Structure from Motion for Object Reconstruction

Easton Potokar

December 13, 2022

Abstract

This project combines two of my interests: 3D printing and state estimation. A common 3D printing tool, a 3D scanner is used to scan objects that need to be duplicated, and is often of the do-it-yourself variety and equipped with a monocular camera [1]. Once images are taken, they are generally post-processed in a commercial software to create a 3D reconstruction of the "scanned" object. For my project, I replaced this commercial software by implementing structure from motion [2] to estimate camera poses and generate a dense point cloud. From this point cloud, a 3rd-party software can be used to fully reconstruct the mesh. To fully understand structure from motion, I implemented it by hand in python with the exceptions of OpenCV and a sparse matrix solver; no gtsam [3] or Ceres [4] were used for the bookkeeping or optimization. In this document, I review these mathematical concepts, as well as a few of the implementation choices I made.

1 Introduction

Structure from Motion (SfM) is a technique used to jointly estimate camera intrinsics, camera poses, and 3D points from a sequence of images. It relies on the perspective geometry used by cameras as well as a number of computer vision algorithms for initialization purposes, all of which I review in Section 2. Additionally, once initialization is performed, the problem is formulated as a nonlinear least-squares problem, which I review in Section 3. Finally, the full SfM pipeline and results are given in Section 4. My code implementation can be found on github at contagon/sfm.

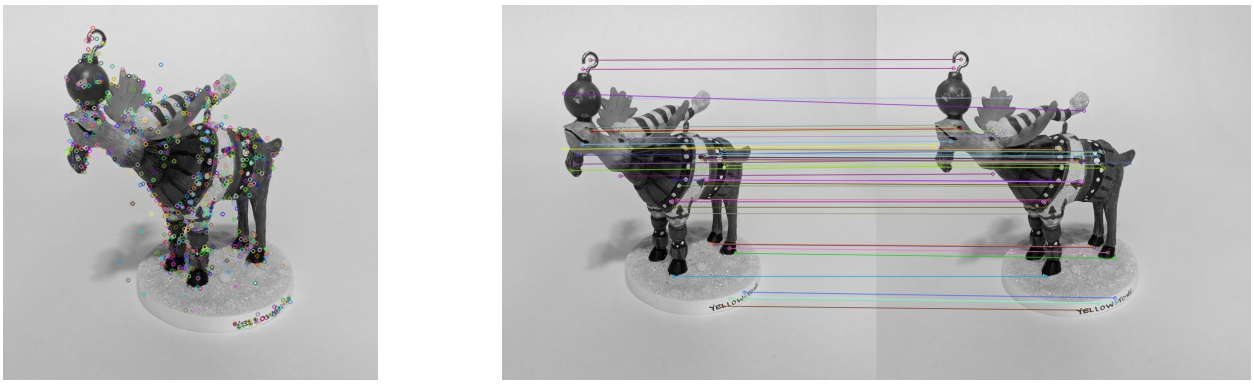
2 Computer Vision

2.1 Perspective Geometry

Typical cameras operate through what is known as perspective geometry, which means they project 3D homogeneous world coordinates into image pixels. This is done as follows

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \alpha KTP \triangleq \alpha \begin{bmatrix} f_x & 0 & p_x & 0 \\ 0 & f_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

where the matrix K is known as the intrinsic parameters of the camera, $T \in SE(3)$ is the transform from world coordinates to camera coordinates, P is a 3D point in world coordinates, and finally α is a scale factor to normalize the resulting homogeneous coordinate.



(a) SIFT features

(b) Matches

Figure 1: Examples of found SIFT features (a) and cross checked brute force matches with no outlier removal (b). The image was converted to grayscale and scaled to 25% for visualization purposes.

Throughout this write-up, I separate the scaling and projective functions as follows

$$f(K,T,P) = KTP, \qquad h(p) = h\left(\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}\right) = \begin{bmatrix} p_x/p_z \\ p_y/p_z \\ 1 \end{bmatrix} \tag{2}$$

$$\pi(K,T,P) = h(f(K,T,P)). \tag{3}$$

This will be helpful as we derive Jacobians in the optimization section.

2.2 Features

In order to find 3D points to solve for, "features" have to be found in each image and matched between images. Features are often chosen as corners as they are invariant to things such as scale, rotation, etc. Once a feature is found, a unique descriptor is assigned to it, often derived from gradient information from its image patch. These descriptors can then be used to match features across images.

After testing a number of different features and descriptors, I found I got the best results when using scale-invariant feature transform (SIFT) [5] along with a brute-force matching scheme with cross checking. An example of SIFT features and brute force matches can be seen in Figure 1.

2.3 Initialization

Since SfM is heavily dependent on the initialization of parameters, it's important to properly extract good estimates once features are matched. For conciseness, I don't go into mathematical details, but I quickly review my implementations here.

