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 (SLAM). 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, the same algorithm can be used to solve the so-called rank-restricted semidefinite relaxations of PGO, a crucial class of non-convex Riemannian optimization problems at the center of recent PGO solvers with global optimality guarantees. 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 both simulated and real-world SLAM datasets demonstrate the speedup achieved with ASAPP and show the algorithm's resilience against a wide range of communication 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 endow participating robots a common spatial understanding. Many approaches to multi-robot PGO requires 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. To the best of our knowledge, however, 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 cause divergence, both in theory and practice.

[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

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]–[7], 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. Furthermore, we show that the same algorithm can be applied straightforwardly to solve the so-called rank-restricted semidefinite relaxations of PGO, a crucial class of non-convex Riemannian optimization problems that lies at the heart of recent PGO solvers with global optimality guarantees [2], [8], [9].

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 establishes 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 always achieves *global* first-order convergence using a sufficiently small stepsize. 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. In our experiments, we verify the convergence property of ASAPP, and demonstrate its resilience against a wide range of communication delays.

Contributions We present ASAPP, the first *asynchronous* algorithm to solve distributed PGO and its rank-restricted semidefinite relaxations. Under suitable hypotheses of the worst-case delay due to asynchrony, we prove that ASAPP converges to a first-order critical point for a sufficiently small stepsize, and establish a *global* sublinear convergence rate. The derived stepsize depends on the worst-case delay and inherent problem sparsity, and furthermore matches the result in existing synchronous algorithms when delay is zero. Numerical evaluations on simulated and real-world datasets demonstrate the power of ASAPP in accelerating convergence in the presence of communications latency, which improves the practicality of distributed PGO.

Preliminaries on Riemannian Optimization

This work heavily uses the first-order geometry of Riemannian manifolds. The reader is referred to [10] for a

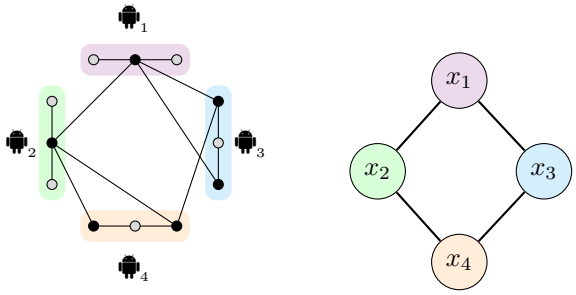
rigorous treatment of this subject. In SLAM, examples of matrix manifolds that frequently appear include the orthogonal group $\mathrm{O}(d)$, special orthogonal group $\mathrm{SO}(d)$, and the special Euclidean group $\mathrm{SE}(d)$. In this work, we use $\mathcal{M} \subseteq \mathcal{E}$ to denote 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 $\eta_1, \eta_2 \in T_x\mathcal{M}$, $\langle \eta_1, \eta_2 \rangle \triangleq \mathrm{tr}(\eta_1^\top \eta_2)$. 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 [10].

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 uses the Riemannian gradient $\mathrm{grad}\, f(x) \in T_x\mathcal{M}$, 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 normal Euclidean gradient $\nabla f(x)$ onto the tangent space, i.e., $\mathrm{grad}\, f(x) = \mathrm{Proj}_{T_x\mathcal{M}} \nabla f(x)$ [10]. We call $x^\star \in \mathcal{M}$ a first-order critical point if $\mathrm{grad}\, f(x^\star) = 0$.

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. [11]–[14] propose a distributed consensus protocol based on Riemannian gradient descent. The key insight which departs from vanilla distributed gradient method is the definition of a set of reshaped cost functions based on the geodesic distance, under which the method provably converges. Similar gradient-based method with line-search has also been proposed by [15]. Choudhary et al. [16] proposes 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 successive over-relaxation (SOR) [3] to solve a relaxed or linearized PGO problem. The two-stage approach [1] is further combined with outlier rejection schemes in [17]. In our recent work [2], we avoid explicit linearization by directly optimizing PGO and its rank-restricted semidefinite relaxations [9]. The proposed solver performs distributed block-coordinate descent over the product of Riemannian manifolds, and provably converge to first-order critical points with global sublinear rate. In a separate line of research, Fan and Murphey [18] propose an 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 (edge direction is omitted). 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 for robot i to optimize its pose estimates x_i, it only needs to receive its neighboring poses through the network. At any time, robot i does not need to know the private poses of any other robots.

B. Asynchronous Parallel Optimization

The aforementioned works are promising, but critically rely on *synchronization* that limits their practical value for networked autonomous systems. However, we note that 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], and have led to the recent development of asynchronous randomized block coordinate and stochastic gradient algorithms – see [4]–[7], [19]–[21] and references therein. We are especially interested in asynchronous parallel schemes for non-convex optimization, which have been studied in [7], [21]. In this work, we generalize these approaches to the setting where the feasible set is a the product of non-convex matrix manifolds, motivated by PGO. Our model of asynchrony is comparable to [20]: workers exchange local models asynchronously during optimization. However, unlike [20], we obviate the need for local averaging to achieve consensus, as each robot is only responsible for updating its own trajectory.

III. Problem Formulation

A. Pose Graph Optimization

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 $\mathcal{R} = \{1, 2, \ldots, n\}$ 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 $\widetilde{T}_{j_s}^{i_\tau} = (\widetilde{R}_{j_s}^{i_\tau}, \widetilde{t}_{j_s}^{i_\tau}) \in \mathrm{SE}(d)$. We assume the following noise

model for our measurements,

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

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

Above, $\underline{T}_{j_s}^{i_\tau} = (\underline{R}_{j_s}^{i_\tau}, \underline{t}_{j_s}^{i_\tau}) \in \text{SE}(d)$ denotes the groundtruth (i.e., noiseless) relative transformation. The isotropic Langevin distribution on rotations [8] plays an analogous role as the multivariate normal distribution on translations. Assuming the relative measurements are free of outliers, the above noise model is thus well-suited and has become the default setup in recent work [2], [8], [9].[1]

Given noisy observations of the form (1) - (2), we seek to find the maximum likelihood configurations of pose graphs for all robots in $\mathcal{R}$. Doing so amounts to the following non-convex program [8],

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

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

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, and directed edges from i_τ to j_s to E are formed when there is a relative measurement from T_{i_τ} to T_{j_s}. Figure 1a shows an example pose graph.

