Rotation Coordinate Descent for Fast Globally Optimal Rotation Averaging

Álvaro Parra[1]* Shin-Fang Chng[1]* Tat-Jun Chin[1] Anders Eriksson[2] Ian Reid[1]
[1] School of Computer Science, The University of Adelaide
[2] School of Information Technology and Electrical Engineering, University of Queensland

Abstract

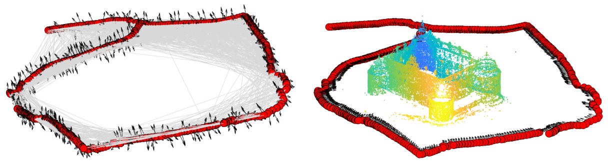
Under mild conditions on the noise level of the measurements, rotation averaging satisfies strong duality, which enables global solutions to be obtained via semidefinite programming (SDP) relaxation. However, generic solvers for SDP are rather slow in practice, even on rotation averaging instances of moderate size, thus developing specialised algorithms is vital. In this paper, we present a fast algorithm that achieves global optimality called rotation coordinate descent (RCD). Unlike block coordinate descent (BCD) which solves SDP by updating the semidefinite matrix in a row-by-row fashion, RCD directly maintains and updates all valid rotations throughout the iterations. This obviates the need to store a large dense semidefinite matrix. We mathematically prove the convergence of our algorithm and empirically show its superior efficiency over state-of-the-art global methods on a variety of problem configurations. Maintaining valid rotations also facilitates incorporating local optimisation routines for further speed-ups. Moreover, our algorithm is simple to implement [1].

1. Introduction

Rotation averaging, a.k.a. multiple rotation averaging [17] or $SO(3)$ synchronisation [4], is the problem of estimating absolute rotations (orientations w.r.t. a common coordinate system) from a set of relative rotation measurements. In vision and robotics, rotation averaging plays a crucial role in SfM [21, 25, 24, 8, 7, 20, 43, 36] and visual SLAM [5, 29, 33, 26, 19], in particular for initialising bundle adjustment. Fig. 1 illustrates the result of rotation averaging. With the increase in the size of SfM problems and continued emphasis on real-time visual SLAM, developing efficient rotation averaging algorithms is an active research area. In particular, real-world applications often give rise to problem instances with thousands of cameras.

*equal contribution

[1]Source code is available at `https://github.com/sfchng/Rotation_Coordinate_Descent`.



(a) Camera graph (b) Result

Figure 1. (a) Input camera graph from *Orebro Castle* [25] with $n = 761$ views and $116,589$ connections (relative rotations; grey lines). The initial absolute rotations (represented as black arrows) were randomly chosen. For visualisation, the ground truth positions were used to locate the cameras (red points). (b) Globally optimal absolute rotations computed from our RCD algorithm in 1.96 s (Shonan averaging [9] required 54.62 s on the same input). Note the alignment of the arrows along the path of the camera (the reconstructed point cloud is also plotted for visualisation).

The input to rotation averaging is a set of noisy relative rotations $\{\tilde{R}_{ij}\}$, where each $\tilde{R}_{ij}$ is a measurement of the orientation difference between cameras i and j which overlap in view. From the relative rotations, rotation averaging aims to recover the absolute rotations $\{R_i\}_{i=1}^n$ which represent the orientations of the cameras. In the ideal case where there is no noise in the relative rotations $\{R_{ij}\}$,

$$R_{ij} = R_j R_i^T. \tag{1}$$

The input relative rotations $\{\tilde{R}_{ij}\}$ define a *camera graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, n\}$ is the set of cameras, and $(i,j) \in \mathcal{E}$ is an edge in $\mathcal{G}$ if the relative rotation $\tilde{R}_{ij}$ between cameras i and j is measured. We assume a connected undirected graph $\mathcal{G}$, hence only $\tilde{R}_{ij}$ with $i < j$ needs to be considered. See Fig. 1(a) for an example camera graph.

Rotation averaging is usually posed as a nonlinear optimisation problem with nonconvex domain

$$\min_{R_1,\dots,R_n \in SO(3)} \sum_{(i,j)\in\mathcal{E}} d(R_j R_i^T, \tilde{R}_{ij})^p, \tag{2}$$

where $d : SO(3) \times SO(3) \mapsto \mathbb{R}$ is a distance function that measures the deviation from the identity (1) based on measured and estimated quantities. For example,

$$d_{\text{chordal}}(R, S) = \|R - S\|_F, \tag{3}$$

which is known as the chordal distance, and

$$d_{\angle}(R, S) = \| \log(RS^T) \|_2 \tag{4}$$

which is called the angular distance ($\log : SO(3) \mapsto \mathbb{R}^3$ is the logarithmic map in $SO(3)$ [17]). Also, usually $p = 1, 2$.

The general form of (2) can be challenging to solve [17, 38]. Earlier efforts devised locally convergent methods [14, 23, 15, 16, 35, 6, 17], e.g., IRLS [6] and the Weiszfeld algorithm [16], though most are not able to guarantee local correctness [38]. In contrast with local methods, convex relaxation methods which include linear method [22], spectral decomposition methods [3, 2] and semidefinite method [37, 31] solve a relaxed problem. However, their formulations are either prone to suboptimal solution or the deviation between the relaxed solution and the global solution is unknown. Tron *et al*. [36] surveyed and benchmarked approximate rotation averaging methods in the context of SfM. Recently, learning-based approaches [27] that can exploit the statistics of camera graphs have been developed.

1.1. Strong duality

Building upon empirical observations (e.g., [13]), Eriksson *et al*. [11] proved that the specific version

$$\min_{R_1,\ldots,R_n \in SO(3)} \sum_{(i,j)\in\mathcal{E}} d_{\text{chordal}}(R_j R_i^T, \tilde{R}_{ij})^2, \tag{5}$$

which is a standard formulation in the literature [16, 17, 11, 6], satisfies *strong duality* [30] under mild conditions on the noise of the input relative rotations (see [11, Eq. (22)] or the supp. material for details). This means that the global solution to (5) can be obtained by solving its Langrangian dual, which is a semidefinite program (SDP) (details in Sec. 2).

Our work focuses on solving the SDP relaxation of (5), especially for large-scale problems. Although SDPs are tractable, generic SDP solvers (e.g., conic optimisation [32]) can be slow on instances derived from rotation averaging. Thus, exploiting the problem structure to construct faster algorithms is an active research endeavour.

