IEEE Robotics and Automation Letters (RAL) paper presented at the
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Asynchronous and Parallel Distributed Pose Graph Optimization

Yulun Tian[1], Alec Koppel[2], Amrit Singh Bedi[2], and Jonathan P. How[1]

Abstract— **We present Asynchronous Stochastic Parallel Pose Graph Optimization (ASAPP), the first *asynchronous* algorithm for distributed pose graph optimization (PGO) in multi-robot simultaneous localization and mapping. By enabling robots to optimize their local trajectory estimates without synchronization, ASAPP offers resiliency against communication delays and alleviates the need to wait for stragglers in the network. Furthermore, ASAPP can be applied on the rank-restricted relaxations of PGO, a crucial class of non-convex Riemannian optimization problems that underlies recent breakthroughs on globally optimal PGO. Under bounded delay, we establish the global first-order convergence of ASAPP using a sufficiently small stepsize. The derived stepsize depends on the worst-case delay and inherent problem sparsity, and furthermore matches known result for synchronous algorithms when there is no delay. Numerical evaluations on simulated and real-world datasets demonstrate favorable performance compared to state-of-the-art synchronous approach, and show ASAPP's resilience against a wide range of delays in practice.**

I. Introduction

Multi-robot simultaneous localization and mapping (SLAM) is a fundamental capability for many real-world robotic applications. *Pose graph optimization* (PGO) is the backbone of state of the art approaches to multi-robot SLAM, which fuses individual trajectories together and endows participating robots with a common spatial understanding of the environment. Many approaches to multi-robot PGO require the centralized processing of observations at a base station, which is communication intensive and vulnerable to single point of failure. In contrast, decentralized approaches are favorable as they effectively mitigate communication, privacy, and vulnerability concerns associated with centralization.

Recent works on distributed PGO have achieved important progress; see e.g., [1], [2] and the references therein. However, to the best of our knowledge, existing distributed algorithms are inherently *synchronous*, which necessitates that robots, for instance, pass messages over the network or wait at predetermined points, in order to ensure up-to-date information sharing during distributed optimization. Doing so may incur considerable communication overhead and increase the complexity of implementation. On the other hand, simply dropping synchronization in the execution of synchronous algorithms may slow down convergence or even cause divergence, both in theory and practice.

In this work, we overcome the aforementioned challenge by proposing ASAPP (**A**synchronous **S**toch**A**stic **P**arallel **P**ose Graph Optimization), the first *asynchronous* and *provably convergent* algorithm for distributed PGO. We take inspiration from existing parallel and asynchronous algorithms [3]–[11], and adapt these ideas to solve the *non-convex* Riemannian optimization problem underlying PGO. In ASAPP, each robot executes its local optimization loop at a high rate, without waiting for updates from others over the network. This makes ASAPP easier to implement in practice and flexible against communication delay. In addition, recent breakthroughs in centralized PGO, starting with SE-Sync [12], show how one can obtain globally optimal solutions to PGO by solving a hierarchy of rank-restricted relaxations. In this work, we leverage the important insights provided by SE-Sync and subsequent works [2], [13], and show that ASAPP can solve both PGO and its rank-restricted relaxations.

This work focuses on solving the distributed SLAM back-end (i.e., pose graph optimization). Our solver can be combined with a separate front-end module into a full SLAM solution, similar to what is done in state-of-the-art multi-robot SLAM systems, e.g., [14], [15].

Contributions Since asynchronous algorithms allow communication delays to be substantial and unpredictable, it is usually unclear under what conditions they converge in practice. In this work, we provide a rigorous answer to this question and establish the first known convergence result for asynchronous algorithms on the *non-convex* PGO problem. In particular, we show that as long as the worst-case delay is not arbitrarily large, ASAPP with a sufficiently small stepsize always converges to first-order critical points when solving PGO and its rank-restricted relaxations, with *global* sublinear convergence rate. The derived stepsize depends on the maximum delay and inherent problem sparsity, and furthermore reduces to the well known constant of $1/L$ (where L is the Lipschitz constant) for synchronous algorithms when there is no delay. Numerical evaluations on simulated and real-world datasets demonstrate that ASAPP compares favorably against state-of-the-art synchronous methods, and furthermore is resilient against a wide range of communication delays. Both results show the practical value of the proposed algorithm in a realistic distributed setting.

Preliminaries on Riemannian Optimization

This work relies heavily on the first-order geometry of Riemannian manifolds. The reader is referred to [16] for a rigorous treatment of this subject. In SLAM, examples of

This work was supported in part by ARL DCIST under Cooperative Agreement Number W911NF-17-2-0181 and by NASA Convergent Aeronautics Solutions project Design Environment for Novel Vertical Lift Vehicles (DELIVER).

[1]Y. Tian and J. P. How are with the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA, {yulun, jhow}@mit.edu

[2]A. Koppel and A. S. Bedi are with the U.S. Army Research Laboratory, Adelphi, MD 20783 alec.e.koppel.civ@mail.mil, amrit0714@gmail.com

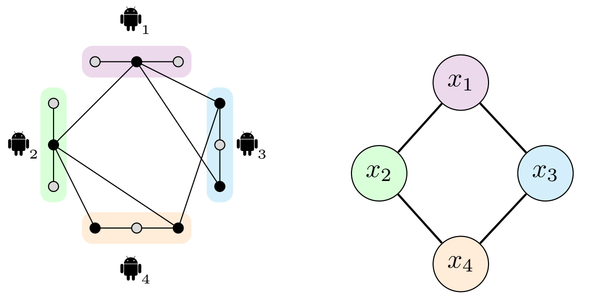
matrix manifolds that frequently appear include the orthogonal group $\mathrm{O}(d)$, special orthogonal group $\mathrm{SO}(d)$, and the special Euclidean group $\mathrm{SE}(d)$. In this work, we use $\mathcal{M} \subseteq \mathcal{E}$ to denote a general matrix submanifold, where $\mathcal{E}$ is the so-called ambient space (in this work, $\mathcal{E}$ is always the Euclidean space). Each point $x \in \mathcal{M}$ on the manifold has an associated tangent space $T_x\mathcal{M}$. Informally, $T_x\mathcal{M}$ contains all possible directions of change at x while staying on $\mathcal{M}$. As $T_x\mathcal{M}$ is a vector space, we also endow it with the standard Frobenius inner product, i.e., for two tangent vectors $\eta_1, \eta_2 \in T_x\mathcal{M}$, $\langle \eta_1, \eta_2 \rangle \triangleq \operatorname{tr}(\eta_1^\top \eta_2)$. The inner product induces a norm $\|\eta\| \triangleq \sqrt{\langle \eta, \eta \rangle}$. Finally, a tangent vector can be mapped back to the manifold through a retraction $\mathrm{Retr}_x : T_x\mathcal{M} \to \mathcal{M}$, which is a smooth mapping that preserves the first-order structure of the manifold [16].

