Rotation Averaging for Global SfM

Bryce Evans Rafael Farias Marinheiro

Cornell University*

1 Introduction

Structure from motion is a powerful tool in reconstructing environments without artist modeling. By taking a series of images and finding common points between them, a network of relative camera is created to represent the scene but often has error that causes the reconstruction to have no feasible solution. Robust and efficient rotation averaging [Chatterjee and Govindu 2013] proposes several methods to solve this problem, but does not provide well useable code. Given the power in SfM applications, there is a long pipeline of steps but they are fractured and inconsistent. We present a Python module publicly available to integrate into the greater pipeline that solves the rotation averaging problem and interfaces well with other code available such as the translation averaging solver by [Wilson and Snavely 2014].

2 Background

Current SfM pipelines can be divided in two different classes: An Iterative SfM pipeline (see figure 1a) will try to iteratively add cameras to the working set and it will run a bundle adjustment after each iteration. The quality of the results generated by such pipelines depends on the ordering of the cameras and it is highly sensitive to outliers. One example of such a pipeline is the well-known Photo-Tourism [**?**].

An Global SfM pipeline (see figure 1b) will instead divide the problem in two different global subproblems: The Rotation Averaging problem and the Translation Averaging problem. These problems can be described as follows: Consider a connected graph $G = (V, E)$. Associated with each edge $e = (i, j)$, there is a relative rotation $R_{ij} \in SO(3)$ and relative translation $t_{ij} \in R^3$. We want to assign to each vertex of the graph a global rotation and translation R_i, t_i such that we minimize:

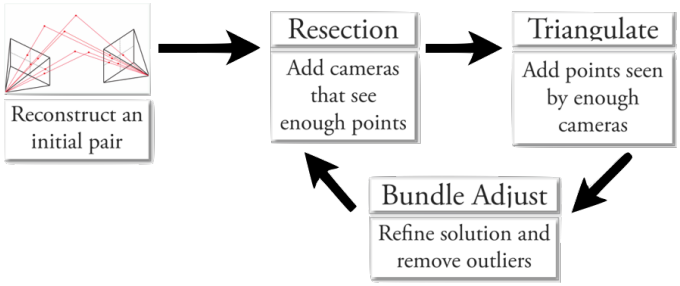
$$\sum_{(i,j)\in E} d_1(R_j R_i^{-1}, R_{ij}) \tag{1}$$

$$\sum_{(i,j)\in E} d_2(t_j - t_i, R_i t_{ij}) \tag{2}$$

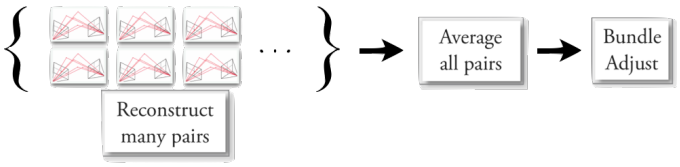
Where d_1 and d_2 are metrics of $SO(3)$ and R^3 respectively. The first equation describes the Rotation Averaging Problem and the second equation describes the Translation Averaging problem. Both problems can be solved globally, so the final solution does not depend on the ordering of the cameras and it is more robust when working with outliers.

In this project, we have implemented the approach described in [Chatterjee and Govindu 2013] to solve the Rotation Averaging problem and we have used the approach described in [Wilson and Snavely 2014] to solve the Translation Averaging problem.

*{bae43, rf356}@cornell.edu



(a) *Iterative SfM Pipeline*



(b) *Global SfM Pipeline*

Figure 1: *(a) On top, the Iterative SfM Pipeline. (b) At bottom, the Global SfM Pipeline.*

3 Rotation Averaging

A Rotation Matrix R is a 3×3 orthonormal matrix (i.e., $RR^\intercal = I = R^\intercal R$) such that $|R| = +1$. All rotation matrices form a closed group known as the Special Orthonormal Group $SO(3)$. The $SO(3)$ is also a Lie Group, so it has the following properties: (1) The neighborhood of a point in the group is topologically equivalent to a vector space known as the Lie algebra. (2) There exist a direct mapping between an element in the Lie Group and an element in the Lie Algebra. In the case of $SO(3)$, the associated Lie Algebra is know as $\mathfrak{so}(3)$.