If none of the feature matches correspond to previously triangulated 3D points, an essential matrix E can be estimated between the two frames using the five-point algorithm [6], and from E a relative pose transformation T can be found up to scale. Matched features can then be triangulated.

If there is previously triangulated 3D points matched, then a perspective-n-point (PnP) [7] algorithm can be used to find the 3D pose of the new camera image. The rest of the matched features can then be triangulated using the relative pose between frames.

3 Nonlinear Least-Squares

Once features have been matched, an optimization is performed to minimize the square of the reprojective error as follows

$$argmin_{K,T,P} \frac{1}{2} \sum_{i}^{M} \sum_{j}^{N} (z_{ij} - \pi(K, T_i, P_j))^2 \tag{4}$$

where M is the number of images, and N is the number of matched features seen in image i. Note, I assume that each image was taken from the same camera, therefore I only solve for a single intrinsic matrix K.

This is a nonlinear least-squares problem, and I solve it using traditional techniques as follows. Define a residual by $r_{ij} = z_{ij} - \pi(K, T_i, P_j)$ and stack them into a single vector $r(\theta)$, where θ is the set of all the parameters being optimized. This results in the objective function $f(\theta) = \frac{1}{2} r^T r$ which I can expand using it's Taylor series as

$$f(\theta \oplus \delta\theta) = f(\theta) + g^T \delta\theta + \frac{1}{2} \delta\theta^T H \delta\theta \tag{5}$$

where g, H are the gradient and Hessian of f respectively. Note, I use $\oplus$ here, since I perturb $T_i \in SE(3)$ using the matrix exponential and multiplication as in $T_i \oplus \delta T_i = T_i \exp(\delta T_i^{\wedge})$ with $^{\wedge}$ the operator that maps from $\mathbb{R}^6$ to a Lie algebra. For K and the points P, I simply use vector addition. Differentiating the above expansion with respect to $\delta\theta$ and setting it equal to 0 to find an extrema, gives

$$H \delta\theta = -g \tag{6}$$

Further note, g and H have the following formulations, letting $\frac{\partial r}{\partial \theta} = J$

$$g = \frac{\partial f}{\partial \theta} = J^T r \tag{7}$$

$$H = \frac{\partial^2 f}{\partial x^2} = J^T J + \frac{\partial^2 r}{\partial \theta^2} r \approx J^T J \tag{8}$$

which gives us the following step equation

$$J^T J \delta\theta = -g \tag{9}$$

Solving this iteratively results in the Gauss-Newton optimizer. Oftentimes, $J^T J$ is singular and causes too large of an optimization step, so I modify H by doing $H = J^T J + \lambda I$ for some $\lambda \in \mathbb{R}$. This results in the Levenberg-Marquardt method and avoids many of the pitfalls of Gauss-Newton. In my implementation, I solve this using the sparse LSMR [8] implementation found in scipy.

3.1 Jacobians

The Jacobian is a sparse matrix, since the measurements z_{ij} only depend on a single pose and 3D point. For completeness, since I couldn't find them anywhere else, I include analytical definitions of the blocks of the Jacobian here. Recalling the projective function π found in Eq. (2),

$$\frac{\partial \pi}{\partial K} = \frac{\partial h}{\partial p} \frac{\partial f}{\partial K}, \qquad \frac{\partial \pi}{\partial T} = \frac{\partial h}{\partial p} \frac{\partial f}{\partial T}, \qquad \frac{\partial \pi}{\partial P} = \frac{\partial h}{\partial p} \frac{\partial f}{\partial P} \tag{10}$$

Each of these partials has the following forms,

$$\frac{\partial h}{\partial p} = \begin{bmatrix} 1/p_z & 0 & -p_x/p_z^2 \\ 0 & 1/p_z & -p_y/p_z^2 \end{bmatrix}, \quad p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{11}$$

$$\frac{\partial f}{\partial P} = K \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{12}$$

$$\frac{\partial f}{\partial K} = \begin{bmatrix} P'_x & 0 & P'_z & 0 \\ 0 & P'_y & 0 & P'_z \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad P' = \begin{bmatrix} P'_x \\ P'_y \\ P'_z \end{bmatrix} = TP \tag{13}$$

$$\frac{\partial f}{\partial T} = \lim_{\tau \to 0} \frac{KT \exp(\tau^\wedge) P - KTP}{\tau} \tag{14}$$

$$\approx \lim_{\tau \to 0} \frac{KT(I + \tau^\wedge) P - KTP}{\tau} \tag{15}$$

$$= \lim_{\tau \to 0} \frac{KT\tau^\wedge P}{\tau} = K \begin{bmatrix} -P^\wedge & I \\ 0 & 0 \end{bmatrix} \tag{16}$$

where that last definition has some questionable notation since we're dividing by a vector. It does give the same result as perturbing in the Lie algebra one dimension at a time, however.