Riemannian optimization considers minimizing a function $f : \mathcal{M} \to \mathbb{R}$ on the manifold. First-order Riemannian optimization algorithms, including the one proposed in this work, often use the Riemannian gradient $\operatorname{grad} f(x) \in T_x\mathcal{M}$, which corresponds to the direction of steepest ascent in the tangent space. For matrix submanifolds, the Riemannian gradient is obtained by an orthogonal projection of the usual Euclidean gradient $\nabla f(x)$ onto the tangent space, i.e., $\operatorname{grad} f(x) = \operatorname{Proj}_{T_x\mathcal{M}} \nabla f(x)$ [16]. We call $x^\star \in \mathcal{M}$ a first-order critical point if $\operatorname{grad} f(x^\star) = 0$.

II. Related Work

A. Distributed and Parallel PGO

In pursuit of decentralized *asynchronous* algorithms, we note that synchronized decentralized PGO has been well-studied. Tron et al. [17], [18] propose distributed Riemannian gradient descent on a set of reshape cost functions based on the geodesic distance, under which the method provably converges. A similar gradient-based method with line-search has been proposed [19]. Cunningham et al. [20]–[22] propose DDF-SAM, a decentralized system for feature-based SLAM where robots exchange Gaussian marginals over commonly observed variables. The use of local cache in [20]–[22] further allows asynchronous communication, although convergence is not discussed in this setting. Choudhary et al. [23] propose the alternating direction method of multipliers (ADMM) as a decentralized method to solve PGO. However, convergence of ADMM is not established due to the non-convex nature of the optimization problem. More recently, Choudhary et al. [1] propose a two-stage approach where each stage uses distributed Gauss-Seidel [3] to solve a relaxed or linearized PGO problem. The two-stage approach [1] is further combined with outlier rejection schemes in [15]. In our recent work [2], we avoid explicit linearization by directly optimizing PGO and its rank-restricted relaxations [13]. The proposed solver performs distributed block-coordinate descent over the product of Riemannian manifolds, and provably converges to first-order critical points with global sublinear rate. In a separate line of research, Fan and Murphey [24] propose an empirically accelerated PGO solver suitable for distributed optimization based on generalized proximal methods.



(a) Pose graph G (b) Robot-level graph $G_{\mathcal{R}}$

Fig. 1: (a) Example pose graph G with 4 robots, each with 3 poses. Each edge denotes a relative pose measurement. Private poses are colored in gray. (b) Corresponding robot-level graph $G_{\mathcal{R}}$. Two robots are connected if they share any relative measurements (inter-robot loop closures). Note that at any time during distributed optimization, robots do not need to share their private poses with any other robots.

B. Asynchronous Parallel Optimization

The aforementioned works are promising, but critically rely on *synchronization* for convergence, which limits their practical value for networked autonomous systems. However, within the broader optimization literature, there is a plethora of works on parallel and asynchronous optimization, partially motivated by popular applications in large-scale machine learning and deep learning. Study of asynchronous gradient-based algorithms began with the seminal work of Bertsekas and Tsitsilis [3]. Recent work further developed asynchronous randomized block coordinate and stochastic gradient algorithms [4]–[7]. Some work study asynchronous optimization in decentralized multi-agent systems [8], [9]. We are especially interested in asynchronous parallel schemes for *non-convex* optimization, which have been studied in [10], [11]. In this work, we generalize these approaches to the setting where the feasible set is the product of non-convex matrix manifolds, motivated by PGO.

III. Problem Formulation

In this section, we formally define pose graph optimization (PGO) in the context of multi-robot SLAM. Given relative pose measurements (possibly between different robots), we aim to *jointly* estimate the trajectories of all robots in a global reference frame. Let $\mathcal{R} = \{1, 2, \dots, n\}$ be the set of indices associated with n robots. Denote the pose of robot $i \in \mathcal{R}$ at time step τ as $T_{i_\tau} = (R_{i_\tau}, t_{i_\tau}) \in \mathrm{SE}(d)$, where $d \in \{2, 3\}$ is the dimension of the estimation problem. Here $R_{i_\tau} \in \mathrm{SO}(d)$ is a rotation matrix, and $t_{i_\tau} \in \mathbb{R}^d$ is a translation vector. A relative pose measurement from T_{i_τ} to T_{j_s} is denoted as $\widetilde{T}^{i_\tau}_{j_s} = (\widetilde{R}^{i_\tau}_{j_s}, \widetilde{t}^{i_\tau}_{j_s}) \in \mathrm{SE}(d)$. We assume the following standard noise model, which is used by SE-Sync [12] and subsequent works [2], [13],

$$\widetilde{R}^{i_\tau}_{j_s} = \underline{R}^{i_\tau}_{j_s} R^{\epsilon}_{i_\tau, j_s}, \qquad R^{\epsilon}_{i_\tau, j_s} \sim \mathrm{Langevin}(I_d, w_R), \tag{1}$$

$$\widetilde{t}^{i_\tau}_{j_s} = \underline{t}^{i_\tau}_{j_s} + t^{\epsilon}_{i_\tau, j_s}, \qquad t^{\epsilon}_{i_\tau, j_s} \sim \mathcal{N}(0, w_t^{-1} I_d). \tag{2}$$

Above, $\underline{T}^{i_\tau}_{j_s} = (\underline{R}^{i_\tau}_{j_s}, \underline{t}^{i_\tau}_{j_s}) \in \mathrm{SE}(d)$ denotes the true (i.e., noiseless) relative transformation. The isotropic Langevin noise on rotations [12] plays an analogous role as the Gaussian noise on translations. Given noisy measurements of the form (1)-(2), we seek to find the maximum likelihood estimates (MLE) for the trajectories of all robots in $\mathcal{R}$. Doing so amounts to the following non-convex program [12].