In this paper, we further consider the *rank-restricted semidefinite relaxation* of (P$_1$) [9]. Define the Stiefel manifold as $\text{St}(d, r) \triangleq \{Y \in \mathbb{R}^{r \times d} : Y^\top Y = I_d\}$, where $r \geq d$ [10]. The rank-r relaxation of (P$_1$) is defined as the following non-convex Riemannian optimization problem.

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

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

Observe that for $r = d$, the Stiefel manifold is identical to the orthogonal group $\text{St}(d, d) = \text{O}(d)$. In this case, (P$_2$) is referred to as the *orthogonal relaxation* of (P$_1$), obtained by dropping the determinant constraint on $\text{SO}(d)$. 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. This hierarchy of rank-restricted problems lies at the heart of the so-called Riemannian Staircase procedure [22] for solving the semidefinite relaxation of (P$_1$), which has proven extremely successful in the design of PGO solvers with global optimality guarantees [2], [8], [9]. Once we solve (P$_2$), either globally or locally to a critical point, we can apply a distributed *rounding* procedure (e.g. as detailed in [2]) to obtain an feasible solution to the original MLE problem (P$_1$). Note that (P$_2$) shares the same sparsity structure as encoded by the pose graph.

For the purpose of designing decentralized algorithms (Section IV), it is more convenient to rewrite (P$_1$) and (P$_2$) into a more abstract problem one at the level of robots, which may be done as follows.

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

$$\begin{aligned} \min \sum_{(i,j) \in E_R} & f_{ij}(x_i, x_j) + \sum_{i \in \mathcal{R}} h_i(x_i), \\ \text{s.t.} \quad & x_i \in \mathcal{M}_i,\ \forall i \in \mathcal{R}. \end{aligned} \tag{P}$$

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,

$$x_i = \begin{bmatrix} Y_{i_1} & p_{i_1} & \dots & Y_{i_{n_i}} & p_{i_{n_i}} \end{bmatrix}, \tag{5}$$

$$\mathcal{M}_i = (\text{St}(d, r) \times \mathbb{R}^r)^{n_i}. \tag{6}$$

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. For both (P$_1$) and (P$_2$), 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 Figure 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.

In the literature, neighboring poses are also referred to as the *separators* [1]. 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 Figure 1). This property is crucial to preserve the *privacy* of participating robots [1], [2], i.e., at any time, 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 [6]), in which multiple processors update randomly selected coordinates of the variable concurrently. 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).

[1]For simplicity, in (1) and (2) we have assumed the same noise parameters for all relative measurements. Our approach trivially generalizes to the general case where measurements have distinct measurement noise, as in [2], [8], [9].

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. The 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 the robot's neighbors. We note that 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*. We formally introduce this assumption in Section V.

The communication threads performs the following two operations over the cache.

• **Receive**: After receiving an updated 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 estimate x_i to its neighboring robots. Recall from Section III that robot i does not need to send its private poses, as these pose are not needed by other robots to optimize their estimates.

B. The 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 [23])**.** *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. [24] and is a commonly used tool for analyzing the global behavior of distributed randomized algorithms. We assume that the rate parameter λ is equal and shared among robots. In practice, we can adjust λ based on the extent of network delay and the robots' computational capacity. Using the local clock, the optimization thread performs the following operations in a loop.

• **Read**: For all neighboring robot $j \in \mathrm{N}(i)$, read the values 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 (Figure 1). However, for ease of notation and analysis (Section V), 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,

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

where $\gamma > 0$ is a constant stepsize. Equation (10) gives the simplest update rule that robots can follow, and forms the basis of our convergence analysis in Section V. To further accelerate convergence in practice, state-of-the-art solvers often implement a heuristic known as *preconditioning* [2], [8], [9]. We note that ASAPP can be straightforwardly extended to use preconditioning, by using the following alternative update direction,

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

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 to 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,

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 $k = 0, 1, \ldots$ **do**
2: Select robot $i_k \in \mathcal{R}$ uniformly at random.
3: Read $\hat{x}_{i_k} = x^k_{i_k}$.
4: Read $\hat{x}_{j_k} = x^{k-B(j_k)}_{j_k}$, $\forall j_k \in \mathrm{N}(i_k)$.
5: Compute local gradient $\eta^k_{i_k} = \operatorname{grad} g_{i_k}(\hat{x}_{i_k})$.
6: Update $x^{k+1}_{i_k} = \operatorname{Retr}_{\hat{x}_{i_k}}(-\gamma \eta^k_{i_k})$.
7: Carry over all $x^{k+1}_j = x^k_j$, $\forall j \neq i_k$.
8: **end for**

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 [23], one can also decrease the clock rate λ to create more time for each update.

In addition, we note that although the optimization thread and communication thread run concurrently, minimal 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. In practice, however, due to the large number of poses owned by each robot, the aforementioned synchronization only happens relatively rarely.

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], [19], [20], [24]. 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 is in practice unknown to any of the robot.

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 [23]. Crucially, this implies that 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 $\mathcal{R}$.

Using this result, we can write the iterations of ASAPP from the global view; see Algorithm 1. We use $x^k \triangleq \begin{bmatrix} x^k_1 & x^k_2 & \ldots & x^k_n \end{bmatrix}$ to represent the value of all robots' poses after global iteration k. Note that x lives on the product manifold $\mathcal{M} \triangleq \mathcal{M}_1 \times \mathcal{M}_2 \times \ldots \mathcal{M}_n$. 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), while all other robots do not update (line 7). Note that in the pseudocode, 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 $B(j_k)$ global iterations (line 4).