Eriksson *et al*. [11] presented a block coordinate descent (BCD) algorithm to solve the SDP relaxation, which consumed one order of magnitude less time than SeDuMi [32] on small to moderately sized instances ($n \leq 300$). The BCD algorithm maintains and iteratively improves a dense $3n \times 3n$ positive semidefinite (PSD) matrix by updating $3 \times 3n$ submatrices (called "block rows") until convergence. At convergence, each block row contains rotation matrices (up to correcting for reflection) which are the solution to (5) (the solution of different block rows differ by a gauge freedom; see Sec. 2.3). However, recent results [34, 9] suggest that BCD is still not practical for large-scale problems encountered in SfM and SLAM, where $n \geq 1000$.

1.2. Riemannian staircase methods

The Riemannian staircase framework [1] has been applied successfully to pose graph optimisation (PGO) or $SE(3)$ synchronisation, which aim to recover absolute camera poses (6 DOF) from measurements of relative rigid motion. Under this framework, Rosen *et al*. [28] presented SE-Sync for PGO which guarantees global optimality for moderate noise levels. Tian *et al*. [34] builds upon SE-Sync to solve PGO in a distributed optimisation setting targetting collaborative SLAM for multi-robot missions.

Recently, Dellaert *et al*. [9] adapted SE-Sync for rotation averaging. Their algorithm, called Shonan rotation averaging (henceforth, "Shonan") can globally solve the SDP relaxation of (5) through a chain of sub-problems on increasingly higher-dimensional domains $SO(d)$, with $d \geq 3$. A certification mechanism checks if the solution of each subproblem has reached global optimality by computing the minimum eigenvalue of a large $3n \times 3n$ matrix. While optimality is ensured for $d \leq 3n + 1$, in practice the algorithm only needs to expand d once or twice to reach optimality. Results show that Shonan was an order of magnitude faster than BCD on moderate size instances ($n \leq 200$) and was able to solve large-scale instances ($n \geq 1000$) that were not achievable by BCD with impressive runtimes (instances with $n = 5750$ could be solved in 115 seconds).

1.3. Our contributions

We propose a novel algorithm called *rotation coordinate descent (RCD)* to solve rotation averaging (5) globally optimally. Unlike BCD, RCD neither maintains a $3n \times 3n$ dense PSD matrix nor updates the matrix block row-by-block row. Instead, the operation of RCD is equivalent to directly updating the n rotation matrices $R_1, \ldots, R_n$, with provable convergence to global optimality. Moreover, since RCD maintains valid rotations at all times, local methods [6, 16] can be employed for further speed-ups.

We will present results which show that RCD can be *up to two orders of magnitude* faster than Shonan, depending on the structure of the camera graph $\mathcal{G}$. More specifically, RCD is comparable to Shonan for sparse $\mathcal{G}$ (*e.g*., SLAM camera graphs). However, RCD considerably outperforms Shonan on denser graphs (*e.g*., SfM camera graphs). This makes RCD a much more scalable algorithm.

On outliers An outlier in rotation averaging (2) is a measured relative rotation $\tilde{R}_{ij}$ that significantly deviates from the true value. Note that formulation (5), i.e., least sum of squared chordal distances, is non-robust. Thus, if there are outliers in the input, BCD, Shonan and RCD will fail, in the sense that they do not return results that closely resemble the "desired" solutions. In practice, such negative outcomes can be prevented by removing outliers with a preprocessing step [41, 10, 25]. We also emphasise that the theoretical validity of our work is not invalidated by the lack of robustness in the standard formulation (5) [16, 17, 6, 11].

Yang and Carlone [40] proposed a robust SDP relaxation for *single* rotation averaging, a special case where $n = 1$ (see [17]). The method has been demonstrated on relatively small scale problems (less than 100 measurements). [39, 18] addressed outliers on SDP based PGO though without providing global optimality guarantees.

2. Preliminaries

2.1. Notation

We operate on block matrices composed of 3×3 blocks (submatrices in $\mathbb{R}^{3\times 3}$). A block matrix is represented with a capital letter, e.g., $A \in \mathbb{R}^{3m\times 3n}$, and element (i,j) of a block matrix, denoted $A_{i,j}$, is the submatrix formed by rows $3(i-1)+1$ to $3(i-1)+3$ and columns $3(j-1)+1$ to $3(j-1)+3$ of A. Thus, $A_{i,i}$ are diagonal blocks.

We also define the k-th "row" of A as the submatrix

$$A_{k,:} = [A_{k,1} A_{k,2} \cdots A_{k,n}] \in \mathbb{R}^{3\times 3n} \tag{6}$$

and similarly for the k-th "column" of A. If A has a single block column, we call it a "vector". We use the notation

$$A_{(a:b);(c:d)} = \begin{bmatrix} A_{a,c} & \cdots & A_{a,d} \\ \vdots & & \vdots \\ A_{b,c} & \cdots & A_{b,d} \end{bmatrix} \in \mathbb{R}^{3m\times 3n} \tag{7}$$

for the submatrix of A from rows a to b and columns c to d. If A is a vector we use the notation $A_k = A_{k,1}$ and $A_{a:b} = A_{(a:1);(b:1)}$.

We denote the 3×3 identity and zero matrices as I_3 and 0_3, and the trace and Moore–Penrose pseudoinverse of a matrix M as $\mathrm{tr}(M)$ and $M^\dagger$, respectively.

2.2. SDP relaxation

We first present the SDP relaxation of (5) following [11]. By rewriting the chordal distance using trace, (5) becomes

$$\min_{R_1,\ldots,R_n \in SO(3)} - \sum_{(i,j)\in\mathcal{E}} \mathrm{tr}(R_j^T \tilde{R}_{ij} R_i). \tag{8}$$

This can be further written more compactly as

$$\min_{R \in SO(3)^n} - \mathrm{tr}(R^T \tilde{R} R) \tag{P}$$

using matrix notations, where

$$R = \left[R_1^T\, R_2^T \cdots R_n^T\right]^T \in SO(3)^n \tag{9}$$

contains the target variables, and $\tilde{R}$ is the $3n \times 3n$ block symmetric matrix with upper-triangle elements (i,j) equal to $\tilde{R}_{ij}^T$ if $(i,j) \in \mathcal{E}$ and 0_3 otherwise (diagonal elements are 0_3's). Problem (P) is called the primal problem.