Problem 1 (Maximum Likelihood Estimation)**.**

$$\min \sum_{(i_\tau, j_s) \in E} w_R \left\| R_{j_s} - R_{i_\tau} \widetilde{R}^{i_\tau}_{j_s} \right\|_{\mathrm{F}}^2 + w_t \left\| t_{j_s} - t_{i_\tau} - R_{i_\tau} \widetilde{t}^{i_\tau}_{j_s} \right\|_2^2,$$
$$\text{s.t.} \quad R_{i_\tau} \in \mathrm{SO}(d),\ t_{i_\tau} \in \mathbb{R}^d,\ \forall i \in \mathcal{R},\ \forall \tau. \qquad (\mathrm{P}_1)$$

Problem (P_1) can be compactly represented with a *pose graph* $G \triangleq (V, E)$, where each vertex in V corresponds to a single pose owned by a robot. Observe that the sum in the objective is taken over all edges in E, where an edge from i_τ to j_s is formed if there is a relative measurement from T_{i_τ} to T_{j_s}. Fig. 1a shows an example pose graph.

In this paper, we further consider the *rank-restricted relaxation* of (P_1). Denote the Stiefel manifold as $\mathrm{St}(d, r) \triangleq \{Y \in \mathbb{R}^{r \times d} : Y^\top Y = I_d\}$, where $r \geq d$. The rank-r relaxation of (P_1) is defined as the following non-convex Riemannian optimization problem.

Problem 2 (Rank-r Relaxation)**.**

$$\min \sum_{(i_\tau, j_s) \in E} w_R \left\| Y_{j_s} - Y_{i_\tau} \widetilde{R}^{i_\tau}_{j_s} \right\|_{\mathrm{F}}^2 + w_t \left\| p_{j_s} - p_{i_\tau} - Y_{i_\tau} \widetilde{t}^{i_\tau}_{j_s} \right\|_2^2,$$
$$\text{s.t.} \quad Y_{i_\tau} \in \mathrm{St}(d, r),\ p_{i_\tau} \in \mathbb{R}^r,\ \forall i \in \mathcal{R},\ \forall \tau. \qquad (\mathrm{P}_2)$$

Observe that for $r = d$, the Stiefel manifold is identical to the orthogonal group $\mathrm{St}(d, d) = \mathrm{O}(d)$. In this case, (P_2) is referred to as the *orthogonal relaxation* of (P_1), obtained by dropping the determinant constraint on $\mathrm{SO}(d)$. As r increases beyond d, we obtain a hierarchy of rank-restricted problems, each having the form of (P_2) but with a slightly "lifted" search space as determined by r. The idea of rank relaxations is first proposed in SE-Sync [12], where Rosen et al. use a very similar hierarchy of rank relaxations as a proxy to solve the semidefinite relaxation of PGO, thereby recovering *global minimizers* to the original MLE (P_1).[1] This elegant result motivates us to consider (P_2) in addition to the original MLE formulation (P_1). While SE-Sync assumes a centralized (single-robot) scenario, we focus on distributed PGO and develop an *asynchronous* algorithm to solve both (P_1) and (P_2) in the presence of communication delay.

For the purpose of designing decentralized algorithms (Section IV), it is more convenient to rewrite (P_1) and (P_2) into a more abstract form at the level of robots, which may be done as follows.

Problem 3 (Robot-level Optimization Problem)**.**

$$\min \sum_{(i,j) \in E_{\mathcal{R}}} f_{ij}(x_i, x_j) + \sum_{i \in \mathcal{R}} h_i(x_i), \qquad (\mathrm{P})$$
$$\text{s.t.} \quad x_i \in \mathcal{M}_i,\ \forall i \in \mathcal{R}.$$

[1]The only difference between the rank relaxations in [12] and (P_2) is that translations variables are analytically eliminated in [12]. In (P_2), we keep translations in our optimization to retain sparsity. This approach is first used in Cartan-Sync [13].

In (P), each variable x_i concatenates all variables owned by robot $i \in \mathcal{R}$. For instance, for (P_2), x_i contains all the "lifted" rotation and translation variables of robot i. Let n_i be the number of poses of robot i. Then,

$$x_i = \begin{bmatrix} Y_{i_1} & p_{i_1} & \dots & Y_{i_{n_i}} & p_{i_{n_i}} \end{bmatrix}, \qquad (5)$$
$$\mathcal{M}_i = (\mathrm{St}(d, r) \times \mathbb{R}^r)^{n_i}. \qquad (6)$$

The cost function in (P) consists of a set of *shared costs* $f_{ij} : \mathcal{M}_i \times \mathcal{M}_j \to \mathbb{R}$ between pairs of robots, and a set of *private costs* $h_i : \mathcal{M}_i \to \mathbb{R}$ for individual robots. Intuitively, f_{ij} is formed by relative measurements between any of robot i's poses and j's poses. In contrast, h_i is formed by relative measurements within robot i's own trajectory.

Similar to the way a pose graph is defined, we can encode the structure of (P) using a *robot-level graph* $G_{\mathcal{R}} \triangleq (\mathcal{R}, E_{\mathcal{R}})$; see Fig. 1b. $G_{\mathcal{R}}$ can be viewed as a "reduced" graph of the pose graph, in which each vertex corresponds to the entire trajectory of a single robot $i \in \mathcal{R}$. Two robots i, j are connected in $G_{\mathcal{R}}$ if they share any relative measurements $(i_\tau, j_s) \in E$. In this case, we call j a *neighboring robot* of i, and j_s a *neighboring pose* of robot i. If a pose variable is not a neighboring pose to any other robots, we call this pose a *private pose* [2]. We note that for robot i to evaluate the shared cost f_{ij}, it only needs to know its neighboring poses in robot j's trajectory (see Fig. 1). This property is crucial in preserving the *privacy* of participating robots [1], [2], i.e., at any time, a robot does not need to share its private poses with any of its teammates.

IV. Proposed Algorithm

We present our main algorithm, Asynchronous Stochastic Parallel Pose Graph Optimization (ASAPP), for solving distributed PGO problems of the form (P). Our algorithm is inspired by asynchronous stochastic coordinate descent (e.g., see [7]). In the context of distributed PGO, each coordinate corresponds to the stacked relative pose observations x_i of a single robot as defined in (P).

