~~Buildings~~ Testudo Built in Minutes: an SFM Approach

Vikram Setty
University of Maryland
vikrams@umd.edu

Lowell Lobo
University of Maryland
lorocks@umd.edu

Apoorv Thapliyal
University of Maryland
apoorv10@umd.edu

Vinay Lanka
University of Maryland
vlanka@umd.edu

Abstract

This report provides the foundations and implementation details for using a Structure from Motion (SFM) pipeline to reconstruct a 3D point cloud (PCL) of a scene using images from multiple views of the scene submitted as the final project of the course Perception for Autonomous Systems (ENPM673) at the University of Maryland.

https://github.com/vikrams169/Structure-from-Motion-SFM-/

1. Introduction

Structure from Motion (SFM) is a popular technique to reconstruct a 3D scene and simultaneously obtain the camera poses of a monocular camera with respect to the given scene. With major strides in the field being introduced in the paper *Building Rome in a Day* by Agarwal et. al. [1], this technique uses a series of geometric computer vision techniques and optimizations to reconstruct a 3D point cloud from a series of images with overlapping parts of the scene. The pipeline followed in this project very closely resembles the method outlined by Agarwal et. al. [1] and a detailed explanation is available on the University of Maryland's CMSC733 Webpage [2].

2. Structure from Motion (SFM) Pipeline

The high-level pipeline used in this project is shown in Figure 1. Starting from making feature associations among all images and matching and tracking them, the key points of these features are extracted. Using these key points and known camera intrinsic parameters, the fundamental matrix and essential matrix between two reference images are computed. Assuming the camera frame of one of these two reference images to be the world frame, the other image's camera pose is estimated by using decomposing the essential matrix and checking the cheirality condition among all possible poses. After selecting the best pose, linear and non-linear triangulation methods are used to extract a good estimate of the 3D points of the corresponding matching 2D key points of the two images in the world frame. Then, for the remaining images, perspective-n-point (PnP) estimation (combination of PnP RANSAC and non-linear PnP) is used to extract their approximate camera pose. Then, bundle adjustment is performed (for each image apart from the first two references) to optimize all camera poses and 3D point locations all at once. The result of the final bundle adjustment process for the final image being registered is considered the final set of 3D points used to construct a 3D point cloud of the reconstructed scene.

The following subsections go over the steps outlined above in further detail.

2.1. Feature Extraction and Matching

The first and the most difficult step of the entire pipeline is collecting features from all the images and maintaining a record of which features are common among which images and where they are. Even small errors in extracting this information can lead to bad results that propagate through the pipeline. Various methods like optical flow, traditional methods with approximations (like planar surfaces), and other deep-learning-based approaches are common methods to solve this problem. However, they can seem to prove computationally expensive and often don't produce results with the level of accuracy desired. Thus, for the scope of this project, we use a set of images that have a large part of the scene overlapping among all of them (all the images) and extract only the features common to/present in all of the images. We make use of a SIFT feature descriptor to extract the images and then use a brute-force KNN (K-Nearest Neighbor) Matcher to match the features among all the images to get the corresponding key point locations. Though this method limits the field of view from where the scene

```
Data: Image Matches, K
Result: X, C, R
for all possible pair of images do
    // Reject outlier correspondences
    [x1, x2] = GetInliersRANSAC(x1, x2);
end
// For first two images
F = EstimateFundamentalMatrix(x1, x2);
E = EssentialMatrixFromFundamentalMatrix(F, K);
[Cset, Rset] = ExtractCameraPose(E);
// Perform linear triangulation
for i = 1:4 do
    Xseti = LinearTriangulation(K, zeros(3,1), eye(3), Cseti, Rseti, x1, x2);
end
// Check cheirality condition
[C R] = DisambiguateCameraPose(Cset, Rset, Xset);
// Perform Non-linear triangulation
X = NonlinearTriangulation(K, zeros(3,1), eye(3), C, R, x1, x2, X0));
Cset = C, Rset = R;
// Register camera and add 3D points for the rest of images
for i=3:I do
    // Register the i^th image using PnP.
    [Cnew Rnew] = PnPRANSAC(X, x, K);
    [Cnew Rnew] = NonlinearPnP(X, x, K, Cnew, Rnew);
    Cset = Cset∪Cnew, Rset = Rset∪Rnew;
    // Add new 3D points.
    Xnew = LinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2);
    Xnew = NonlinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2, X0);
    X = X∪Xnew;
    // Build Visibility Matrix.
    V = BuildVisibilityMatrix(traj);
    // Perform Bundle Adjustment.
    [Cset Rset X] = BundleAdjustment(Cset, Rset, X, K, traj, V);
end
```