The first property can be used to locally linearize the Rotation Averaging problem. Let $\omega = \theta n \in \mathfrak{so}(3)$, be the vector in the Lie Algebra that represents the rotation by θ radians around the unit-norm axis n. Let $R \in SO(3)$ be the equivalent rotation matrix (i.e. $\omega = \log(R)$). We can extrapolate the properties of the Lie Group to use the following approximation:

$$\log(R_j R_i^{-1}) = \log(R_{ij}) = \omega_{ij} \approx \omega_j - \omega_i \tag{3}$$

Given the set of relative rotations, one can write a matrix A that encodes a linear system with all the equation involving ω_{ij}. Then, we would have the following linear system:

$$A\omega_{global} = \omega_{rel} \tag{4}$$

However, equation (3) is an approximation. So instead of solving directly for ω_{global}, we can iteratively compute the error of our solution ($\Delta\omega_{rel}$) and then we can update our solution ($\Delta\omega_{global}$) by using the following equation:

$$A\Delta\omega_{global} = \Delta\omega_{rel} \tag{5}$$

The algorithm is described in listings (1). The linear system in line (1:10) can be solved using different methods. One could use the Linear Least Squares Method to solve it, however it is not robust enough to deal with outliers. Instead, we use two different methods. The first one ($\ell 1$ Rotation Averaging - $\ell 1$RA) tries to minimize the $\ell 1$-norm, i.e., it tries to minimize $|Ax - b|_1$ and the second one (Iteratively Reweighted Least Squares - IRLS) tries to robustly minimize the $\ell 2$-norm (See [Chatterjee and Govindu 2013] for more details).

Algorithm 1 Rotation Averaging Algorithm

1: **Input:** The Relative Rotations $R_{rel} = \{R_{ij}\}$ and an initial guess of the solution
2: **Output:** The Global Rotations $R_{global} = \{R_i\}$
3: **procedure** ROTATIONAVERAGING(R_{rel}, R_{guess})
4: $A \leftarrow ComputeMatrix(R_{rel})$
5: **for all** $(i, j) \in R_{rel}$ **do** ▷ Compute $\Delta\omega_{rel}$
6: $\Delta R_{ij} \leftarrow R_j^{-1} R_{ij} R_i$
7: $\Delta\omega_{ij} \leftarrow \log(\Delta R_{ij})$
8: **end for**
9: **while** $|\Delta\omega_{rel}| \geq \epsilon$ **do**
10: $\Delta\omega_{global} \leftarrow A \backslash \Delta\omega_{rel}$ ▷ Solve for $\Delta\omega_{global}$
11: **for all** $i \in R_{global}$ **do** ▷ Update the solution
12: $R_i \leftarrow R_i \exp(\Delta\omega_i)$
13: **end for**
14: **end while**
15: **return** R_{global}
16: **end procedure**

4 Implementation and Results

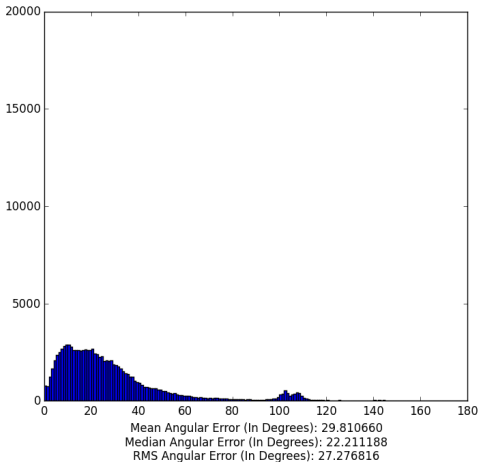
We have implemented the Rotation Averaging algorithms in Python. The source code only depends on a few Python packages (SciPy [Jones et al. 2001] and NetworkX [Hagberg et al. 2008]) and it is publicly available[1]. We have also integrated our implementation with the SfM Init package[2] [Wilson and Snavely 2014].