In a practical multi-robot SLAM scenario, each robot can optimize its own pose estimates at any time, and can additionally share its (non-private) poses with others when communication is available. Correspondingly, each robot running ASAPP has two concurrent onboard processes, which we refer to as the *optimization thread* and *communication thread*. We emphasize that the robots perform both optimization and communication completely in parallel and without synchronization with each other. We begin by describing the communication thread and then proceed to the optimization thread. Without loss of generality, we describe the algorithm from the perspective of robot $i \in \mathcal{R}$.

A. Communication Thread

As part of the communication module, each robot $i \in \mathcal{R}$ implements a local data structure, called a *cache*, that contains the robot's own variable x_i, together with the most recent copies of neighboring poses received from the robot's neighbors. A very similar design that allows asynchronous communication is proposed by Cunningham et al. [20]–[22], although the authors have not discussed convergence in the asynchronous setting.

Since only i can modify x_i, the value of x_i in robot i's cache is guaranteed to be up-to-date at anytime. In contrast, the copies of neighboring poses from other robots can be *out-of-date* due to communication delay. For example, by the time robot i receives and uses a copy of robot j's poses, j might have already updated its poses due to its local optimization process. In Section V, we show that ASAPP is resilient against such network delay. Nevertheless, for ASAPP to converge, we still assume that the total delay induced by the communication process remains *bounded* (Section V). The communication thread performs the following two operations over the cache.

• **Receive**: After receiving a neighboring pose, e.g., (R_{j_s}, t_{j_s}) from a neighboring robot j over the network, the communication thread updates the corresponding entry in the cache to store the new value.

• **Send**: Periodically (when communication is available), robot i also transmits its latest public pose variables (i.e., poses that have inter-robot measurements with other robots) to its neighbors. Robot i does not need to send its private poses, as these poses are not needed by other robots to update their estimates.

B. Optimization Thread

Concurrent to the communication thread, the optimization thread is invoked by a local clock that ticks according to a Poisson process of rate $\lambda > 0$.

Definition 1 (Poisson process [25])**.** *Consider a sequence $\{X_1, X_2, ...\}$ of positive, independent random variables that represent the time elapsed between consecutive events (in this case, clock ticks). Let $N(t)$ be the number of events up to time $t \geq 0$. The counting process $\{N(t), t \geq 0\}$ is a Poisson process with rate $\lambda > 0$ if the interarrival times $\{X_1, X_2, ...\}$ have a common exponential distribution function,*

$$P(X_k \leq a) = 1 - e^{-\lambda a},\ a \geq 0. \tag{7}$$

The use of Poisson clocks originates from the design of randomized gossip algorithms by Boyd et al. [26] and is a commonly used tool for analyzing the global behavior of distributed randomized algorithms. The rate parameter λ is equal among robots. In practice, we can adjust λ based on the extent of network delay and the robots' computational capacity. Using this local clock, the optimization thread performs the following operations in a loop.

• **Read**: For each neighboring robot $j \in \mathrm{N}(i)$, read the value of x_j stored in the local cache. Denote the read values as $\hat{x}_j$. Recall that $\hat{x}_j$ can be *outdated*, for example if robot i has not received the latest messages from robot j. In addition, read the value of x_i, denoted as $\hat{x}_i$. Recall from Section IV-A that $\hat{x}_i$ is guaranteed to be up-to-date.

In practice, $\hat{x}_j$ only contains the set of neighboring poses from robot j since f_{ij} is independent from the rest of j's poses (Fig. 1). However, for ease of notation and analysis, we treat $\hat{x}_j$ as if it contains the entire set of j's poses.

• **Compute**: Form the local cost function for robot i, denoted as $g_i(x_i) : \mathcal{M}_i \to \mathbb{R}$, by aggregating relevant costs in (P) that involve x_i,

$$g_i(x_i) = h_i(x_i) + \sum_{j \in \mathrm{N}(i)} f_{ij}(x_i, \hat{x}_j). \tag{8}$$

Compute the Riemannian gradient at robot i's current estimate $\hat{x}_i$,

$$\eta_i = \operatorname{grad} g_i(\hat{x}_i) \in T_{\hat{x}_i}\mathcal{M}_i. \tag{9}$$

• **Update**: At the next local clock tick, update x_i in the direction of the negative gradient,

$$x_i \leftarrow \mathrm{Retr}_{\hat{x}_i}(-\gamma \eta_i), \tag{10}$$

where $\gamma > 0$ is a constant stepsize. Equation (10) gives the simplest update rule that robots can follow. In Section V, we further prove convergence for this update rule.

To accelerate convergence in practice, SE-Sync [12] and Cartan-Sync [13] use a heuristic known as *preconditioning*, which is also applicable to ASAPP. With preconditioning, the following alternative update direction is used,

$$x_i \leftarrow \mathrm{Retr}_{\hat{x}_i}(-\gamma \operatorname{Precon} g_i(\hat{x}_i)[\eta_i]). \tag{11}$$

In (11), $\operatorname{Precon} g_i(\hat{x}_i) : T_{\hat{x}_i}\mathcal{M}_i \to T_{\hat{x}_i}\mathcal{M}_i$ is a linear, symmetric, and positive definite mapping on the tangent space that approximates the inverse of Riemannian Hessian. Intuitively, preconditioning helps first-order methods benefit from using the (approximate) second-order geometry of the cost function, which often results in significant speedup especially on poorly conditioned problems.

C. Implementation Details

To make the local clock model valid, we require that the total execution time of the **Read**-**Compute**-**Update** sequence be smaller than the interarrival time of the Poisson clock, so that the current sequence can finish before the next one starts. This requirement is fairly lax in practice, as all three steps only involve minimal computation and access to local memory. In the worst case, since the interarrival time is determined by $1/\lambda$ on average [25], one can also decrease the clock rate λ to create more time for each update.

In addition, we note that although the optimization and communication threads run concurrently, minimal thread-level synchronization is required to ensure the so-called *atomic read and write* of individual poses. Specifically, a thread cannot read a pose in the cache if the other thread is actively modifying its value (otherwise the read value would not be valid). Such synchronization can be easily enforced using software locks.