Figure 1. Structure from Motion (SFM) Pseudocode

can be reconstructed (as we can only use points that overlap in all images), it ensures that features are matched accurately and thus maintains the quality of the points being reconstructed.

2.2. Fundamental Matrix Calculation

The next step of the SFM pipeline includes calculating the Fundamental Matrix (F) between two reference images from the image set. The Fundamental Matrix is an algebraic representation of epipolar geometry. Each corresponding key point in the two images (x_i' and x_i) (where i represents the key point index) is related by the expression $x_i' F x_i = 0$. The Fundamental Matrix can be calculated using the *Eight-Point Algorithm* where eight key point correspondences are used to solve for the entries of the Fundamental Matrix by using the equations below.

$$x_i' F x_i = 0$$

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

$$\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ x_mx_m' & x_my_m' & x_m & y_mx_m' & y_my_m' & y_m & x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

The entries of the Fundamental Matrix (F_{ij}), and thus the Matrix itself too are calculated using Singular Value Decomposition (SVD). Under ideal conditions, the Fundamental Matrix is supposed to have a rank of 2. However, due to noise and other inaccuracies, the epiloilar lines do not all exactly intersect at a single point making the rank 3. To overcome this issue, the last singular value of the Essential Matrix (E) calculated from the Fundamental Matrix using the equation below is manually set to 0 to make $rank(E) = 2$ while finding the camera pose in further steps.

$$E = K^T F K$$

Here, K represents the camera intrinsic matrix (assumed to be constant/equal among all images). Though possible to take any eight points and calculate the Fundamental Matrix using the Eight Point Algorithm, imperfections and inaccuracies in the system hamper performance. Thus, to get the best estimate of the Fundamental Matrix, we use Random Consensus Sampling (RANSAC) to choose the Fundamental Matrix that has the most number of inliers (points that satisfy $|x_i' F x_i| < \epsilon$, where ϵ is a tunable threshold value) by performing multiple iterations of sampling eight random points from all key points to calculate F from.

2.3. Triangulation and Cheirality Check

The next step in the pipeline includes finding the camera poses of the two reference images and simultaneously making an estimate for the 3D points of the common features between them. This is done using linear and non-linear triangulation along with a cheirality check step to disambiguate incorrect camera poses.

Firstly, the camera pose of one of the reference images (R_1, C_1) is assumed to be the world frame. Thus, that camera pose is related to the world frame by an identity matrix for rotation and a zero vector for translation. With this assumption, the second reference image's camera pose can take one of four combinations (which can be obtained by performing Singular Value Decomposition (SVD) on

the Essential Matrix calculated in the previous step). To check which of these four poses is the correct camera pose for the second reference image, linear triangulation is performed using each possible camera pose to get a set of 3D points corresponding to each feature key point. A set $X = X_1, X_2, X_3, X_4$ of possible 3D point locations is thus obtained. To get the most appropriate pose of the second reference image (R_2, C_2) out of the four possibilities, a cheirality check is performed. A 3D point from $X_i \in X$ is considered to be an inlier if $r_3(X_i - C_2) > 0$ where r_3 is the third row of R_2. The camera poses (R_2, C_2) that has the most inliers is said to be the most appropriate camera pose for the second reference image. The cheirality check just performed just ensures that points considered inliers appear in front of the camera (i.e. have a positive Z value in the camera's coordinate frame).