4 Structure from Motion

4.1 Pipeline

My implementation of SfM then proceeds iteratively by adding one image at a time as follows,

1. Find features and corresponding descriptors in image. I used the SIFT implementation found in OpenCV.

2. Match descriptors with descriptors in previous 7 images. Due to the large number of features in my images, I used the fast library for approximate nearest neighbors (FLANN) matching first, followed by a brute force matcher with cross checking. Cross checking ensures a match is the nearest descriptor for both the query and training point.

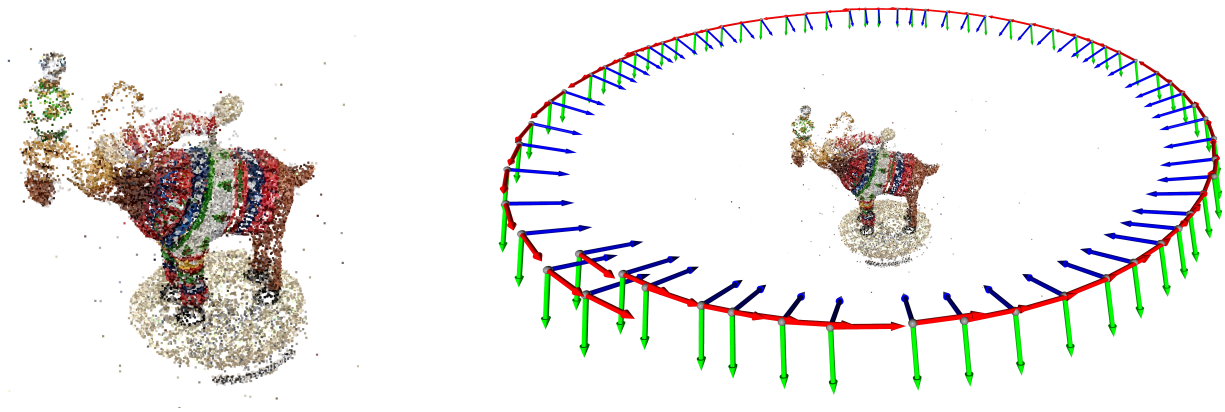
3. Estimate essential matrix. I used OpenCV's five-point essential matrix estimation with RANSAC. This step is largely done to remove non-geometric outliers.

4. Initialize frame pose T_i and any new 3D feature locations P_j. As mentioned in Section 2, T_i can be estimated through the essential matrix up to scale, or through a PnP method. I use OpenCV's implementation for either case, both of which include a RANSAC step to further remove outliers. Triangulation is done through OpenCV's implementation as well.

5. Finally, these estimates were tuned through a Levenburg-Marquadt optimization, using the Jacobians mentioned in the previous section, and my own python implementation.

4.2 Results

To verify implementation, I ran my SfM implementation on a sequence of 66 images taken from around a moose ornament. This ornament was chosen for its textured surface, which provided plenty of features to be matched across the frames. There was around 9k features per frame



(a) Resulting point cloud

(b) Point cloud & camera poses

Figure 2: SfM results for a sequence of 66 images of a moose ornament. The resulting point cloud has around 50k points in it.

and around 4.5k were matched successfully. After removal of outliers via RANSAC methods, around 1.5k were matched to previously triangulated features, and 1k were matched to previously unmatched features. This resulted in around 50k features total after the entire sequences was run. The resulting point cloud and poses can be seen in Figure 2. This point cloud could then be put into a mesh reconstruction algorithm such as ball pivoting [9] or Poisson surface reconstruction [10].

References

[1] OpenScan, "Openscan mini - printable 3d scanner." https://www.printables.com/model/83322, 2022 (accessed December 12, 2022).

[2] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113, 2016.

[3] F. Dellaert and et. al, "borglab/gtsam." https://doi.org/10.5281/zenodo.5794541, 5 2022 (accessed December 12, 2022).

[4] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver." https://github.com/ceres-solver/ceres-solver, 3 2022 (accessed December 12, 2022).

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.

[6] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 756–770, June 2004.

[7] S. Li, C. Xu, and M. Xie, "A robust O(n) solution to the perspective-n-point problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, pp. 1444–1450, July 2012.

[8] D. C.-L. Fong and M. Saunders, "LSMR: An iterative algorithm for sparse least-squares problems," *SIAM Journal on Scientific Computing*, vol. 33, pp. 2950–2971, Jan. 2011.

[9] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, pp. 349–359, Oct. 1999.

[10] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, (Goslar, DEU), pp. 61–70, Eurographics Association, June 2006.