Algorithm 1 Block coordinate descent (BCD) for (DD).

```
Require: $\tilde{R}$ and $Y^{(0)} \succeq 0$.
 1: $t \leftarrow 0$.
 2: repeat
 3:   Select an integer $k$ in the interval $[1, n]$.
 4:   $W \leftarrow$the $k$-th column of $\tilde{R}$.
 5:   $Z \leftarrow Y^{(t)} W$.
 6:   $S \leftarrow Z \left[ (W^T Z)^{\frac{1}{2}} \right]^\dagger$.
 7:   $Y^{(t+1)} \leftarrow \begin{bmatrix} Y^{(t)}_{(1:k-1);(1:k-1)} & S_{1:(k-1)} & Y^{(t)}_{(1:k-1);(k+1:n)} \\ S^T_{1:(k-1)} & I_3 & S^T_{(k+1):n} \\ Y^{(t)}_{(k+1:n);(1:k-1)} & S_{(k+1):n} & Y^{(t)}_{(k+1:n);(k+1:n)} \end{bmatrix}$
 8:   $t \leftarrow t + 1$.
 9: until convergence
10: return $Y^* = Y^{(t)}$.
```

As derived in Eriksson *et al.* [11], the dual of the Lagrangian dual of (P) is the SDP relaxation

$$\min_{Y \in \mathbb{R}^{3n\times 3n}} \quad - \mathrm{tr}(\tilde{R} Y) \tag{DD}$$
$$\text{s.t.} \quad Y_{i,i} = I_3,\ i = 1,\ldots,n. \tag{10a}$$
$$Y \succeq 0, \tag{10b}$$

where Y is a $3n \times 3n$ PSD matrix, and $Y_{i,i}$ is the i-th diagonal block of Y. The interested reader is referred to Eriksson *et al.* for the detailed derivations. It is proven that, under mild conditions (see supp. material), that

$$-tr(\tilde{R} Y^*) = -\mathrm{tr}({R^*}^T \tilde{R} R^*), \tag{11}$$

where R^* and Y^* are respectively the optimisers of (P) and (DD), i.e., zero duality gap between (P) and (DD).

Output rotations Note that constraint (10a) in (DD) merely enforces orthogonality in each diagonal block. Hence, in general a feasible Y for (DD) is *not* factorisable as the product of two rotation matrices RR^T. It can be shown, however, that the optimiser Y^* of (DD) is rank-3 [11], which admits the factorisation

$$Y^* = Q^* {Q^*}^T, \tag{12}$$

where $Q^* \in O(3)^n$ contains n 3×3 orthogonal matrices. To obtain R^*, first Q^* is obtained via SVD on Y^*, then for each Q_i^* whose determinant is negative, the sign of the Q_i^* is flipped to positive to yield a valid rotation.

2.3. Block coordinate descent

Algorithm 1 presents BCD [11] for (DD) using our notation, which also includes a minor improvement to the original. Specifically, instead of working on an auxiliary square matrix obtained by removing the k-th row and column from

$Y^{(t)}$ (see [11, Step 3 of Algorithm 1]), we directly operate over $Y^{(t)}$ and create a temporary block vector Z (Line 5). Since Z is smaller than the auxiliary square matrix, the efficiency of Line 6 which requires operating over Z twice is marginally improved. We emphasise that Algorithm 1 is intrinsically the same as the original (see supp. material for details and validity of the improvement).

The PSD matrix Y can be initialised as an arbitrary PSD matrix. A simple choice is setting $R_i = I_3$ for all i in R and initialising $Y^{(0)} = RR^T$. However, we remind again that the subsequent $Y^{(t)}$ are not factorisable as the product of rotations $R^{(t)}{R^{(t)}}^T$ in general; see Sec 2.2.

Gauge freedom Note that the factorisation (12) is up to an arbitrary orthogonal transformation $G \in O(3)$, i.e.,

$$Y^* = Q^* {Q^*}^T = (Q^* G)(Q^* G)^T. \tag{13}$$

We say that G represents a "gauge freedom" in the solution. This leads to another approach to retrieve R^* from Y^*, which recognises that the columns (and rows) of Y^* are related by orthogonal transformations as Y^* is rank-3 with diagonal elements equal to I_3. Thus, for any two columns k and k' in Y^*, there exists an orthogonal transformation $G_{k,k'} \in O(3)$ such that

$$Y^*_{:,k'} = Y^*_{:,k}\, G_{k,k'}. \tag{14}$$

Hence, $G_{k,k'}$ must transform the k'-th element of $Y^*_{:,k}$ to I_3 (i.e., $Y^*_{k',k} G_{k,k'} = I_3$). Therefore

$$G_{k,k'} = (Y^*_{k',k})^T = Y^*_{k,k'} \tag{15}$$

as columns in Y^* are orthogonal and Y^* is symmetric.

The set of transformations relating columns (15)

$$\mathscr{G} = \{G_{k,k'}, \text{ for all } k, k' = 1, \ldots, n\} \subset O(3) \tag{16}$$

corresponds to an special case of gauge freedom. Since all columns in Y^* are up to some transformation in $\mathscr{G}$ to another column, we can take any as R^*; the choice will depend on selecting one of the cameras as the reference frame, i.e., which camera takes $R_i^* = I_3$.

3. Rotation coordinate descent

In this section, we will describe our novel method called *rotation coordinate descent (RCD)*, summarised in Algorithm 2. While seemingly a minor modification to BCD, RCD is based on nontrivial insights (Sec. 3.1). More importantly, a major contribution is to mathematically prove the global convergence of RCD (Sec. 3.2). Another fundamental advantage is that since RCD maintains valid rotations throughout the iterations (in contrast to BCD; see Sec. 2.3), it can exploit local optimisation routines for (P) to speed-up convergence (Sec. 4). As the results will show (Sec. 5), our approach can be up to two orders of magnitude faster than Shonan [9], which is the state of the art for (DD).

Algorithm 2 Rotation coordinate descent (RCD) for (DD).

```
Require: $\tilde{R}$ and $R^{(0)}$.
 1: $t \leftarrow 0$.
 2: repeat
 3:     Select an integer $k$ in the interval $[1, n]$.
 4:     $W \leftarrow$the $k$-th column of $\tilde{R}$.
 5:     $Z \leftarrow R^{(t)}({R^{(t)}}^T W)$.
 6:     $S \leftarrow Z\left[(W^T Z)^{\frac{1}{2}}\right]^{\dagger}$.
 7:     $Q^{(t+1)} \leftarrow \left[(S_{1:(k-1)})^T \;\; I_3 \;\; (S_{(k+1):n})^T\right]^T$.
 8:     $R^{(t+1)} \leftarrow$ Flip determinants over $Q^{(t+1)}$ (if needed)
        to ensure rotations.
 9:     $t \leftarrow t + 1$.
10: until convergence
11: return $Y^* = R^{(t)}{R^{(t)}}^T$.
```

3.1. Main ideas

As summarised in Algorithm 1, BCD requires to maintain and operate on a large dense PSD matrix $Y \in \mathbb{R}^{3n \times 3n}$. While the values of each update can be computed in constant time (specifically, SVD of a 3×3 matrix; Line 6), manipulating Y is unwieldy. Specifically, Line 5 performs

$$Z = Y^{(t)} W \tag{17}$$

to obtain temporary vector $Z \in \mathbb{R}^{3n \times 3}$ from a subset of the measurements $W \in \mathbb{R}^{3n \times 3}$, which costs

$$27n^2 \text{ multiplications} \equiv \mathcal{O}(n^2). \tag{18}$$

This quadratic dependence on n makes BCD slow on large-scale SfM or SLAM problems [34], e.g., where $n \geq 1000$, as will be demonstrated in Sec. 5.