After disambiguation of the most appropriate camera pose for the second reference image, a more accurate estimate of the 3D point locations corresponding to the feature key points common to both the reference images is obtained by using non-linear triangulation which uses a least-squares optimization to minimize the projection error (squared euclidean distance in pixel coordinates between the actual keypoint features and the 3D points reprojected using the projection matrix). The loss to be minimized during non-linear triangulation is shown below.

$$min_x \sum_{j=1,2} \left(\left(u^j - \frac{P_1^{jT} \tilde{X}}{P_3^{jT} X} \right)^2 + \left(v^j - \frac{P_2^{jT} \tilde{X}}{P_3^{jT} X} \right)^2 \right)$$

Here, P_i refers to the rows of the projection matrix, and (u_j, v_j) refers to actual feature key point locations. X refers to the corresponding 3D point and $\tilde{X}$ refers to the homogenous representation of the same. Overall, all this loss function signifies the Euclidean distance (in terms of pixels) between the key point locations and the reprojected pixel locations of the corresponding estimated 3D points. Minimizing this error with respect to the current estimate of the 3D point locations using a least-squares optimizer in non-linear triangulation helps tune the estimate of the 3D points.

2.4. Perspective-N-Point (PnP) Estimation

After registering the first two reference images, it is necessary to register the remaining images from the images from the image set as well. This includes estimating each remaining image's camera pose as well as using the key point feature information to refine the estimation of the 3D point locations. This is achieved to a certain level by Perspective N Point (PnP) Estimation. The first step in PnP estimation is performing PnP RANSAC that returns the estimate for the camera pose (R_i, C_i) for the i^{th} image. PnP RANSAC makes use of linear PnP, a technique that uses Singular Value Decomposition (SVD) on top of computation performed on the initial estimate of 3D points already available reprojected using the projection matrix against the key points locations of that image. Again by thresholding the linear PnP error against a threshold like done while calculating the Fundamental Matrix using RANSAC, the camera pose (R_i, C_i) that maximizes the number of inliers is considered the best estimate. This estimate then becomes an input for the non-linear PnP step which further optimizes the estimate of the camera pose (R_i, C_i). This is again done by using a least-squares optimizer.

Using the linear and non-linear PnP process elaborated above, a good estimate of the camera pose (R_i, C_i) for the i^{th} image is obtained. This estimate can be considered reasonably good considering the amount of optimization done to obtain it. By taking this into account, we can use the new camera pose information to estimate the 3D point locations of each feature key point in $image_i$ by using linear triangulation with earlier registered images ($image_j$ where $j < i$) (including the reference images). Then, using non-linear triangulation, this estimate can be refined considering the new triangulated points. the loss function in non-linear PnP to optimize the estimate using the least-squares technique is shown below.

$$min_{C,R} \sum_{i=1,j} \left(\left(u^j - \frac{P_1^{jT} \tilde{X}_j}{P_3^{jT} X_j} \right)^2 + \left(v^j - \frac{P_2^{jT} \tilde{X}_j}{P_3^{jT} X_j} \right)^2 \right)$$

2.5. Bundle Adjustment

The last step of the pipeline (for each new registered $image_i$) is bundle adjustment. This step helps minimize outliers and accounts for rare erroneous/misaligned points and prevents them from propagating throughout the rest of the pipeline. Here, the estimate for camera poses of all the images (including the reference images) along with the locations of all the 3D points across all the images registered so far (including the current one being processed) gets optimized using a least-squares optimizer. Bundle adjustment, like the other steps mentioned before aims to minimize projection error. This step, however, optimizes a lot of parameters $(3N + 7(i+1))$ parameters while registering the i^{th} image in total (where N is the total number of 3D points/features common among all images). Thus, this step is computationally expensive, and setting lower thresholds in the least-squares optimizer though effective can sometimes be unrealistic. The loss function (resembling the one used in non-linear triangulation and non-linear PnP) that the least-squares optimizer uses is shown below.

$$min_{C,R\in[1,I],X_{j=1}^{J}} \sum_{i=1}^{I}\sum_{j=1}^{J} V_{ij}\left(u^j - \frac{P_1^{jT}\tilde{X}_j}{P_3^{jT}X_j}\right)^2 + V_{ij}\left(v^j - \frac{P_2^{jT}\tilde{X}_j}{P_3^{jT}X_j}\right)^2$$

Here, V_{ij} is the visibility matrix which in our case is a matrix full of ones because we consider only features that are observable/present in all images of the images set.