V. Convergence Analysis

A. Global View of the Algorithm

In Section IV, we described ASAPP from the local perspective of each robot. For the purpose of establishing convergence, however, we need to analyze the systematic behavior of this algorithm from a global perspective [6], [7], [9], [26]. To do so, let $k = 0, 1, \ldots$ be a virtual counter that counts the total number of **Update** operations applied by all robots. In addition, let the random variable $i_k \in \mathcal{R}$ represent the robot that updates at global iteration k. We emphasize that k and i_k are purely used for theoretical analysis, and are unknown to any of the robots in practice.

Recall from Section IV-B that all **Update** steps are generated by $n = |\mathcal{R}|$ independent Poisson processes, each with rate λ. In the global perspective, merging these local processes is equivalent to creating a single, global Poisson clock with rate λn. Furthermore, at any time, all robots have equal probabilities of generating the next **Update** step, i.e., for all $k \in \mathbb{N}$, i_k is i.i.d. uniformly distributed over the set $\mathcal{R}$. See [25] for proofs of these results.

Using this result, we can write the iterations of ASAPP from the global view; see Algorithm 1. We use $x^k \triangleq \begin{bmatrix} x_1^k & x_2^k & \dots & x_n^k \end{bmatrix}$ to represent the value of all robots' poses after k global iterations (i.e., after k total **Update** steps). Note that x lives on the product manifold $\mathcal{M} \triangleq \mathcal{M}_1 \times \mathcal{M}_2 \times \dots \mathcal{M}_n$. At global iteration k, a robot i_k is selected from $\mathcal{R}$ uniformly at random (line 2). Robot i_k then follows the steps in Section IV-B to update its own variable (line 3-6). We have used the fact that $\hat{x}_{i_k}$ is always up-to-date (line 3), while $\hat{x}_{j_k}$ is outdated for $B(j_k)$ total **Update** steps (line 4). Except robot i_k, all other robots do not update (line 7).

Algorithm 1 GLOBAL VIEW OF ASAPP (For Analysis Only)

```
Input:
    Initial solution $x^0 \in \mathcal{M}$ and stepsize $\gamma > 0$.
1: for global iteration $k = 0, 1, \dots$ do
2:     Select robot $i_k \in \mathcal{R}$ uniformly at random.
3:     Read $\hat{x}_{i_k} = x_{i_k}^k$.
4:     Read $\hat{x}_{j_k} = x_{j_k}^{k-B(j_k)}$, $\forall j_k \in \mathrm{N}(i_k)$.
5:     Compute local gradient $\eta_{i_k}^k = \operatorname{grad} g_{i_k}(\hat{x}_{i_k})$.
6:     Update $x_{i_k}^{k+1} = \operatorname{Retr}_{\hat{x}_{i_k}}(-\gamma \eta_{i_k}^k)$.
7:     Carry over all $x_j^{k+1} = x_j^k$, $\forall j \neq i_k$.
8: end for
```

B. Sufficient Conditions for Convergence

We establish sufficient conditions for ASAPP to converge to first-order critical points. Due to space limitation, all proofs are deferred to the appendix [27]. We adopt the commonly used *partially asynchronous* model [3], which assumes that delay caused by asynchrony is not arbitrarily large. In practice, the magnitude of delay is affected by various factors such as the rate of communication (Section IV-A), the rate of local optimization (Section IV-B), and intrinsic network latency. For the purpose of analysis, we assume that all these factors can be summarized into a single constant B, which bounds the maximum delay in terms of number of *global iterations* (i.e., **Update** steps applied by all robots) in Algorithm 1.

Assumption 1 (Bounded Delay)**.** *In Algorithm 1, there exists a constant $B \geq 0$ such that $B(j_k) \leq B$ for all j_k.*

Assumption 1 imposes a worst-case upper bound on the delay, and allows the actual delay to fluctuate within this upper bound. In addition, for both the MLE problem (P$_1$) and its rank-restricted relaxations (P$_2$), the gradients enjoy a Lipschitz-type condition, which is proved in our previous work [2] and will be used extensively in the rest of the analysis.

Lemma 1 (Lipschitz-type gradient for pullbacks [2], [28])**.** *Denote the cost function of* (P$_1$) *and* (P$_2$) *as $f : \mathcal{M} \to \mathbb{R}$. Define the* pullback *cost as $\hat{f}_x \triangleq f \circ \operatorname{Retr}_x : T_x\mathcal{M} \to \mathbb{R}$. There exists a constant $L \geq 0$ such that for any $x \in \mathcal{M}$ and $\eta \in T_x\mathcal{M}$,*

$$\left|\hat{f}_x(\eta) - [f(x) + \langle \eta, \operatorname{grad}_x f \rangle]\right| \leq \frac{L}{2} \|\eta\|^2 . \quad (12)$$

The condition (12) is first proposed by [28] as an adaptation of Lipschitz continuous gradient to Riemannian optimization. Using the bounded delay assumption and the Lipschitz-type condition in (12), we can proceed to analyze the change in cost function after a single iteration of Algorithm 1 (in the global view). We formally state the result in the following lemma.

Lemma 2 (Descent Property of Algorithm 1)**.** *Under Assumption 1, each iteration of Algorithm 1 satisfies,*

$$\begin{aligned} f(x^{k+1}) - f(x^k) \leq -\frac{\gamma}{2} \left\|\operatorname{grad}_{i_k} f(x^k)\right\|^2 - \frac{\gamma - L\gamma^2}{2} \left\|\eta_{i_k}^k\right\|^2 \\ + \frac{\Delta B L^2 \alpha^2 \gamma^3}{2} \sum_{j_k \in N(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|^2 , \end{aligned} \quad (13)$$

where η_i^k denotes the update taken by robot i at iteration k, $\alpha > 0$ is a constant related to the retraction, and $\Delta > 0$ is the maximum degree of the robot-level graph $G_{\mathcal{R}}$.

In (13), the last term on the right hand side sums over the squared norms of a set of $\{\eta_{j_k}^{k'}\}$, where each $\eta_{j_k}^{k'}$ corresponds to the update taken by a neighbor j_k at an earlier iteration k'. This term is a direct consequence of delay in the system, and is also the main obstacle for proving convergence in the asynchronous setting. Indeed, without this term, it is straightforward to verify that any stepsize that satisfies $0 < \gamma < 1/L$ guarantees $f(x^{k+1}) \leq f(x^k)$, and thus leads to convergent behavior. With the last term in (13), however, the overall cost could increase after each iteration.