B. Sufficient Conditions for Convergence

We establish sufficient conditions for ASAPP to converge to first-order critical points.[2] 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 frequency of communication (Section IV-A), the frequency 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 $B > 0$ such that $B(j_k) \leq B$ for all $k \in \mathbb{N}$.*

For both the MLE problem (P_1) and its rank-restricted semidefinite relaxations (P_2), the gradients of the cost functions 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])**.** *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 $\eta \in T_x\mathcal{M}$,*

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

The condition (12) is first proposed by [25] 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,*

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

where $\alpha > 0$ is a constant related to the retraction, and $\Delta > 0$ is the maximum degree of the robot-level graph $G_{\mathcal{R}}$.

[2] Due to space limitation, all proofs are deferred to the appendix.

In (13), the last term on the right hand side sums over quantities from earlier iterations, and is a direct consequence of delay in the system. This term is 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 this term, 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,*

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

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

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

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,*

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

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 expression (15) offers the correct qualitative insights with respect to various problem-specific parameters, which we discuss next.

Relation with maximum delay (B): $\bar{\gamma}$ increases as maximum delay B decreases. Intuitively, as communication becomes increasingly available, each robot may take larger stepsize 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 sparser, robots can use larger stepsize as their problems become increasing 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., [25].

VI. Experimental Results

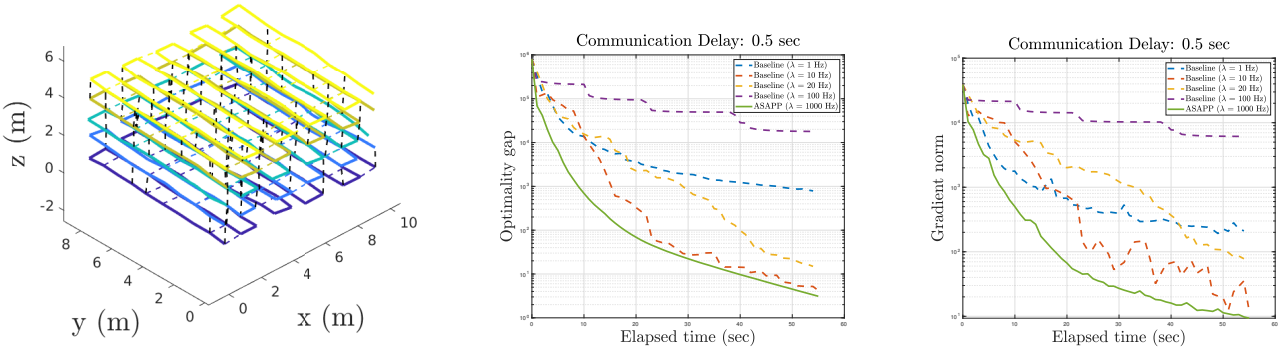
We implement ASAPP in C++ and evaluate its performance on both simulated and real-world PGO datasets. We use ROPTLIB [26] for manifold related computations, and the Robot Operating System (ROS) [27] for inter-robot communication. The Poisson clock is implemented by halting the optimization thread after 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 that 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 16GB memory.

For each PGO problem, we use ASAPP to solve its rank-restricted semidefinite relaxation (P_2) with $r = 5$. 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 the globally optimal solution $x^\star$ is computed using the centralized Cartan-Sync solver [9]. After optimization, we round the solution to $\mathrm{SE}(d)$ and then compute the rotation and translation root mean squared error (RMSE) with respect 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 (Figure 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 standard deviation 2°, and Gaussian translation noise with standard deviation 0.05 m. To simulate a scenario in which robots begin distributed optimization without any prior communication, we initialize each robot's local trajectory using odometry, while the relative transformations between trajectories are uninitialized. As is commonly done in prior work [4]–[7], [19]–[21], in our experiments we select the stepsize empirically, in this case $\gamma = 5 \times 10^{-4}$.

In the first experiment, we simulate a fixed communication delay by letting each robot communicate every 0.5 second. We compare the performance of ASAPP with a baseline algorithm in which each robot uses the second-order Riemannian trust-region (RTR) method to optimize its local variable. RTR has emerged as the default solver in the synchronous setting 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 the baseline at different optimization rates (i.e. frequency at which robots update their local trajectories).



(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 $f(x^k) - f(x^\star)$. (c) Riemannian gradient norm $\left\|\text{grad}\, f(x^k)\right\|$. Note that ASAPP outperforms all variants of the baseline in terms of both optimality gap and gradient norm.

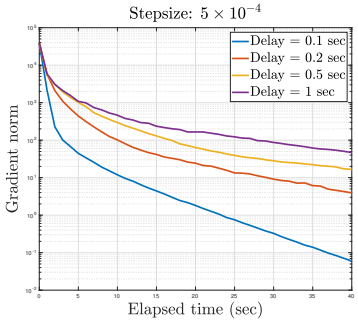


Fig. 3: Convergence speed of ASAPP (stepsize $\gamma = 5 \times 10^{-4}$) with varying communication delay. As delay decreases, convergence becomes faster because robots have access to more up-to-date information from each other.

Figure 2b shows the optimality gaps achieved by the evaluated algorithms as a function of wall clock time. The corresponding reduction in Riemannian gradient norm is shown in Figure 2c. Clearly, 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 (blue dashed curve), the baseline algorithm operates in the synchronous regime and shows convergence behavior. The empirical convergence speed is nevertheless slow, as each robot needs to wait for up-to-date information to arrive after each iteration. At a high rate, e.g., $\lambda = 100$ Hz (purple dashed curve), robots essentially behave asynchronously. However, since RTR does not regulate the stepsize at each iteration, robots often significantly alter their solutions in the wrong direction (as a result of using outdated information), which again leads to slow convergence or even non-convergence. This comparison demonstrates the advantages offered by ASAPP: while asynchrony effectively reduces the penalty on execution time, the use of conservative stepsize also counters the negative effects of delay and ensures convergence.