2.6. Pipeline Approach 1

We have implemented two versions of the pipeline, both based on the pseudocode mentioned in Figure 1, though slightly different from one another. The first approach follows the pipeline exactly, starting out with finding the features common to all the images in the image set followed by running the entire pipeline (including bundle adjustment) to calculate and optimize a set of 3D point locations, with each point corresponding to a matched feature.

2.7. Pipeline Approach 2

Another approach, slightly different from the one mentioned in the method aforementioned includes adjusting the image set so that the images surrounding the scene (with not necessarily a set of overlapping features in all images) appear in chronological order of when they were taken. In this method, a set of 3D points is calculated for the features common between each consecutive pair of images in the set. Though this approach is repetitive in the points being calculated for the scene, it actually helps increase the density of the point cloud/mesh when trying to visualize and understand the reconstructed scene. The slight error in each unique point's calculation among each consecutive image pair also prevents points from overlapping one another leading to a decent point cloud for visualization. Although, the disadvantage of this method is that the number of points in now in the order of NM instead of M (where M is the number of images in the image set and N is the number of features corresponding to 3D points), leading to a faster time of execution. In addition, this method significantly increases the number of parameters to optimize in bundle adjustment which often makes it infeasible to perform.

2.8. Visualization

2.8.1 Point Cloud Visualization

After registering all the images from the image set, the refined set of 3D points is obtained. These set of points are further used to reconstruct the 3D view of the scene. This can be done using a variety of visualization techniques. the most popular representations include point clouds and meshes with the former being preferred as point clouds are lighter, computationally more efficient, and better able to visualize and understand the mapping of key points to 3D points. To construct and visualize the point cloud, *Open3D* is used.

2.8.2 3D Object Visualization

Another method of visualizing the results is by creating a ".ply" file and appending all points and their resulting colors into it. The ".ply" file can be viewed either online or by using a 3D object display software.

3. Hardware Setup

The SFM pipeline was executed on a Jetson Orin Nano using images collected from a ZED camera.

The ZED Camera is a perfectly calibrated camera with known intrinsic parameters. It's also a stereo camera that has a real-time 3D mapping mode where the ZED can be moved around to generate a 3D mesh of any indoor and outdoor environment.

The Jetson Orin Nano on the other hand is single-board computer that is optimized for computer vision-based applications. It works perfectly with the ZED camera since there exists an SDK, specific to the Jetson Orin Nano, that is optimized for CUDA and provides depth estimation, motion sensing, and spatial AI. The SFM pipeline was executed within a virtual environment for easy development.

Although the official documentation provides the intrinsic camera parameters, the ZED camera was re-calibrated using an inbuilt ZED calibration tool. The calibration file contains information about the precise position of the right and left cameras and their optical characteristics. This file is key for the SFM process and guarantees its quality. The calibration app works by having the ZED camera recognize patterns on a chessboard pattern with instructions on how to orient the camera to get various poses.

Figure 2. Jetson Orin Nano & ZED Camera

4. Results

Multiple experiments were conducted some using images collected from the above mentioned hardware, and others from online datasets that provide images and camera intrinsic matrix for execution. The entire pipeline was primarily tested on three online datasets and finally using the ZED camera, custom images were collected and SFM was performed. The experiments were performed starting with simple scenes and building up to complex scenes. Most of the dataset contained around 30 to 70 images.

4.1. Experiment 1

For this experiment, an online dataset that contains images of the entryway of a building is used.

4.1.1 Experiment Environment

Below is displayed the image of the scene on which SFM will be applied.