While the delay-dependent error term gives rise to additional challenges, our next theorem states that with sufficiently small stepsize, this error term is inconsequential and ASAPP provably converges to first-order critical points.

Theorem 1 (Global convergence of ASAPP)**.** *Let $f^\star$ be any global lower bound on the optimum of* (P)*. Define $\rho \triangleq \Delta/n$. Let $\bar{\gamma} > 0$ be an upper bound on the stepsize that satisfies,*

$$2\rho\alpha^2 B^2 L^2 \bar{\gamma}^2 + L\bar{\gamma} - 1 \leq 0. \quad (14)$$

In particular, the following choice of $\bar{\gamma}$ satisfies (14)*:*

$$\bar{\gamma} = \begin{cases} \frac{\sqrt{1+8\rho\alpha^2 B^2}-1}{4\rho\alpha^2 B^2 L}, & B > 0, \\ 1/L, & B = 0. \end{cases} \quad (15)$$

Under Assumption 1, if $0 < \gamma \leq \bar{\gamma}$, ASAPP *converges to a first-order critical point with global sublinear rate. Specifically, after* K *total update steps,*

$$\min_{k \in \{0,\ldots,K-1\}} \mathbb{E}_{i_{0:K-1}} \left[\left\| \operatorname{grad} f(x^k) \right\|^2 \right] \leq \frac{2n(f(x^0) - f^\star)}{\gamma K}. \tag{16}$$

Remark 1. To the best of our knowledge, Theorem 1 establishes the first convergence result for asynchronous algorithms when solving a *non-convex* optimization problem over the product of matrix manifolds. While the existence of a convergent stepsize $\bar{\gamma}$ is of theoretical importance, we further note that its expression (15) offers the correct qualitative insights with respect to various problem-specific parameters, which we discuss next.

Relation with maximum delay (B): The step size $\bar{\gamma}$ increases as maximum delay B decreases. Intuitively, as communication becomes increasingly available, each robot may take larger steps without causing divergence. The inverse relationship between $\bar{\gamma}$ and B is well known in the asynchronous optimization literature, and is first established by Bertsekas and Tsitsilis [3] in the Euclidean setting.

Relation with problem sparsity (ρ): $\bar{\gamma}$ increases as ρ decreases. Recall that $\rho \triangleq \Delta/n$ is defined as the ratio between the maximum number of neighbors a robot has and the total number of robots. Thus, ρ is a measure of *sparsity* of the robot-level graph $G_{\mathcal{R}}$. Intuitively, as $G_{\mathcal{R}}$ becomes more sparse, robots can use larger stepsize as their problems become increasingly decoupled. Such positive correlation between $\bar{\gamma}$ and problem sparsity has been a crucial feature in state-of-the-art asynchronous algorithms; see e.g., [5].

Relation with problem smoothness (L): From (15), it can been seen that $\bar{\gamma}$ increases asymptotically with $\mathcal{O}(1/L)$. Moreover, when there is no delay ($B = 0$), our stepsize matches the well-known constant of $1/L$ with which synchronous gradient descent converges to first-order critical points; see e.g., [28].

VI. Experimental Results

We implement ASAPP in C++ and evaluate its performance on both simulated and real-world PGO datasets. We use ROPTLIB [29] for manifold related computations, and the Robot Operating System (ROS) [30] for inter-robot communication. The Poisson clock is implemented by halting the optimization thread after each iteration for a random amount of time exponentially distributed with rate λ (default to 1000 Hz). Since the time taken by each iteration is negligible, we expect the practical difference between this implementation and the theoretical model in Section IV-B to be insignificant. All robots are simulated as separate ROS nodes running on a desktop computer with an Intel i7 quad-core CPU and 16 GB memory.

For each PGO problem, we use ASAPP to solve its rank-restricted relaxation (P_2) with $r = 5$. As is commonly done in prior work [4]–[7], [9]–[11], in our experiments we select the stepsize empirically. During optimization, we record the evolution of the Riemannian gradient norm $\left\| \operatorname{grad} f(x^k) \right\|$, which measures convergence to a first-order critical point. In addition, we also record the optimality gap $f(x^k) - f(x^\star)$, where $x^\star$ is a global minimizer to the PGO problem (P_1) computed by Cartan-Sync [13]. In Section VI-B, we also round the solution to $\mathrm{SE}(d)$ using the method in [2] and then compute the translation root mean squared error (RMSE) and rotation RMSE (in chordal distance) with repspect to the global minimizer.

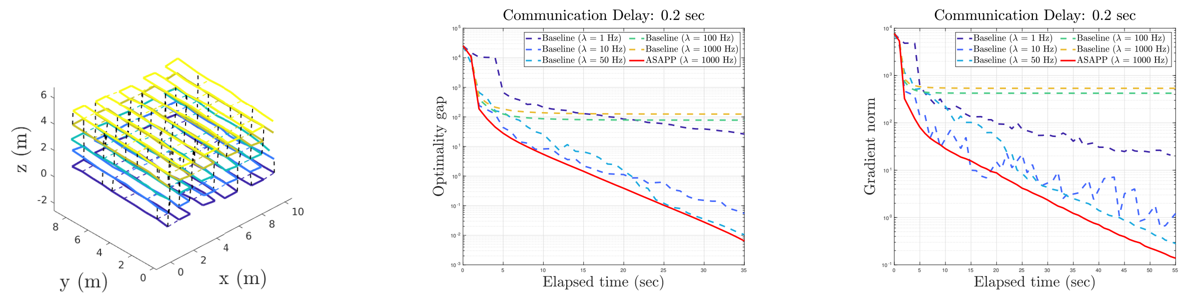
A. Evaluation in Simulation

We evaluate ASAPP in a simulated multi-robot SLAM scenario in which 5 robots move next to each other in a 3D grid with lawn mower trajectories (Fig. 2a). Each robot has 100 poses. With probability 0.3, loop closures within and across trajectories are generated for poses within 1 m of each other. All measurements are corrupted by Langevin rotation noise with 2° standard deviation, and Gaussian translation noise with 0.05 m standard deviation. To minimize communication during initialization, we initialize the solution by propagating relative measurements along a spanning tree of the global pose graph. The stepsize used in simulation is $\gamma = 5 \times 10^{-4}$.