In addition, we also evaluate ASAPP under a wide range of communication delays. Due to space limitation, we only show performance in terms of gradient norm in Figure 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

To further demonstrate the effectiveness of ASAPP, we evaluate the algorithm on several benchmark SLAM datasets. Each dataset is divided into 5 segments simulating a collaborative SLAM mission with 5 robots.[3] To accelerate empirical convergence, we run ASAPP with preconditioning as described in Section IV-B. To minimize communication usage during initialization, we initialize robots' trajectory estimates by propagating measurements along a spanning tree of the pose graph. On the synthetic `Sphere` dataset, however, the spanning tree initialization gives a particularly poor initial guess, and we use the distributed chordal initialization [1] instead.

In Table I, we report the performance of ASAPP after running for 60s under a fixed communication delay of 0.1 second. We first note that with preconditioning, ASAPP can afford larger stepsizes (recall that without preconditioning, we need to use a stepsize of 5×10^{-4} in the previous section). This demonstrates the power of preconditioning in countering the poor conditioning of the optimization problem. Furthermore, on all datasets, ASAPP is convergent and significantly reduces the optimality gap from the initial solution, and the small rotation and translation errors (last two columns) indicate that the solutions are near-optimal.

VII. Conclusion

We presented ASAPP, the first *parallel* and *provably delay-tolerant* algorithm to solve distributed pose graph optimization and its rank-restricted semidefinite relaxations.

[3]Due to space limitations, figures of these datasets are postponed to the appendix.

TABLE I: Performance of ASAPP with preconditioning on benchmark PGO datasets. Each dataset is divided into trajectories of 5 robots to simulate a collaborative SLAM scenario. We then run ASAPP for 60 s under a fixed communication delay of 0.1 second. For each dataset, we report its size, the stepsize used by ASAPP, and the optimality gaps of the initial and final solution. We also report the final gradient norm achieved with ASAPP, as well as the corresponding rotation and translation root mean squared errors (RMSE).

Datasets	# Poses	# Edges	Stepsize	Init. Opt. Gap	Final Opt. Gap	Gradnorm	Rot. Error [deg]	Trans. Error [m]
CSAIL (2D)	1045	1171	0.1	628.7	0.10	0.55	0.22	0.004
Intel Research Lab (2D)	1228	1483	0.1	342.2	0.82	0.62	0.99	0.003
Parking Garage (3D)	1661	6275	0.01	418.2	0.22	0.17	3.00	0.01
Sphere (3D)	2500	4949	0.2	694.3	14.7	2.79	1.32	0.01

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. We performed numerical evaluations on both simulation and real-world datasets, and results 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 is based on a worst-case analysis and involves unknown constants such as the maximum delay B and Lipschitz constant L. Future work could consider a strategy (e.g., based on average-case analysis) that is less conservative and furthermore explicitly deal with the unknown constants in the expression. Another open question is conditions under which stronger performance guarantees may be hold, i.e., local or global extrema. Doing so is challenging due to the geometric aspects of the analysis coming to bear on the non-convexity of the problem, and that typical efforts to improve upon convergence to stationarity require operating in *unconstrained* settings [28], [29].

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," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1286–1311, 2017.

[2] Y. Tian, K. Khosoussi, and J. P. How, "Block-coordinate descent on the riemannian staircase for certifiably correct distributed rotation and pose synchronization," 2019.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23. Prentice hall Englewood Cliffs, NJ, 1989.

[4] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[5] F. Niu, B. Recht, C. Re, and S. Wright, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," *Advances in Neural Information Processing Systems (NIPS)*, 2011.

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

[7] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[8] 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," *The International Journal of Robotics Research*, 2019.

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

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

[11] R. Tron and R. Vidal, "Distributed image-based 3-d localization of camera sensor networks," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 2009.

[12] R. Tron, *Distributed optimization on manifolds for consensus algorithms and camera network localization*. The Johns Hopkins University, 2012.

[13] R. Tron and R. Vidal, "Distributed 3-d localization of camera sensor networks from 2-d image measurements," *IEEE Transactions on Automatic Control*, vol. 59, pp. 3325–3340, Dec 2014.

[14] R. Tron, J. Thomas, G. Loianno, K. Daniilidis, and V. Kumar, "A distributed optimization framework for localization and formation control: Applications to vision-based measurements," *IEEE Control Systems Magazine*, vol. 36, pp. 22–44, Aug 2016.

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

[16] 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.

[17] 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.

[18] T. Fan and T. D. Murphey, "Generalized proximal methods for pose graph optimization," in *The International Symposium on Robotics Research*, 2019.

[19] 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.

[20] 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.

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

[22] N. Boumal, "A riemannian low-rank method for optimization over semidefinite matrices with block-diagonal constraints," 2015.

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

[24] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[25] N. Boumal, P.-A. Absil, and C. Cartis, "Global rates of convergence for nonconvex optimization on manifolds," *IMA Journal of Numerical Analysis*, vol. 39, pp. 1–33, 02 2018.

[26] 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.

[27] 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.

[28] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points – online stochastic gradient for tensor decomposition," in *Conference on Learning Theory*, pp. 797–842, 2015.

[29] S. Paternain, A. Mokhtari, and A. Ribeiro, "A newton-based method for nonconvex optimization with fast evasion of saddle points," *SIAM Journal on Optimization*, vol. 29, no. 1, pp. 343–368, 2019.

