -0.02952256  0.99842068]]
Translation vector t =
[[-0.01056558]
 [-0.06876257]
 [ 0.9975771 ]]
```

Figure 4: Visualization of the essential matrix decomposition.

In addition to this method, quaternions representing camera orientation were also used. The quaternion-to-rotation matrix conversion formula:

$$R = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}$$

was employed to obtain the rotation matrix R directly from quaternion values. While both the essential matrix decomposition and quaternion-based methods yielded consistent results, the former approach was preferred due to its integration with the intrinsic calibration and its robustness in accurately aligning the camera poses for the subsequent stages of 3D reconstruction. Both the essential and fundamental matrix approaches rely heavily on accurate feature point correspondences between the

views, making the quality of point matching crucial. The essential matrix, in particular, provides a more direct relationship between the camera pose and the 3D points, making it more effective in scenarios where precise point correspondences are available.

```
Camera pose for the first image (ID = 341):
Rotation matrix:
[[ 0.99553092  0.00909729 -0.09399696]
 [-0.0059991   0.99943103  0.03319065]
 [ 0.09424542 -0.03247842  0.99501907]]
Translation vector:
[[-0.08342583]
 [-0.51367691]
 [-1.80565123]]
```

Figure 5: Visualization of R,T through qauternion values.

2.5 3D Point Triangulation

In this step, the 3D geometry of the scene was reconstructed using linear triangulation to compute the 3D coordinates of points from corresponding 2D points in image pairs. The triangulation process utilized the camera projection matrices derived during pose estimation. For each image pair, the camera projection matrices P_1 and P_2 are formed by combining the intrinsic camera matrices K_1 and K_2 with the rotation matrices R and translation vectors T from the pose estimation. Specifically, P_1 is the standard projection matrix for the first camera, while P_2 is the projection matrix for the second camera, which incorporates the rotation and translation. The 3D points X_{3D} are then computed using the following linear triangulation equation:

$$P_1 \cdot X_{3D} = \lambda_1 \cdot \mathbf{p}_1, \quad P_2 \cdot X_{3D} = \lambda_2 \cdot \mathbf{p}_2$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the homogeneous coordinates of the matched 2D points in the first and second images, and λ_1 and λ_2 are the scale factors for each camera. The points are then normalized by dividing by their homogeneous coordinate, yielding the final 3D coordinates. This triangulation is computed for all the matched points between image pairs, resulting in a set of 3D points that represent the reconstructed scene.

The triangulated 3D points are computed using the function `cv2.triangulatePoints()` in OpenCV, which requires the projection matrices and the 2D point correspondences as input. The final output is a collection of 3D points stored in the point cloud, which is used for further scene reconstruction and analysis.

2.6 Point Cloud Generation

The reconstructed sparse 3D points were visualized as a point cloud using Open3D and Matplotlib for better clarity. To enhance the visualization, the points were filtered to remove outliers and redundant points, and then the points were scaled appropriately. The point cloud was then plotted in a 3D space, where the x, y, and z coordinates of the points were displayed with a color map based on the z-values, giving a clear representation of the spatial distribution of the points. The plot was customized to hide the axes and present the 3D point cloud from a top-down view, allowing for a more intuitive understanding of the reconstructed scene. Although mesh reconstruction was considered, it was not implemented in this case, focusing solely on the point cloud for visual representation.

Figure 6: Visualization of the generated point cloud.

3 Challenges

The project encountered several challenges, primarily related to camera matrix computation, feature matching, triangulation, and computational efficiency. Each obstacle required technical expertise and careful consideration of available computational resources and dataset characteristics.

The first challenge involved the accurate calculation of intrinsic camera parameters, critical for pose estimation and triangulation. Small errors in these parameters could significantly impact the accuracy of camera pose estimation, resulting in distortions in the final 3D reconstruction. Ensuring the correct application of these parameters was vital to avoid substantial discrepancies in the output.

Feature matching posed another difficulty. The initial brute-force matching technique was inefficient and yielded few correct correspondences. FLANN-based matching improved speed but introduced more false positives, necessitating the use of RANSAC for outlier rejection. Additionally, poorly lit scenes and foreground objects obstructing the landmarks in many images further reduced the matching effectiveness. To address this, image preprocessing was improved by equalizing brightness and adding blur, enhancing the visibility of landmarks and improving feature matching.

The triangulation process also presented challenges, particularly in ensuring numerical stability. Errors in 2D correspondences or camera matrices could destabilize triangulation, particularly when points were near-degenerate. Maintaining stability was essential for accurate 3D point estimation.

Computational demands were another significant issue. Initially, a combinatorial $n2$ approach for feature matching across image pairs proved too computationally expensive, especially as the dataset grew. To address this, the approach was changed to sequential matching over the entire dataset. This reduced the computational burden, though it required balancing runtime efficiency with matching accuracy.

Finally, while semantic maps were explored to improve feature detection, they did not yield significant improvements. Despite their potential in some contexts, their use in this project did not enhance feature matching or the reconstruction process, leading us to focus back on traditional methods.

4 Conclusion

This project successfully applied the Structure from Motion (SfM) technique to generate a 3D point cloud from a series of images, demonstrating the effectiveness of feature-based methods and robust estimation techniques. Despite challenges related to feature matching and computational efficiency, solutions such as preprocessing image brightness and applying blur helped improve feature detection and matching accuracy. The resulting sparse 3D point cloud effectively captured the scene's geometry, and the modular pipeline design allows for easy future enhancements, including integrating photogrammetry or deep learning-based depth estimation to refine the reconstruction.

The project also highlighted the importance of balancing computational cost and accuracy in large datasets, with the shift from combinatorial matching to sequential feature matching providing a more

manageable workflow. While semantic maps did not significantly improve results in this context, the project provided valuable insights into handling noisy data and optimizing SfM pipelines. Looking ahead, the modular nature of the approach offers opportunities for further development, making it suitable for various applications, such as digital modeling and autonomous navigation.

References

[1] GeeksForGeeks. (n.d.). *Python OpenCV – Epipolar Geometry*. Retrieved from `https://www.geeksforgeeks.org/python-opencv-epipolar-geometry/`

[2] Muja, M., & Lowe, D. G. (2009). Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *VISAPP International Conference on Computer Vision Theory and Applications* (pp. 331-340).

[3] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2), 91-110.

[4] OpenCV Documentation. *OpenCV Library*. Retrieved from `https://docs.opencv.org/5.x/`

[5] Addison, Automatic. (2024). *How to Convert a Quaternion to a Rotation Matrix*. Retrieved from `https://automaticaddison.com/how-to-convert-a-quaternion-to-a-rotation-matrix/`