In the first experiment, we simulate communication delay by letting each robot communicate every 0.2 s. We compare the performance of ASAPP (without preconditioning) against a baseline algorithm in which each robot uses the second-order Riemannian trust-region (RTR) method to optimize its local variable, similar to the approach in [2]. Starting with SE-Sync [12], RTR has been used as the default solver in centralized or synchronous settings due to its global convergence guarantees and ability to exploit second-order geometry of the cost function. For a comprehensive evaluation, we record the performance of this baseline at different optimization rates (i.e. frequency at which robots update their local trajectories).

Fig. 2b shows the optimality gaps achieved by the evaluated algorithms as a function of wall clock time. The corresponding reduction in the Riemannian gradient norm is shown in Fig. 2c. ASAPP outperforms all variants of the baseline algorithm (dashed curves). We note that the behavior of the baseline algorithm is expected. At a low rate, e.g., $\lambda = 1$ Hz (dark blue dashed curve), the baseline algorithm is essentially synchronous as each robot has access to up-to-date poses from others. The empirical convergence speed is nevertheless slow, since each robot needs to wait for up-to-date information to arrive after each iteration. At a high rate, e.g., $\lambda = 1000$ Hz (dark yellow dashed curve), robots essentially behave asynchronously. However, since RTR does not regulate stepsize at each iteration, robots often significantly alter their solutions in the wrong direction (as a result of using outdated information), which leads to slow convergence or even non-convergence. In contrast, ASAPP is provably convergent, and furthermore is able to exploit asynchrony effectively to achieve speedup.

In addition, we also evaluate ASAPP under a wide range of communication delays. Due to space limitation, we only



(a) Simulation (b) Optimality gap (c) Riemannian gradient norm

Fig. 2: Performance evaluation on 5 robot simulation. The communication delay is fixed at 0.5 s. We compare ASAPP (with stepsize $\gamma = 5 \times 10^{-4}$) with a baseline algorithm in which each robot uses Riemannian trust-region method to optimize its local variables. For a comprehensive evaluation, we run the baseline with varying optimization rate to record its performance under both synchronous and asynchronous regimes. (a) Example trajectories estimated by ASAPP, where trajectories of 5 robots are shown in different colors. Inter-robot measurements (loop closures) are shown as black dashed lines. (b) Optimality gap with respect to the centralized global minimizer $f(x^k) - f(x^\star)$. (c) Riemannian gradient norm $\|\text{grad}\, f(x^k)\|$.

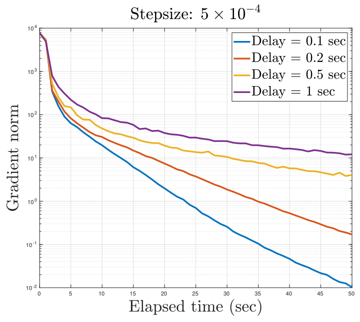


Fig. 3: Convergence speed of ASAPP (stepsize $\gamma = 5\times10^{-4}$) with varying communication delay. As delay decreases, convergence becomes faster because robots have access to more up-to-date information from each other.

show performance in terms of gradient norm in Fig. 3. We note that ASAPP converges in all cases, demonstrating its resilience against various delays in practice. Furthermore, as delay decreases, convergence becomes faster as robots have access to more up-to-date information from each other.

B. Evaluation on benchmark PGO datasets

We evaluate ASAPP on benchmark PGO datasets and compare its performance with Distributed Gauss-Seidel (DGS) [1], a state-of-the-art synchronous approach for distributed PGO used in recent multi-robot SLAM systems [14], [15]. Each dataset is divided into 5 segments simulating a collaborative SLAM mission with 5 robots.

Following Choudhary et al. [1], we initialize rotation estimates via distributed chordal initialization. To initialize the translations, we fix the rotation estimates and use distributed Gauss-Seidel to solve the reduced linear system over translations. To test on scenarios where accurate initializations are not available, we restrict the number of Gauss-Seidel iterations to 50 for both rotation and translation initialization.

Starting from the initial estimate, we run ASAPP with preconditioning for 60s, assuming for simplicity a fixed delay of 0.1s. Accordingly, we run DGS [1] on the problem (linearized at the initial estimate) for $60/0.1 = 600$ synchronous iterations. This setup favors DGS inherently, since each DGS iteration requires robots to communicate multiple times and update according to a specific order, which is likely to increase execution time in reality.

Table I compares the final cost values achieved by the two approaches, where for ASAPP we first round the solutions to $\mathrm{SE}(d)$. For ASAPP, we also report the used stepsize, final gradient norm, and estimation errors with respect to the global minimizer computed by Cartan-Sync [13]. As our results show, ASAPP often compares favorably against DGS, especially when the quality of initialization is poor. This is an important advantage, as good initialization schemes (such as distributed chordal initialization developed in [1]) are usually iterative and thus expensive in terms of communication. Furthermore, the ASAPP solution is close to the global minimizer, except on the `Manhattan` dataset where the rotation error is relatively high.

We conclude this section by observing that on certain large datasets, convergence of ASAPP is slow as the iterate approaches a critical point. This is a consequence of the sublinear convergence rate, and in our case convergence is further impacted by the presence of communication delay. To address this, future work could consider accelerated methods to achieve higher precision. To this end, the recent paper [24] that generalizes Nesterov's accleration to PGO provides a promising direction.