APPENDIX

A. Proof of Lemma 2

Proof. Suppose that at iteration k, robot i_k is selected for update. Recall that $x^k = \begin{bmatrix} x_1^k & x_2^k & \dots & x_n^k \end{bmatrix} \in \mathcal{M}$, where $\mathcal{M}$ is the product manifold $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \times \dots \mathcal{M}_n$ (Section V-A). For all $k \in \mathbb{N}$, define the aggregate tangent vector $\eta^k \in T_{x^k}\mathcal{M}$ as,

$$\eta_i^k \triangleq \begin{cases} \eta_{i_k}^k, & \text{if } i = i_k, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

By definition of η^k, the update step in Algorithm 1 (line 6-7) is equivalent to,

$$x^{k+1} = \mathrm{Retr}_{x^k}(-\gamma \eta^k). \tag{18}$$

By Lemma 1, f has Lipschitz-type gradient for pullbacks. Therefore,

$$f(x^{k+1}) - f(x^k) \leq -\gamma \langle \mathrm{grad}\, f(x^k), \eta^k \rangle + \frac{L\gamma^2}{2} \left\| \eta^k \right\|^2 \quad = -\gamma \langle \mathrm{grad}_{i_k} f(x^k), \eta_{i_k}^k \rangle + \frac{L\gamma^2}{2} \left\| \eta_{i_k}^k \right\|^2. \tag{19}$$

Using the equality $\langle \eta_1, \eta_2 \rangle = \frac{1}{2}[\|\eta_1\|^2 + \|\eta_2\|^2 - \|\eta_1 - \eta_2\|^2]$, we can convert the inner product that appears on the right hand side of (19) into,

$$\langle \mathrm{grad}_{i_k} f(x^k), \eta_{i_k}^k \rangle = \frac{1}{2} \left[\left\| \mathrm{grad}_{i_k} f(x^k) \right\|^2 + \left\| \eta_{i_k}^k \right\|^2 - \left\| \mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k \right\|^2 \right]. \tag{20}$$

Substitute (20) into (19). After collecting relevant terms, we have,

$$\begin{aligned} f(x^{k+1}) - f(x^k) &\leq -\gamma \langle \mathrm{grad}_{i_k} f(x^k), \eta_{i_k}^k \rangle + \frac{L\gamma^2}{2} \left\| \eta_{i_k}^k \right\|^2 \\ &= -\frac{\gamma}{2} \left[\left\| \mathrm{grad}_{i_k} f(x^k) \right\|^2 + \left\| \eta_{i_k}^k \right\|^2 - \left\| \mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k \right\|^2 \right] + \frac{L\gamma^2}{2} \left\| \eta_{i_k}^k \right\|^2 \\ &= -\frac{\gamma}{2} \left\| \mathrm{grad}_{i_k} f(x^k) \right\|^2 - \frac{\gamma - L\gamma^2}{2} \left\| \eta_{i_k}^k \right\|^2 + \frac{\gamma}{2} \left\| \mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k \right\|^2. \end{aligned} \tag{21}$$

We proceed to bound the last term on the right hand side of (21). Recall from (8) and (9) that $\eta_{i_k}^k$ is formed with stale gradients,

$$\eta_{i_k}^k = \mathrm{grad}_{i_k} h_{i_k}(x_{i_k}^k) + \sum_{j_k \in \mathrm{N}(i_k)} \mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k - B(j_k)}), \tag{22}$$

where we abbreviate the notation by defining $e_k \triangleq (i_k, j_k) \in E_{\mathcal{R}}$. In contrast, the Riemannian gradient $\mathrm{grad}_{i_k} f(x^k)$ is by definition formed using up-to-date variables,

$$\mathrm{grad}_{i_k} f(x^k) = \mathrm{grad}_{i_k} h_{i_k}(x_{i_k}^k) + \sum_{j_k \in \mathrm{N}(i_k)} \mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k). \tag{23}$$

Note that the only difference between (22) and (23) is that delayed information is used in (22). In order to form the last term on the right hand side of (21), we subtract (22) from (23) and compute the norm distance. Subsequently, we use the triangle inequality to obtain an upper bound on this norm distance,

$$\begin{aligned} \left\| \mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k \right\| &= \left\| \sum_{e_k = (i_k, j_k) \in E_{\mathcal{R}}} \left[\mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k - B(j_k)}) \right] \right\| \\ &\leq \sum_{e_k = (i_k, j_k) \in E_{\mathcal{R}}} \underbrace{\left\| \mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k - B(j_k)}) \right\|}_{\epsilon(i_k, j_k)}. \end{aligned} \tag{24}$$

Next, we proceed to bound each $\epsilon(i_k, j_k)$ term. To do so, we use the fact that for a real-valued function f defined over a matrix submanifold $\mathcal{M} \subseteq \mathcal{E}$, its Riemannian gradient is obtained as the orthogonal projection of the Euclidean gradient onto the tangent space (see [10, Section. 3.6.1]),

$$\mathrm{grad}\, f(x) = \mathrm{Proj}_{T_x \mathcal{M}} \nabla f(x). \tag{25}$$

Substituting (25) into the right hand side of (24), it holds that,

$$\begin{aligned}\epsilon(i_k, j_k) &= \left\| \mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \mathrm{grad}_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k-B(j_k)}) \right\| \\ &= \left\| \mathrm{Proj}_{T_{x_{i_k}^k}} \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \mathrm{Proj}_{T_{x_{i_k}^k}} \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k-B(j_k)}) \right\|.\end{aligned} \tag{26}$$

Furthermore, since the tangent space is identified as a linear subspace of the ambient space $\mathcal{E}$ [10, Section 3.5.7], the orthogonal projection operation is a *non-expansive* mapping, i.e.,

$$\begin{aligned}\epsilon(i_k, j_k) &= \left\| \mathrm{Proj}_{T_{x_{i_k}^k}} \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \mathrm{Proj}_{T_{x_{i_k}^k}} \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k-B(j_k)}) \right\| \\ &\leq \left\| \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k-B(j_k)}) \right\|.\end{aligned} \tag{27}$$

Since the norm distance with respect to i_k is no greater than the overall norm distance, we furthermore have,

$$\epsilon(i_k, j_k) \leq \left\| \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \nabla_{i_k} f_{e_k}(x_{i_k}^k, x_{j_k}^{k-B(j_k)}) \right\| \leq \left\| \nabla f_{e_k}(x_{i_k}^k, x_{j_k}^k) - \nabla f_{e_k}(x_{i_k}^k, x_{j_k}^{k-B(j_k)}) \right\|. \tag{28}$$