Although the PSD matrix Y of (DD) has size $3n \times 3n$, the "effective" variables are only $3n$ given that Y^* is rank-3. Our key insight comes from the gauge freedom of Y^* (Sec. 2.3) implying that any row of Y^* provides a valid solution for R^*. Choosing the k-th row implies choosing the k-th camera as the reference frame, i.e., $R_k = I_3$. Based on this insight, we devised RCD to maintain only the effective variables $R^{(t)}$. Each iteration executes what amounts to updating a single column of Y; specifically, in Line 3, a camera k is chosen as the reference frame then set the k-th element of $Q^{(t+1)}$ as I_3 in Line 7 ($Q^{(t)}$ contains orthogonal matrices). Then, in Line 6 the other elements of $Q^{(t+1)}$ are updated via the same explicit form of BCD. To ensure keeping rotations elements during iterations, the sign of the orthogonal elements in $Q^{(t+1)}$ is flipped if negative in Line 9 to produce $R^{(t+1)}$.

Maintaining and updating only $R^{(t)}$ provides immediate computational savings; in Line (5) obtaining the intermediate vector Z is now accomplished as

$$Z = R^{(t)} \underbrace{\left({R^{(t)}}^T W\right)}_{\text{Compute this first}}, \tag{19}$$

which costs

$$27n + 27n \text{ multiplications} \equiv \mathcal{O}(n) \tag{20}$$

and has only linear dependence on n. The next section proves the important result that this computational savings does not come at the expense of global optimality.

3.2. Global convergence of RCD

As proven in [11, 12], Algorithm 1 monotonically decreases the objective $-\operatorname{tr}(\tilde{R}Y)$ at each iteration from any feasible initialisation. Our strategy for proving the global convergence of RCD is to show that updating the variables at each iteration t of Algorithm 2, i.e.,

$$R^{(t)} \to R^{(t+1)}, \tag{21}$$

has an effect on $-\operatorname{tr}(\tilde{R}Y)$ that is equivalent to one iteration of Algorithm 1 initialised with

$$Y^{(0)} = R^{(t)}{R^{(t)}}^T. \tag{22}$$

If this equivalence can be established, Algorithm 2 also provably monotonically decreases $-\operatorname{tr}(\tilde{R}Y)$ and will converge to the optimiser Y^* of (DD).

To this end, we will first show (Corollary 3.2.1) that one iteration of Algorithm 1 initialised with (22) produces a PSD matrix $Y^{(1)}$ that is factorisable as

$$Y^{(1)} = R^{(1)}{R^{(1)}}^T. \tag{23}$$

Without loss of generality, we take $k = 1$ (the updated row and column in BCD during the iteration) and define $R^{(1)}_{\text{BCD}}$ as the first column of $Y^{(1)}$, i.e.,

$$R^{(1)}_{\text{BCD}} = Y^{(1)}_{:,1}. \tag{24}$$

Then, we will prove that $R^{(t+1)} = R^{(1)}_{\text{BCD}}$ (Theorem 3.3). From Line 7 in Algorithm 1, $Y^{(1)}$ can be written as

$$Y^{(1)} = \begin{bmatrix} I_3 & X^{*T} \\ X^* & B \end{bmatrix}, \tag{25}$$

where $B = Y^{(0)}_{(2:n);(2:n)}$ is the unchanged sub-matrix during the iteration $Y^{(0)} \to Y^{(1)}$, and $X^* \in \mathbb{R}^{3(n-1)\times 3}$ contains the updated values. From [11, 12], X^* is the optimiser of the following SDP problem:

$$\min_{X\in\mathbb{R}^{3(n-1)\times 3}} \quad -\operatorname{tr}(C^T X) \tag{26a}$$

$$\text{s.t.} \quad \begin{bmatrix} I_3 & X^T \\ X & B \end{bmatrix} \succeq 0, \tag{26b}$$

where $C \in \mathbb{R}^{3(n-1)\times 3}$ is equal to W as in Line 4 in Algorithm 1 but without the k-th element (which is zero).

Note that the optimal PSD matrix in Problem (26) is $Y^{(1)}$ (25). The goal of Problem (26) is to find the optimal update X^* to produce $Y^{(1)}$ that remains feasible (constraint (26b)).

Theorem 3.1. *Problem* (26) *is a special case of* (DD).

Proof. Consider the instance of Problem (DD) with

$$\tilde{R} = \begin{bmatrix} 0_3 & C^T \\ C & 0 \end{bmatrix}. \tag{27}$$

We first show that a feasible PSD matrix in Problem (26)

$$Y = \begin{bmatrix} I_3 & X^T \\ X & B \end{bmatrix} \tag{28}$$

is a feasible solution in (DD). From the initialisation of $Y^{(0)}$ in (22), $B = R^{(t)}_{2:n}{R^{(t)}_{2:n}}^T$; hence, all diagonal elements in Y (28) are identities which fulfill the first constraint (10a) in (DD). From (26b), $Y \succeq 0$, which is the second constraint (10b) in (DD).