An initial guess for the Global Rotations is required to use these algorithms. To find the initial guess, we proceed as follows: Given the graph $G = (V, E)$, we assign to each edge (i, j) a weight $w(i, j) = d(R_{ij}, I)$, then we choose the edges of the Minimum Spanning Tree. Then, we select a random camera and we assign the Global Rotation for each vertex using a Depth-First Order approach.

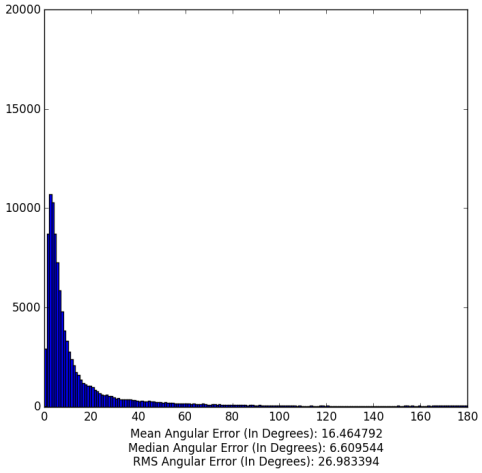
In our implementation, we have discovered that the algorithm has some numerical problems. For instance, if the input matrices are not quite rotation matrices, the conversions between $SO(3)$ and $\mathfrak{so}(3)$ and vice-versa will fail drastically. To solve this problem, we find the closest orthonormal matrix to the input matrix using the formula: $O = M(M^\intercal M)^{-\frac{1}{2}}$.

[1] https://github.com/RafaelMarinheiro/RotationAveraging

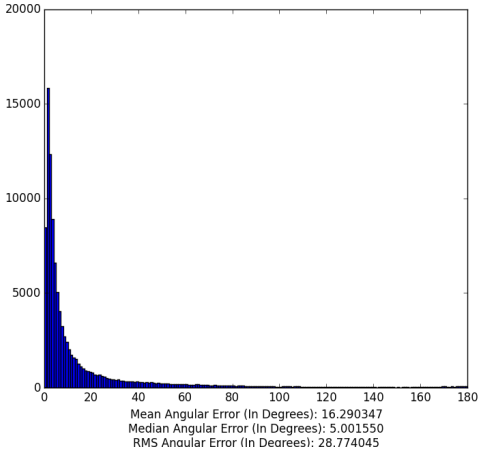
[2] https://github.com/RafaelMarinheiro/SfMInit



(a) *Initial guess after Spanning Tree*



(b) *After $\ell 1$RA*



(c) *After IRLS*

Figure 2: *Histogram of error after each part of the algorithm. The abscissa value represents the error in degrees. The closer to zero, the better.*

We have tested our implementation against the Notredame dataset published by [Wilson and Snavely 2014]. Figure (2) shows the error after each step of the algorithm. First, we run the Spanning Tree algorithm described earlier (see figure 2a), then we run only 5 iterations of the $\ell 1$RA algorithm (see figure 2b). After that, we use the output of the $\ell 1$RA algorithm as an input to the IRLS algorithm and then we run 20 iterations of it (see figure 2c). Notice that the error is indeed reduced after each step. The reported error metrics do match the results published both by [Chatterjee and Govindu 2013] and [Wilson and Snavely 2014].

5 Conclusion

In summary, we have successfuly implemented a Rotation Averaging algorithm. We have also made the code publicly available. Our implementation is easily extensible and it does not depend on non-free tools, so we hope that this will be a valuable tool for the research community.

References

CHATTERJEE, A., AND GOVINDU, V. 2013. Efficient and robust large-scale rotation averaging. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, 521–528.

HAGBERG, A. A., SCHULT, D. A., AND SWART, P. J. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, 11–15.

JONES, E., OLIPHANT, T., PETERSON, P., ET AL., 2001. SciPy: Open source scientific tools for Python. [Online; accessed 2014-12-10].

WILSON, K., AND SNAVELY, N. 2014. Robust global translations with 1dsfm. In *Computer Vision ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8691 of *Lecture Notes in Computer Science*. Springer International Publishing, 61–75.