In (P), the Euclidean gradient of each cost function f_{e_k} is Lipschitz continuous. Furthermore, it is straightforward to show that the Lipschitz constant of f_{e_k} is always less than or equal to the Lipschitz constant of the overall cost function f. Denote the latter as $C > 0$. By definition, we thus have,

$$\epsilon(i_k, j_k) \leq C \left\| x_{j_k}^k - x_{j_k}^{k-B(j_k)} \right\|. \tag{29}$$

In addition, in [2] we have shown that the Riemannian version of the Lipschitz constant L that appears in Lemma 1 is always greater than or equal to the Euclidean Lipschitz constant C (see Lemma 5 in [2]). Thus,

$$\epsilon(i_k, j_k) \leq L \left\| x_{j_k}^k - x_{j_k}^{k-B(j_k)} \right\|. \tag{30}$$

We proceed by bounding the norm on the right hand side of (30). Writing the subtraction as a telescoping sum and invoking triangle inequality, we first obtain,

$$\left\| x_{j_k}^k - x_{j_k}^{k-B(j_k)} \right\| = \left\| \sum_{k'=k-B(j_k)}^{k-1} \left(x_{j_k}^{k'+1} - x_{j_k}^{k'} \right) \right\| \leq \sum_{k'=k-B(j_k)}^{k-1} \left\| x_{j_k}^{k'+1} - x_{j_k}^{k'} \right\|. \tag{31}$$

Recall that for all j_k and iterations k', the next iterate $x_{j_k}^{k'+1}$ is obtained from $x_{j_k}^{k'}$ via the following update,

$$x_{j_k}^{k'+1} = \mathrm{Retr}_{x_{j_k}^{k'}}(-\gamma \eta_{j_k}^{k'}). \tag{32}$$

Furthermore, from Lemma 5 in [2], we know that for each manifold $\mathcal{M}_i$, there exists a corresponding constant $\alpha_i > 0$ such that the Euclidean distance from the initial point to the new point after retraction is always bounded by α_i, i.e.,

$$\|\mathrm{Retr}_{x_i}(\eta_i) - x_i\| \leq \alpha_i \|\eta_i\|, \ \forall x \in \mathcal{M}, \ \forall \eta_i \in T_{x_i}\mathcal{M}. \tag{33}$$

Equation (33) thus provides a way to bound the term on the right hand side of (31),

$$\left\| x_{j_k}^k - x_{j_k}^{k-B(j_k)} \right\| \leq \sum_{k'=k-B(j_k)}^{k-1} \left\| \mathrm{Retr}_{x_{j_k}^{k'}}(-\gamma \eta_{j_k}^{k'}) - x_{j_k}^{k'} \right\| \leq \sum_{k'=k-B(j_k)}^{k-1} \alpha_{j_k} \left\| \gamma \eta_{j_k}^{k'} \right\|. \tag{34}$$

To remove the dependency on α_{j_k}, let $\alpha \triangleq \max_{i \in \mathcal{R}} \alpha_i$. We thus have,

$$\left\| x_{j_k}^k - x_{j_k}^{k-B(j_k)} \right\| \leq \alpha\gamma \sum_{k'=k-B(j_k)}^{k-1} \left\| \eta_{j_k}^{k'} \right\|. \tag{35}$$

We can further more use the bounded delay assumption (Assumption 1) to replace $B(j_k)$ with B,

$$\left\| x_{j_k}^k - x_{j_k}^{k-B(j_k)} \right\| \leq \alpha\gamma \sum_{k'=k-B(j_k)}^{k-1} \left\| \eta_{j_k}^{k'} \right\| \leq \alpha\gamma \sum_{k'=k-B}^{k-1} \left\| \eta_{j_k}^{k'} \right\|. \tag{36}$$

Substituting (36) into (30), we have,

$$\epsilon(i_k, j_k) \leq \alpha\gamma L \sum_{k'=k-B}^{k-1} \left\| \eta_{j_k}^{k'} \right\|. \tag{37}$$

Substituting (37) into (24), we then have,

$$\left\|\mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k\right\| \leq \sum_{j_k \in \mathrm{N}(i_k)} \epsilon(i_k, j_k) \leq \alpha\gamma L \sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|. \tag{38}$$

Squaring both sides of (38), we obtain,

$$\left\|\mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k\right\|^2 \leq \left(\alpha\gamma L \sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|\right)^2 = \alpha^2\gamma^2 L^2 \left(\sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|\right)^2. \tag{39}$$

Recall that the sum of squares inequality states that $(\sum_{i=1}^n a_i)^2 \leq n \sum_{i=1}^n a_i^2$. This gives an upper bound on the squared term in (39),

$$\left\|\mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k\right\|^2 \leq \alpha^2\gamma^2 L^2 B \Delta_{i_k} \sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|^2 \leq \alpha^2\gamma^2 L^2 B \Delta \sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|^2, \tag{40}$$

where $\Delta_{i_k} \leq \Delta$ is robot i_k's degree in the robot-level graph $G_{\mathcal{R}}$. Finally, substituting (40) in (21) concludes the proof,

$$f(x^{k+1}) - f(x^k) \leq -\frac{\gamma}{2}\left\|\mathrm{grad}_{i_k} f(x^k)\right\|^2 - \frac{\gamma - L\gamma^2}{2}\left\|\eta_{i_k}^k\right\|^2 + \frac{\gamma}{2}\left\|\mathrm{grad}_{i_k} f(x^k) - \eta_{i_k}^k\right\|^2 \tag{41}$$

$$\leq -\frac{\gamma}{2}\left\|\mathrm{grad}_{i_k} f(x^k)\right\|^2 - \frac{\gamma - L\gamma^2}{2}\left\|\eta_{i_k}^k\right\|^2 + \frac{\alpha^2\gamma^3 L^2 B\Delta}{2} \sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|^2. \tag{42}$$