VII. Conclusion

We presented ASAPP, the first *asynchronous* and *provably delay-tolerant* algorithm to solve distributed pose graph optimization and its rank-restricted semidefinite relaxations. ASAPP enables each robot to run its local optimization process at a high rate, without waiting for updates from its peers over the network. Assuming a worst-case bound on the communication delay, we established the global first-order convergence of ASAPP, and showed the existence of a convergent stepsize whose value depends on the worst-case delay and inherent problem sparsity. When there is no delay, we further showed that this stepsize matches exactly with the corresponding constant in synchronous algorithms. Numerical evaluations on both simulation and real-world

Dataset	# Poses	# Edges	Cost value f				Additional statistics for ASAPP			
			Opt. [13]	Initial	DGS [1]	ASAPP [ours]	Grad. Norm	Rot. RMSE [chordal]	Trans. RMSE [m]	Stepsize
CSAIL (2D)	1045	1171	31.47	36.10	31.54	**31.51**	0.36	0.0017	0.001	1.0
Intel (2D)	1228	1483	393.7	1205.9	394.6	**393.7**	0.002	2×10^{-6}	2×10^{-6}	1.0
Manhattan (2D)	3500	5453	193.9	1187.2	242.7	**227.7**	5.42	0.07	0.02	0.03
Garage (3D)	1661	6275	1.267	4.477	**1.277**	1.309	0.04	0.014	0.01	0.05
Sphere (3D)	2500	4949	1687.0	3075.7	1743.1	**1711.7**	2.73	0.02	0.01	0.23
Torus (3D)	5000	9048	24227	25812	24305	**24240**	8.38	0.017	0.001	1.0
Cubicle (3D)	5750	16869	717.1	916.2	**720.8**	734.5	12.67	0.030	0.001	0.065

TABLE I: Evaluation on benchmark PGO datasets. Each dataset is divided into trajectories of 5 robots. We run ASAPP for 60 s under a fixed communication delay of 0.1 s. For reference, we also run DGS [1] for 600 synchronous iterations. We compare the final cost values of the two approaches, and highlight the better solution in bold. For ASAPP, we also report the used stepsize, achieved gradient norm, and rotation and translation root mean squared errors (RMSE) with respect to the global minimizer, computed by Cartan-Sync [13].

datasets confirm the advantages of ASAPP in reducing overall execution time, and demonstrate its resilience against a wide range of communication delay.

Our theoretical study in Section V assumes a worst-case bounded delay. Future work could consider less conservative strategies. For example, the recent paper [31] establishes convergence of asynchronous coordinate descent under *unbounded* delay, where it is only assumed that the tail distribution of delay decays sufficiently fast. Another open question is conditions under which stronger performance guarantees may hold for first-order methods, e.g., second-order optimality. Recent works have shown promising results towards this new direction [32], [33].

References

[1] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models," *International Journal of Robotics Research*, 2017.

[2] Y. Tian, K. Khosoussi, D. M. Rosen, and J. P. How, "Distributed certifiably correct pose-graph optimization." `https://arxiv.org/abs/1911.03721`, 2019.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989.

[4] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," *NIPS*, 2011.

[5] F. Niu, B. Recht, C. Re, and S. Wright, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," *NIPS*, 2011.

[6] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar, "An asynchronous parallel stochastic coordinate descent algorithm," *Journal of Machine Learning Research*, 2015.

[7] J. Liu and S. J. Wright, "Asynchronous stochastic coordinate descent: Parallelism and convergence properties," *SIAM Journal on Optimization*, 2015.

[8] A. S. Bedi, A. Koppel, and K. Rajawat, "Asynchronous saddle point algorithm for stochastic optimization in heterogeneous networks," *IEEE Transactions on Signal Processing*, 2019.

[9] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[10] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," *NIPS*, 2015.

[11] L. Cannelli, F. Facchinei, V. Kungurtsev, and G. Scutari, "Asynchronous parallel algorithms for nonconvex optimization," *Mathematical Programming*, 2019.

[12] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard, "Se-sync: A certifiably correct algorithm for synchronization over the special euclidean group," *International Journal of Robotics Research*, 2019.

[13] J. Briales and J. Gonzalez-Jimenez, "Cartan-sync: Fast and global se(d)-synchronization," *IEEE Robotics and Automation Letters*, 2017.

[14] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-efficient decentralized visual slam," in *ICRA*, 2018.

[15] P. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame, "Door-slam: Distributed, online, and outlier resilient slam for robotic teams," *IEEE Robotics and Automation Letters*, 2020.

[16] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[17] R. Tron and R. Vidal, "Distributed image-based 3-d localization of camera sensor networks," in *CDC*, 2009.

[18] R. Tron and R. Vidal, "Distributed 3-d localization of camera sensor networks from 2-d image measurements," *IEEE Transactions on Automatic Control*, 2014.

[19] J. Knuth and P. Barooah, "Collaborative 3d localization of robots from relative pose measurements using gradient descent on manifolds," in *IEEE International Conference on Robotics and Automation*, 2012.

[20] A. Cunningham, M. Paluri, and F. Dellaert, "Ddf-sam: Fully distributed slam using constrained factor graphs," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[21] A. Cunningham, K. M. Wurm, W. Burgard, and F. Dellaert, "Fully distributed scalable smoothing and mapping with robust multi-robot data association," in *2012 IEEE International Conference on Robotics and Automation*, pp. 1093–1100, 2012.

[22] A. Cunningham, V. Indelman, and F. Dellaert, "Ddf-sam 2.0: Consistent distributed smoothing and mapping," in *2013 IEEE International Conference on Robotics and Automation*, 2013.

[23] S. Choudhary, L. Carlone, H. I. Christensen, and F. Dellaert, "Exactly sparse memory efficient slam using the multi-block alternating direction method of multipliers," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[24] T. Fan and T. D. Murphey, "Generalized proximal methods for pose graph optimization," in *ISRR*, 2019.

[25] H. Tijms, *A First Course in Stochastic Models*. John Wiley and Sons, Ltd, 2004.

[26] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, 2006.

[27] Y. Tian, A. Koppel, A. S. Bedi, and J. P. How, "Asynchronous and parallel distributed pose graph optimization." `https://arxiv.org/abs/2003.03281`, 2020.

[28] N. Boumal, P.-A. Absil, and C. Cartis, "Global rates of convergence for nonconvex optimization on manifolds," *IMA Journal of Numerical Analysis*, 2018.

[29] W. Huang, P.-A. Absil, K. A. Gallivan, and P. Hand, "Roptlib: an object-oriented c++ library for optimization on riemannian manifolds," tech. rep., Florida State University, 2016.

[30] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[31] T. Sun, R. Hannah, and W. Yin, "Asynchronous coordinate descent under more realistic assumption," in *NIPS*, 2017.

[32] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, "How to escape saddle points efficiently," in *ICML*, 2017.

[33] C. Criscitiello and N. Boumal, "Efficiently escaping saddle points on manifolds," in *NeurIPS*, 2019.