We now show the objective of (DD) with $\tilde{R}$ from (27) is equivalent to the objective in Problem (26). The objective of (DD) becomes

$$\begin{aligned} -\operatorname{tr}(\tilde{R}Y) &= -\operatorname{tr}\left(\begin{bmatrix} C^T X & C^T B \\ C & C X^T \end{bmatrix}\right) && (29a)\\ &= -\operatorname{tr}\left(C^T X\right) - \operatorname{tr}\left(C X^T\right) && (29b)\\ &= -2\operatorname{tr}\left(C^T X\right) && (29c) \end{aligned}$$

which is twice to the objective of (26). Thus, Problem (26) is a special case of (DD) since any feasible solution of (26) is also feasible in (DD), and both objectives are equivalent. □

Lemma 3.2. *The optimal PSD matrix of Problem* (26) *admits the factorisation*

$$Y^{*(1)} = R^{*(1)}{R^{*(1)}}^T. \tag{30}$$

Proof. Problem (26) is a special case of (DD) (Theorem 3.1) $\implies$ the optimal PSD matrix of Problem (26) also admits the factorisation (30). □

Corollary 3.2.1. *Lemma 3.2 validates* (23) *as BCD optimally solves Problem* (26) *(Lines 5–7 in Algorithm 1) [11, 12]. Thus, Algorithm 2 preserves the rank-3 factorisation during iterations, and also Algorithm 1 if initialised with a rank-3 SDP matrix $R^{(0)}{R^{(0)}}^T$ with $R^{(0)} \in SO(3)^n$.*

Theorem 3.3. $R^{(t+1)} = R^{(1)}_{BCD}$

Proof. The equality is by construction of Algorithm 2. From the definition of $R^{(1)}_{\text{BCD}}$ in (22), $R^{(1)}_{\text{BCD}}$ is the first column in $Y^{(1)}$ (25), i.e., $R^{(1)}_{\text{BCD}} = \begin{bmatrix} I_3 & X^{*T} \end{bmatrix}^T$, where X^* is

S (in Line 6, Algorithm 1) without the k-th element (see sup. material for details). Lines 3–6 in Algorithm 1 are the same as Lines 3–6 in Algorithm 2 except on obtaining Z, which takes the same value since from the initialisation (22) of $Y^{(0)}$ in Algorithm 1, $Z = Y^{(0)}W = R^{(t)}({R^{(t)}}^T W)$ is equal to Z as obtained in Algorithm 2 $\implies$ also $R^{(t+1)} = \begin{bmatrix} I_3 & X^{*T} \end{bmatrix}^T$. $\square$

4. Speeding up RCD with local optimisation

Since Algorithm 2 iterates over $SO(3)^n$, local methods for (P) can be directly used to speedup convergence of Algorithm 2. Contrast this to BCD that updates a PSD matrix from where, in general, valid rotations can be retrieved only at convergence. Algorithm 3 integrates a local method (Line 13) that we design from experimental observations:

1. Substantial reductions in the objective often occur after n iterations. We call it an *epoch* and we ensure we sample all k's during each epoch (Line 4).
2. In practice, one iteration of Algorithm 2 takes $\approx 0.02\%$ of the runtime of solving a local optimisation instance. Thus, Algorithm 3 invokes local optimisation and check for convergence only after completing epochs.
3. Local optimisation produces more drastic "jumps" in the objective at earlier iterations. Thus, Algorithm 3 delays local optimisation when the local method fails on reducing the objective (Line 17).

To demonstrate the effect of local optimisation on the convergence of RCD, Fig. 2 plots the objective value for RCD and RCDL at increasing epochs on the input graph *torus* [5] with $n = 5000$ cameras (see Table 1 in Sec. 5 for more details). During the 1st epoch, the local algorithm drastically reduced the objective (from stage in *green* to stage in *magenta*). This "jump" of the objective value reveals the collaborative strength of global and local methods, which enabled RCDL to converge in much fewer epochs (*red* stage) compared to RCD (*blue* stage).

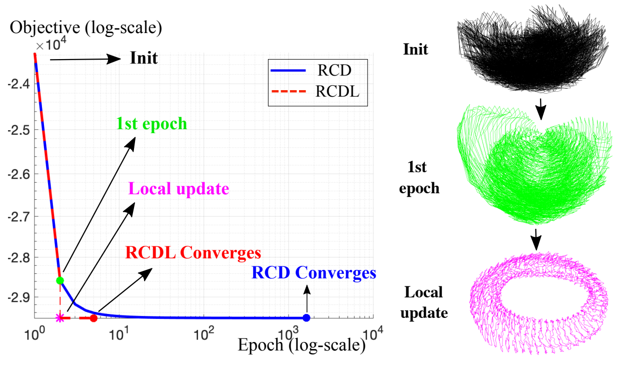
5. Experiments

We benchmarked the following algorithms over a variety of synthetic and real-world camera graph inputs: Algorithm 1 (BCD), Algorithm 2 (RCD), Algorithm 3 (RCDL) with local optimisation routine adapted from [26], and Shonan [9] (SA). We implemented all the optimisation routines in C++ except for SA for which we used the author's implementation (which also has optimisation routines in C++ [2]). As we measured runtime as the total algorithm time, we include the certification times in SA. All experiments are executed on a standard machine with an Intel Core i5 2.3 GHz CPU and 8 GB RAM.

[2]https://github.com/dellaert/ShonanAveraging

Algorithm 3 RCD with local optimisation (RCDL).

```
Require: $\tilde{R}$ and $R^{(0)}$.
 1: $t \leftarrow 0$, $e \leftarrow 0$, $s \leftarrow 0$
 2: repeat
 3:   for $i = 1, \dots, n$ do
 4:     Select an integer $k$ in the interval $[1, n]$ w/o rep.
 5:     $W \leftarrow$ the $k$-th column of $\tilde{R}$.
 6:     $Z \leftarrow R^{(t)}({R^{(t)}}^T W)$.
 7:     $S \leftarrow Z \left[ (W^T Z)^{\frac{1}{2}} \right]^{\dagger}$.
 8:     $R^{(t+1)} \leftarrow \left[ (S_{1:(k-1)})^T \;\; I_3 \;\; (S_{(k+1):n})^T \right]^T$.
 9:     $t \leftarrow t + 1$.
10:   end for
11:   if ( $s = 0$ or MOD$(e, s) = 0$) then
12:     $R^{(t)} \leftarrow$ Flip determinants over $R^{(t)}$ (if needed) to
        ensure rotations.
13:     $\hat{R} \leftarrow$ local method with initial estimate $R^{(t)}$.
14:     if $-\mathrm{tr}(\hat{R}^T \tilde{R} \hat{R}) < -\mathrm{tr}({R^{(t)}}^T \tilde{R} R^{(t)})$ then
15:       $R^{(t)} \leftarrow \hat{R}$.
16:     else
17:       $s \leftarrow s + 2$
18:     end if
19:   end if
20:   $e \leftarrow e + 1$.
21: until convergence
```



(a) (b)

Figure 2. Evolution of RCD and RCDL on the large-scale SLAM instance *torus* [5] with $n = 5000$ cameras. (a) Evolution of the objectives. (b) Camera poses from RCDL. A *single local update* was able to produce a visually correct solution.