Figure 3. Dataset 1: Building Entryway

4.1.2 SFM Pipeline Results

The results are as displayed below,

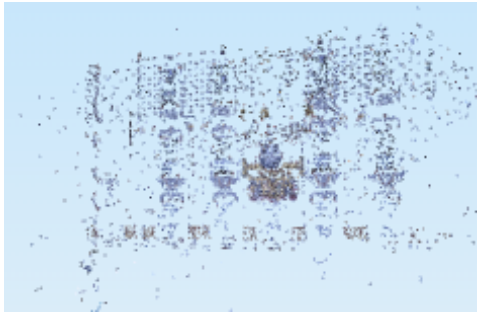

Figure 4. SFM Results for Entryway

4.2. Experiment 2

For this experiment, an online dataset that contains images of a fountain is used.

4.2.1 Experiment Environment

Below is displayed the image of the scene on which SFM will be applied.

Figure 5. Dataset 2: Fountain

4.2.2 SFM Pipeline Results

The results are as displayed below,

Figure 6. SFM Results for Fountain

4.3. Experiment 3

For this experiment, an online dataset that contains images of the entire courtyard of a castle is used. For this experiment, there is far more variation in the images as compared to the experiments 1 and 2.

4.3.1 Experiment Environment

Below are displayed a few images of the scene in which SFM will be applied.

Figure 7. Dataset 3: Image 1

Figure 8. Dataset 3: Image 2

Figure 9. Dataset 3: Image 3

4.3.2 SFM Pipeline Results

The results are displayed in figures Fig.10, Fig.11 and Fig.12, and since the SFM in this case displays an entire scene, multiple images with different perspectives will be displayed.

4.3.3 Discussion

For the experiment, it can be noticed that the entire scene is constructed and displayed as a point cloud. Looking closely, at the point cloud, the tractor within the courtyard is generated in the point cloud. This shows that the pipeline works as expected and the point cloud is being portrayed with the color properly.