□

B. Proof of Theorem 1

Proof. Since $f^\star$ is a global lower bound on f, we can obtain the following inequality,

$$f^\star - f(x^0) \leq \mathbb{E}_{i_{0:K-1}}\left[f(x^K)\right] - f(x^0) = \mathbb{E}_{i_{0:K-1}}\left[\sum_{k=0}^{K-1}(f(x^{k+1}) - f(x^k))\right], \tag{43}$$

where the right hand side rewrites the middle term as a telescoping sum. Using the linearity of expectation, we obtain,

$$f^\star - f(x^0) \leq \mathbb{E}_{i_{0:K-1}}\left[\sum_{k=0}^{K-1}(f(x^{k+1}) - f(x^k))\right] = \sum_{k=0}^{K-1} \mathbb{E}_{i_{0:k}}\left[f(x^{k+1}) - f(x^k)\right]. \tag{44}$$

For each expectation term, applying the law of total expectation yields,

$$f^\star - f(x^0) \leq \sum_{k=0}^{K-1} \mathbb{E}_{i_{0:k-1}}\left[\mathbb{E}_{i_k|i_{0:k-1}}[f(x^{k+1}) - f(x^k)]\right]. \tag{45}$$

Next, recall that Lemma 2 already gives an upper bound on the innermost term of (45). Substituting this upper bound into (45) gives,

$$f^\star - f(x^0) \leq \sum_{k=0}^{K-1} \mathbb{E}_{i_{0:k-1}}\left[\mathbb{E}_{i_k|i_{0:k-1}}\left[-\frac{\gamma}{2}\left\|\mathrm{grad}_{i_k} f(x^k)\right\|^2 - \frac{\gamma - L\gamma^2}{2}\left\|\eta_{i_k}^k\right\|^2 + \frac{\Delta B\alpha^2 L^2\gamma^3}{2} \sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|^2\right]\right]. \tag{46}$$

Next, we simplify individual terms on the right hand side of (46). We start with the first conditional expectation term. Using the definition of conditional expectation,

$$\mathbb{E}_{i_k|i_{0:k-1}}\left[-\frac{\gamma}{2}\left\|\mathrm{grad}_{i_k} f(x^k)\right\|^2\right] = -\frac{\gamma}{2}\sum_{i=1}^n P(i_k = i|i_{0:k-1})\left\|\mathrm{grad}_i f(x^k)\right\|^2. \tag{47}$$

Recall that $\{i_k\}$ are i.i.d. random variables uniformly distributed over 1 to n (Section V-A). Setting $P(i_k = i|i_{0:k-1}) = 1/n$ thus gives,

$$\mathbb{E}_{i_k|i_{0:k-1}}\left[-\frac{\gamma}{2}\left\|\mathrm{grad}_{i_k} f(x^k)\right\|^2\right] = -\frac{\gamma}{2}\sum_{i=1}^n \frac{1}{n}\left\|\mathrm{grad}_i f(x^k)\right\|^2 = -\frac{\gamma}{2n}\left\|\mathrm{grad}\, f(x^k)\right\|^2. \tag{48}$$

Similarly, for the third conditional expectation in (46), we note that,

$$\mathbb{E}_{i_k|i_{0:k-1}}\left[\sum_{j_k \in \mathrm{N}(i_k)} \sum_{k'=k-B}^{k-1} \left\|\eta_{j_k}^{k'}\right\|^2\right] = \sum_{i=1}^n \frac{1}{n} \sum_{j \in \mathrm{N}(i)} \sum_{k'=k-B}^{k-1} \left\|\eta_j^{k'}\right\|^2. \tag{49}$$

In equation (49), exchange the order of summations and collect relevant terms,

$$\sum_{i=1}^{n}\frac{1}{n}\sum_{j\in\mathrm{N}(i)}\sum_{k'=k-B}^{k-1}\left\|\eta_j^{k'}\right\|^2=\frac{1}{n}\sum_{k'=k-B}^{k-1}\sum_{i=1}^{n}\sum_{j\in\mathrm{N}(i)}\left\|\eta_j^{k'}\right\|^2=\frac{2}{n}\sum_{k'=k-B}^{k-1}\left\|\eta^{k'}\right\|^2. \tag{50}$$

Using our simplified expressions for the first and third term on the right hand side of (46), we obtain,

$$f^\star-f(x^0)\leq\sum_{k=0}^{K-1}\mathbb{E}_{i_{0:k-1}}\left[-\frac{\gamma}{2n}\left\|\operatorname{grad}f(x^k)\right\|^2-\mathbb{E}_{i_k|i_{0:k-1}}\left[\frac{\gamma-L\gamma^2}{2}\left\|\eta_{i_k}^k\right\|^2\right]+\frac{\Delta B\alpha^2L^2\gamma^3}{n}\sum_{k'=k-B}^{k-1}\left\|\eta^{k'}\right\|^2\right]. \tag{51}$$

Next, using the independence relations and the linearity of expectation, we obtain,

$$f^\star-f(x^0)\leq\mathbb{E}_{i_{0:K-1}}\sum_{k=0}^{K-1}\left[-\frac{\gamma}{2n}\left\|\operatorname{grad}f(x^k)\right\|^2-\frac{\gamma-L\gamma^2}{2}\left\|\eta^k\right\|^2+\frac{\Delta B\alpha^2L^2\gamma^3}{n}\sum_{k'=k-B}^{k-1}\left\|\eta^{k'}\right\|^2\right]. \tag{52}$$

At this point, our bound still involves the squared norms of update vectors from earlier iterations (last term on the right hand side). To simplify the bound further, note that,

$$\sum_{k=0}^{K-1}\sum_{k'=k-B}^{k-1}\left\|\eta^{k'}\right\|^2\leq B\sum_{k=0}^{K-1}\left\|\eta^k\right\|^2. \tag{53}$$

Using the above inequality in (52), we obtain,

$$f^\star-f(x^0)\leq\mathbb{E}_{i_{0:K-1}}\sum_{k=0}^{K-1}\left[-\frac{\gamma}{2n}\left\|\operatorname{grad}f(x^k)\right\|^2+\underbrace{\left(\frac{\Delta\alpha^2B^2L^2\gamma^3}{n}+\frac{L\gamma^2-\gamma}{2}\right)}_{A_1(\gamma)}\left\|\eta^k\right\|^2\right]. \tag{54}$$