Graph density Consider a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ vertices and $m = |\mathcal{E}|$ edges. Define

$$d_{\mathcal{G}} := (|\mathcal{E}| - |\mathcal{E}_{\min}|)/(|\mathcal{E}_{\max}| - |\mathcal{E}_{\min}|)\,, \qquad (31)$$

as the density of graph $\mathcal{G}$, where $\mathcal{E}_{\max}$ and $\mathcal{E}_{\min}$ denote the set of edges of the complete $(\mathcal{V}, \mathcal{E}_{\max})$ and the cycle graph $(\mathcal{V}, \mathcal{E}_{\min})$. Excluding graphs with $n - 1$ edges[3], d_G takes

[3]Rotation averaging instances are typically overdetermined, i.e., problems with $|\mathcal{E}| > n - 1$ edges.

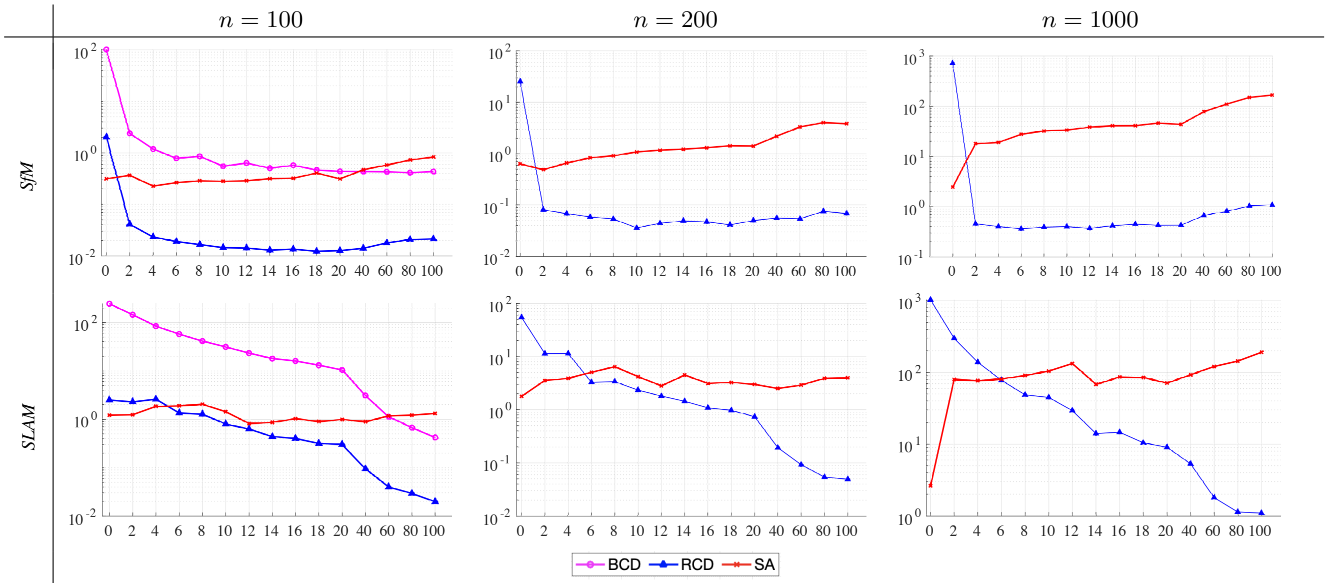


Figure 3. Runtime [s] (y-axis in log-scale) at varying graph densities $d_\mathcal{G}$ (x-axis in $\times 10^{-2}$) for SfM and SLAM graphs with $n = 100, 200, 1000$ cameras. We denser sampled the interval $[0, 0.2]$.

values in $[0, 1]$. Thus, by definition (31), $d_\mathcal{G} = 0$ for a cycle graph and $d_\mathcal{G} = 1$ for a complete graph.

5.1. Synthetic Data

To test RCD over a variety of graph configurations, we synthesised graphs with varying densities to simulate SfM and SLAM problems. As SfM often solves reconstruction from views with large baselines, we generated random camera positions and random connections in the SfM setting. In contrast, for the SLAM setting, we simulated views with a smooth trajectory and connect only nearby views. We created measurements of relative rotations (1) by multiplying the ground truth relative rotations with rotations with random axes and angles normally distributed with $\sigma = 0.1$ rad. For a fair comparison, we initialised all methods with the same initial random absolute rotations.

Varying graph densities Fig. 3 shows the runtimes averaged over 10 runs for all methods. RCD significantly outperformed SA for $d_\mathcal{G} > 0.1$. In general, camera graphs from real-world SfM datasets are often dense. See for example $d_\mathcal{G}$ values for the real-world instances in Table 2 with average $d_\mathcal{G} \approx 0.53$. For larger problems, BCD was not able to terminate within reasonable time ($\leq$ 1000s); we did not report results for BCD for $n > 100$. Although RCD was not considerably faster than SA when $d_\mathcal{G} < 0.04$, the convergence rate can be accelerated by using a local optimisation routine as we show in Sec. 5.2.

Varying noise levels and number of cameras In Fig. 4, we plotted the runtimes of RCD and SA on SfM camera graphs with varying noise levels σ, number of cameras n, but with fixed $d_\mathcal{G} = 0.4$. We omitted the comparison

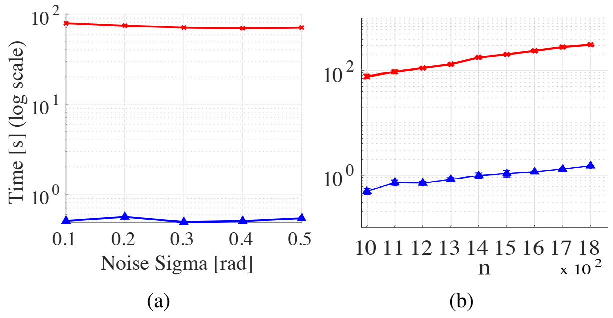


Figure 4. Runtime [s] (in log-scale) for SfM camera graphs with $d_\mathcal{G} = 0.4$. (a) Varying σ in $[0.1, 0.5]$ rad. and $n = 1000$. (b) Varying n in $[1000, 1800]$ with $\sigma = 0.1$ rad.

against BCD as it did not converge within a sensible time for large problems, as demonstrated in Fig. 3. Fig. 4(a) shows that runtimes for RCD and SA were marginally affected by noise. Fig. 4(b) shows that RCD outperformed SA by two orders of magnitude ($1.5s$ vs $312.9s$ at $n = 1,800$) — this further demonstrates the superior scalability of RCD.

We repeated the above experiment with $d_\mathcal{G} = 0.2$ which simulates SLAM graphs; see supp. material for the results.