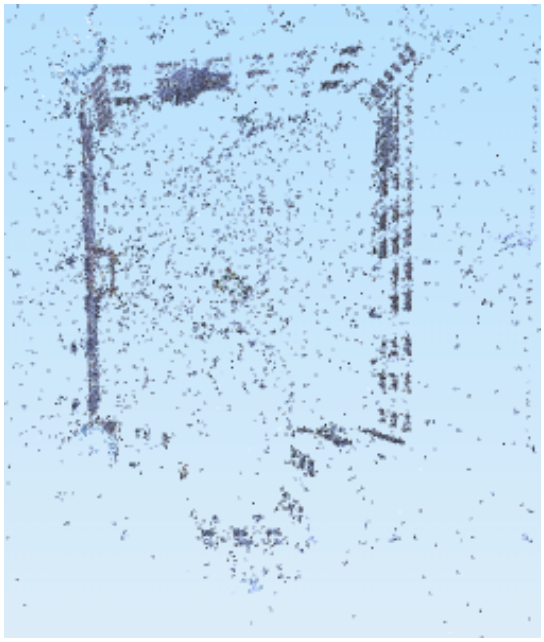

Figure 10. SFM Results - Castle Top View

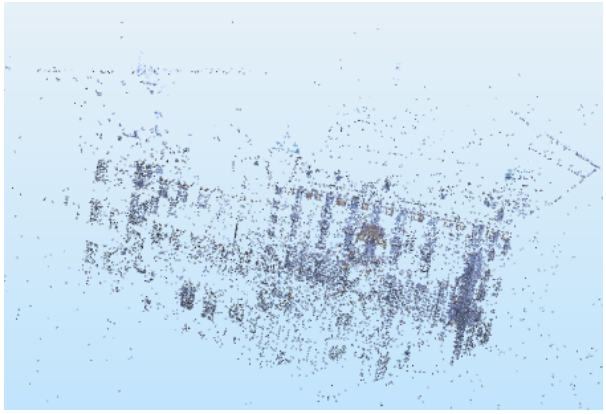

Figure 11. SFM Results - External View

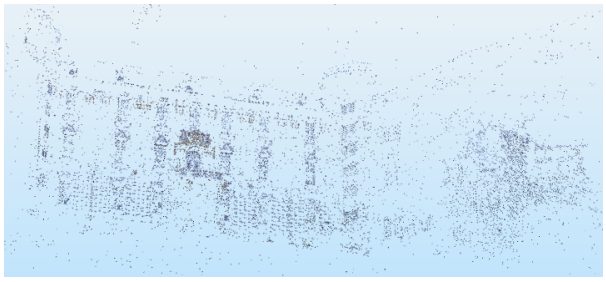

Figure 12. SFM Results - Internal View

4.4. Experiment 4

4.4.1 Experiment Environment

For this experimentation, images of Testudo were taken to perform SFM. Below is shown the front view of Testudo.

Figure 13. Testudo

4.5. SFM Pipeline Results

The results are as displayed below,

Figure 14. SFM Results for Testudo

4.6. ZED Camera API Results

The results from the spatial mapping API using the ZED ADK are as below,

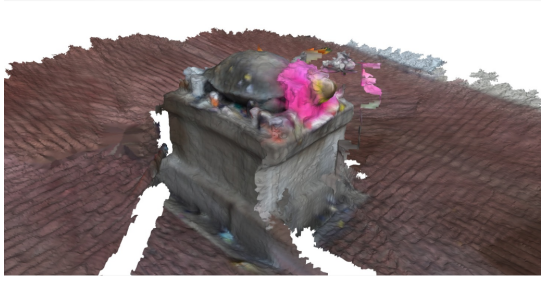

Figure 15. 3D Scene from spatial Mapping API

4.7. Discussion

Staring at the results of the spatial mapping API of the ZED camera, it can be noticed that the statue of Testudo has been generated quite accurately except for the distortion of pink noticed near the head of Testudo.

While comparing the result from the SFM pipeline to the actual ground truth, we see that the SFM pipeline, in this case, is able to highlight prominent features, but is unable to generate a precise point cloud of Testudo. The results seem distorted, albeit the features appear in their expected locations. This distorted point cloud could be attributed to erroneous matching during RANSAC operations due to the abundance of features in the background (trees) as noticed by the green cloud on the right side of the image. Another reason could be the variation in lighting while capturing images. It is also important to note that although good results are obtained for the datasets, the pipeline generates subpar results for the custom dataset. This may be due to some invalid calibration of the camera. The online datasets provide the intrinsic parameters, while the ZED camera needs to be calibrated and displays high distortion coefficients. Another reason for the disparity in results could be caused by the fact that the images taken in the dataset have the camera's center as constant but in the custom images, the camera center keeps being shifted.

5. Conclusion

In conclusion, an SFM pipeline was built to generate point clouds from a set of 2D images. Three online datasets and a custom dataset were used to test and optimize the pipeline.

The pipeline was successfully executed on the Jetson Orin Nano and displayed comparable execution times to that of a normal computer. It is also important to note that the pipeline does not make use of any CUDA capabilities in the Jetson Orin Nano.

The pipeline showed perfect results when executed over the online dataset, but displayed a mildly distorted 3D scene for the custom dataset. This could be attributed to incorrect calibration, variable lighting or an abundance of unnecessary background features that affect feature matching.

Finally, for future work, CUDA could be used to sync with the pipeline to display faster inference times and a better calibration pipeline could be built to auto-calibrate the ZED camera.

6. Appendix

6.1. Contributions

1. Vikram (vikrams) - Linear and Non-Linear PnP and Remaining Image Registration
2. Lowell (lorocks) - Fundamental Matrix RANSAC, Linear and Non-Linear Triangulation, First Two Image Registration
3. Apoorv (apoorv10) - Cheirality Check, Bundle Adjustment
4. Vinay (vlanka) - Hardware Setup, Pipeline Integration, and Benchmarking

References

[1] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *2009 IEEE 12th International Conference on Computer Vision*, pages 72–79, 2009. 1

[2] Cmsc733 sfm webpage. https://cmsc733.github.io/2022/proj/p3/. 1