We establish a sufficient condition on γ such that $A_1(\gamma)$ as a whole is nonpositive. Let us define $\rho\triangleq\Delta/n$. Consider the following factorization of $A_1(\gamma)$,

$$A_1(\gamma)=\frac{\gamma}{2}\underbrace{\left(2\rho\alpha^2B^2L^2\gamma^2+L\gamma-1\right)}_{A_2(\gamma)}. \tag{55}$$

Note that $A_2(\gamma)$ is the same as (14) in Theorem 1. For the moment, suppose that we can find $\gamma>0$ such that $A_2(\gamma)\leq 0$. This implies that $A_1(\gamma)\leq 0$, and we can thus drop this term on the right hand side of (54),

$$f^\star-f(x^0)\leq\leq-\frac{\gamma}{2n}\sum_{k=0}^{K-1}\mathbb{E}_{i_{0:K-1}}\left[\left\|\operatorname{grad}f(x^k)\right\|^2\right]. \tag{56}$$

Replacing the expected value at each iteration with the minimum across all iterations, we have,

$$f^\star-f(x^0)\leq-\frac{\gamma K}{2n}\min_{k\in[K-1]}\mathbb{E}_{i_{0:K-1}}\left[\left\|\operatorname{grad}f(x^k)\right\|^2\right]. \tag{57}$$

Finally, rearranging the last inequality yields,

$$\min_{k\in[K-1]}\mathbb{E}_{i_{0:K-1}}\left[\left\|\operatorname{grad}f(x^k)\right\|^2\right]\leq\frac{2n(f(x^0)-f^\star)}{\gamma K}. \tag{58}$$

Thus we have proved the main part of Theorem 1. To prove the expression (15), note that if $B=0$ (i.e., there is no delay), the condition $A_2(\gamma)\leq 0$ entails $L\gamma\leq 1$. In this case, we can thus set the upper bound $\bar{\gamma}$ to $1/L$. On the other hand, if $B>0$, $A_2(\gamma)$ becomes a quadratic function of γ, and furthermore has the following positive root,

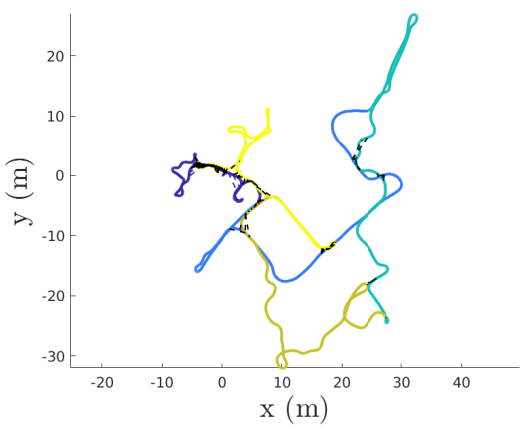
$$\bar{\gamma}=\frac{\sqrt{1+8\rho\alpha^2B^2}-1}{4\rho\alpha^2B^2L}>0. \tag{59}$$

It is straightforward to verify that $A_2(\gamma)\leq 0$ for all $\gamma\in(0,\bar{\gamma}]$. Therefore, we have proved that the condition $A_2(\gamma)\leq 0$ is ensured by the following choice of $\bar{\gamma}$,

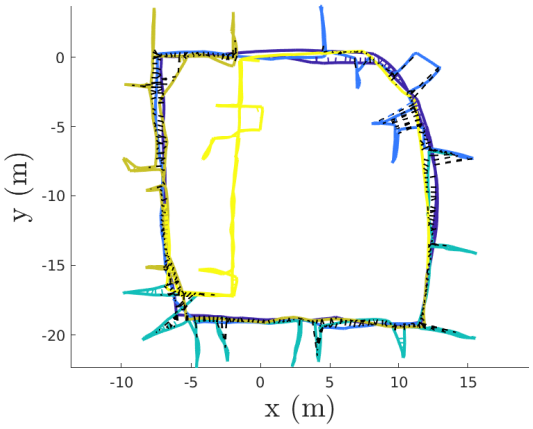
$$\bar{\gamma}=\begin{cases}\frac{\sqrt{1+8\rho\alpha^2B^2}-1}{4\rho\alpha^2B^2L}, & B>0,\\ 1/L, & B=0.\end{cases} \tag{60}$$

In particular, under this choice, ASAPP converges globally to first-order critical points, with an associated convergence rate given in (58).

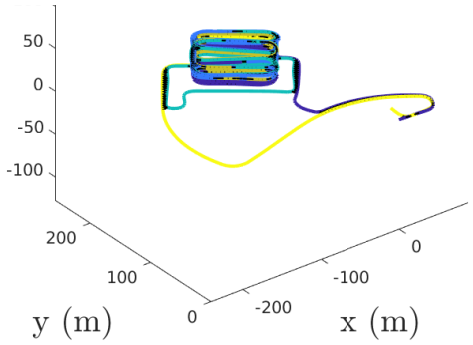
□



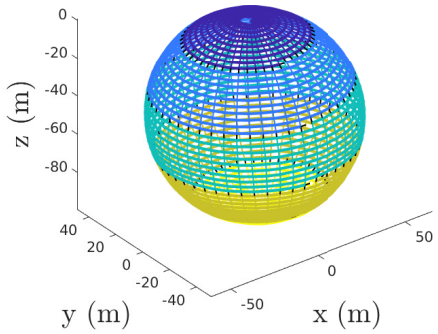
(a) `CSAIL`



(b) `Intel Research Lab`



(c) `Parking Garage`



(d) `Sphere`

Fig. 4: Trajectory estimates returned by ASAPP on benchmark datasets. Each dataset contains trajectories of 5 robots (different colors). Inter-robot measurements (loop closures) are shown as black dashed lines. (a) `CSAIL`; (b) `Intel Research Lab`; (c) `Parking garage`; (d) `Sphere`.