5.2. SLAM benchmark dataset

We compared runtimes on large-scale problems from the SLAM dataset in [5]. Table 1 reports the input characteristics of each benchmarking instance and the results for all methods. Here, we initialised all algorithms with the same initial rotations from a random spanning tree. Note that initialisation does not affect the global optimality of tested algorithms. The spanning tree initialisation is fast and practical. We remark that in real-world applications it is unnec-

Dataset characteristics				Error [%]				Efficiency					
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$						# Epoch		Time [s]			Speedup
Name	n	m	$d_{\mathcal{G}}$	Init.	RCD	RCDL	SA	RCD	RCDL	RCD	RCDL	SA	
smallgrid	125	297	0.02200	-16.13	0	-4.77E-09	-8.38E-05	46	10	0.07	**0.02**	0.06	2.7
garage	1661	6275	0.00340	-7.29E-05	-3.63E-06	0	-1.40E-07	29	2	3.74	**0.28**	4.76	17.1
sphere	2500	4949	0.00078	-1.70	-6.24E-06	0	-7.84E-07	352	2	105.70	**0.66**	17.07	25.7
torus	5000	9898	0.00039	-20.95	-2.55E-05	0	-1.73E-06	1620	4	1808.86	**4.86**	15.76	3.2
grid3D	8000	22819	0.00046	-15.41	-4.54E-06	0	-2.14E-06	409	4	1199.30	**14.78**	23.93	1.6

Table 1. Quantitative results for the SLAM Benchmark dataset [5]. Error of the initial solution (Init.) and each method is the % of its objective w.r.t. the lowest obtained objective among all methods. One epoch is equivalent to n iterations as described in Sec. 4. Speedup is presented for the best result of RCD and RCDL against SA.

Dataset characteristics				Error [%]				Efficiency					
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$						# Epoch		Time [s]			Speedup
Name	n	m	$d_{\mathcal{G}}$	Init.	RCD	RCDL	SA	RCD	RCDL	RCD	RCDL	SA	
Alcatraz Tower	172	14706	1.00	-0.66	-8.66E-10	0	-4.64E-08	4	2	**0.03**	0.12	0.63	25.3
Doge Palace	241	19753	0.68	-0.89	-1.16E-08	0	-1.04E-07	8	2	**0.09**	0.21	1.00	11.1
King's College	328	41995	0.78	-1.57	-5.01E-10	0	-3.81E-08	7	2	**0.11**	0.60	2.37	21.5
Alcatraz Garden	419	51635	0.59	-1.29	-7.70E-09	0	-5.57E-08	11	2	**0.24**	0.89	3.24	13.5
Linkoping	538	34462	0.24	-1.22	-3.62E-07	0	-4.03E-06	37	2	0.90	**0.43**	6.44	15.0
UWO	692	80301	0.33	-1.26	-1.38E-07	0	-7.88E-07	20	2	**0.85**	1.65	11.12	13.1
Orebro Castle	761	116589	0.40	-1.19	-9.40E-08	0	-1.04E-06	20	2	**1.10**	4.01	26.17	23.8
Spilled Blood	781	117814	0.39	-2.81	-3.20E-08	0	-6.61E-07	14	2	**0.79**	4.32	37.64	47.6
Lund Cathedral	1207	177289	0.24	-1.16	-9.62E-07	0	-1.91E-06	78	2	8.45	**7.03**	41.08	5.8
San Marco	1498	757037	0.67	-0.74	-6.60E-09	0	-8.97E-09	6	2	**1.61**	145.46	110.07	68.4

Table 2. Quantitative results for the SfM large scale real-world dataset [25]. See Table 1 for the description of each column.

essary to solve camera orientations from random rotations. We provided the errors (in %) of resulting objective (including the initialisation) relative to the lowest objective value reported among all methods.

Camera graphs are very sparse for the SLAM Benchmark in Table 1 ($d_{\mathcal{G}} \leq 0.022$). Although RCD was not as fast as SA on *sphere*, *torus* and *grid3D* (note that $d_{\mathcal{G}} \leq 0.0007$ in those instances), the use of a local optimisation in RCDL permitted to outperform SA; see also Fig. 2.

5.3. Real-world SfM dataset

Table 2 presents runtimes over real-world SfM datasets [25] where RCD outperformed SA; see the supp. material for errors in °. We remark that RCDL took substantially fewer epochs compared to RCD to converge. However, RCDL did not achieve a better runtime as local optimisation consumed on average ≈ 90% of the total runtime, especially for large graph densities. Fig. 5 shows the reconstructed *Spilled Blood* using the estimated camera orientations of RCDL after running for *one* epoch in 4.2s.

Acknowledgements

This research was supported by the Australian Research Council through the grants ARC DP200101675, ARC FT170100072, and the ARC Centre of Excellence for Robotic Vision CE140100016.

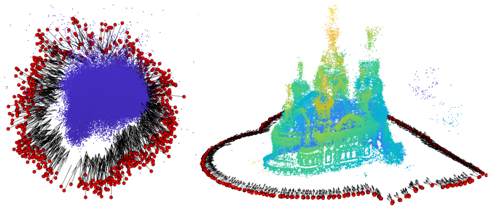


Figure 5. Reconstruction of the *Spilled Blood Cathedral* by solving the *known-rotation-problem* [42]. *Left*: Initial camera orientations. *Right*: Result from RCDL after 1 epoch.

6. Conclusions

We present RCD, a fast rotation averaging algorithm that finds the globally optimal rotations under mild conditions on the noise level of the measurements. Our insights on gauge freedom has circumvented the quadratic computational burden of BCD, which is an established method for global rotation averaging. Also, since RCD maintains valid rotations instead of a dense PSD matrix, local optimisation routines can be beneficially integrated. Experimental results demonstrated the superior efficiency of RCD, which significantly outperformed state-of-the-art algorithms on a variety of problem configurations.

References

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009. 2

[2] Federica Arrigoni, Beatrice Rossi, Pasqualina Fragneto, and Andrea Fusiello. Robust synchronization in so (3) and se (3) via low-rank and sparse matrix decomposition. *CVIU*, 174:95–113, 2018. 2

[3] Federica Arrigoni, Beatrice Rossi, and Andrea Fusiello. Spectral synchronization of multiple views in $SE(3)$. *SIAM SIMAX*, 9(4):1963–1990, 2016. 2

[4] Nicolas Boumal, Amit Singer, P-A Absil, and Vincent D Blondel. Cramér-Rao bounds for synchronization of rotations. *Information and Inference: A Journal of the IMA*, 3(1):1–39, 2014. 1

[5] Luca Carlone, Roberto Tron, Kostas Daniilidis, and Frank Dellaert. Initialization techniques for 3D SLAM: a survey on rotation estimation and its use in pose graph optimization. In *IEEE ICRA*, 2015. 1, 6, 7, 8

[6] Avishek Chatterjee and Venu Madhav Govindu. Efficient and robust large-scale rotation averaging. In *ICCV*, 2013. 2, 3

[7] Hainan Cui, Xiang Gao, Shuhan Shen, and Zhanyi Hu. HSfM: hybrid structure-from-motion. In *CVPR*, 2017. 1

[8] Zhaopeng Cui and Ping Tan. Global structure-from-motion by similarity averaging. In *ICCV*, 2015. 1

[9] Frank Dellaert, David M Rosen, Jing Wu, Robert Mahony, and Luca Carlone. Shonan rotation averaging: Global optimality by surfing $SO(p)^n$. In *ECCV*, 2020. 1, 2, 4, 6

[10] Olof Enqvist, Fredrik Kahl, and Carl Olsson. Non-sequential structure from motion. In *ICCVW*, 2011. 2

[11] Anders Eriksson, Carl Olsson, Fredrik Kahl, and Tat-Jun Chin. Rotation averaging and strong duality. In *CVPR*, 2018. 2, 3, 4, 5

[12] Anders Eriksson, Carl Olsson, Fredrik Kahl, and Tat-Jun Chin. Rotation averaging with the chordal distance: Global minimizers and strong duality. *IEEE TPAMI*, 2019. 5

[13] Johan Fredriksson and Carl Olsson. Simultaneous multiple rotation averaging using Lagrangian duality. In *ACCV*, 2012. 2

[14] Venu Madhav Govindu. Combining two-view constraints for motion estimation. In *CVPR*, 2001. 2

[15] Venu Madhav Govindu. Lie-algebraic averaging for globally consistent motion estimation. In *CVPR*, 2004. 2

[16] Richard Hartley, Khurrum Aftab, and Jochen Trumpf. L1 rotation averaging using the Weiszfeld algorithm. In *CVPR*, 2011. 2, 3

[17] Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *IJCV*, 103(3):267–305, 2013. 1, 2, 3

[18] Pierre-Yves Lajoie, Siyi Hu, Giovanni Beltrame, and Luca Carlone. Modeling perceptual aliasing in slam via discrete–continuous graphical models. *IEEE RAL*, 4(2):1232–1239, 2019. 3

[19] Xinyi Li and Haibin Ling. Hybrid camera pose estimation with online partitioning for SLAM. *IEEE RAL*, 5(2):1453–1460, 2020. 1

[20] Alex Locher, Michal Havlena, and Luc Van Gool. Progressive structure from motion. In *ECCV*, 2018. 1

[21] Daniel Martinec and Tomas Pajdla. Robust rotation and translation estimation in multiview reconstruction. In *CVPR*, 2007. 1

[22] Daniel Martinec and Tomas Pajdla. Robust rotation and translation estimation in multiview reconstruction. In *CVPR*, 2007. 2

[23] Maher Moakher. Means and averaging in the group of rotations. *SIAM SIMAX*, 24(1):1–16, 2002. 2

[24] Pierre Moulon, Pascal Monasse, and Renaud Marlet. Global fusion of relative motions for robust, accurate and scalable structure from motion. In *ICCV*, 2013. 1

[25] Carl Olsson and Olof Enqvist. Stable structure from motion for unordered image collections. In *Springer SCIA*, 2011. 1, 2, 8

[26] Álvaro Parra, Tat-Jun Chin, Anders Eriksson, and Ian Reid. Visual SLAM: Why bundle adjust? In *IEEE ICRA*, 2019. 1, 6

[27] Pulak Purkait, Tat-Jun Chin, and Ian Reid. NeuRoRA: Neural robust rotation averaging. In *ECCV*, 2020. 2

[28] David M Rosen, Luca Carlone, Afonso S Bandeira, and John J Leonard. SE-Sync: A certifiably correct algorithm for synchronization over the special Euclidean group. *IJRR*, 38(2-3):95–125, 2019. 2

[29] David M Rosen, Charles DuHadway, and John J Leonard. A convex relaxation for approximate global optimization in simultaneous localization and mapping. In *IEEE ICRA*, 2015. 1

[30] Andrzej Ruszczynski. *Nonlinear optimization*. Princeton university press, 2011. 2

[31] Amit Singer. Angular synchronization by eigenvectors and semidefinite programming. *ACHA*, 30(1):20–36, 2011. 2

[32] Jos F Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999. 2

[33] Chengzhou Tang, Oliver Wang, and Ping Tan. GSLAM: Initialization-robust monocular visual SLAM via global structure-from-motion. In *IEEE 3DV*, 2017. 1

[34] Yulun Tian, Kasra Khosoussi, David M Rosen, and Jonathan P How. Distributed certifiably correct pose-graph optimization. *arXiv preprint arXiv:1911.03721*, 2019. 2, 4

[35] Roberto Tron, Bijan Afsari, and René Vidal. Intrinsic consensus on SO(3) with almost-global convergence. In *IEEE CDC*, 2012. 2

[36] Roberto Tron, Xiaowei Zhou, and Kostas Daniilidis. A survey on rotation optimization in structure from motion. In *CVPRW*, 2016. 1, 2

[37] Lanhui Wang and Amit Singer. Exact and stable recovery of rotations for robust synchronization. *Information and Inference: A Journal of the IMA*, 2(2):145–193, 2013. 2

[38] Kyle Wilson, David Bindel, and Noah Snavely. When is rotations averaging hard? In *ECCV*, 2016. 2

[39] Heng Yang, Pasquale Antonante, Vasileios Tzoumas, and Luca Carlone. Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection. *IEEE RAL*, 5(2):1127–1134, 2020. 3

[40] Heng Yang and Luca Carlone. One ring to rule them all: Certifiably robust geometric perception with outliers. In *NeurIPS*, 2020. 3

[41] Christopher Zach, Manfred Klopschitz, and Marc Pollefeys. Disambiguating visual relations using loop constraints. In *CVPR*, 2010. 2

[42] Qianggong Zhang, Tat-Jun Chin, and Huu Minh Le. A fast resection-intersection method for the known rotation problem. In *CVPR*, 2018. 8

[43] Siyu Zhu, Runze Zhang, Lei Zhou, Tianwei Shen, Tian Fang, Ping Tan, and Long Quan. Very large-scale global SfM by distributed motion averaging. In *CVPR*, 2018. 1