riMESA: Consensus ADMM for Real-World Collaborative SLAM

Daniel McGann and Michael Kaess

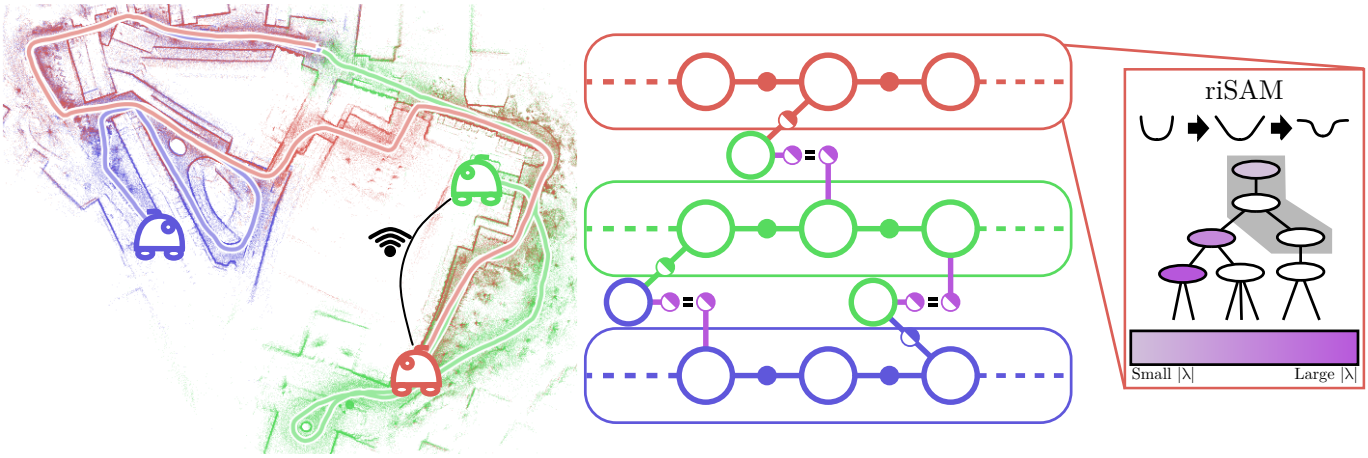


Fig. 1: An illustration of riMESA operating on real-world data (`kth_r3_00_proradio`). riMESA estimates the state of a multi-robot team (○,○,○) from noisy, potentially incorrect measurements (●,●,●) using only sparse, unreliable communication (📶). riMESA is a C-ADMM-based distributed optimization algorithm in which robots locally constrain shared state using "biased priors" (●). Over time, as communication is available, riMESA tightens equality constraints with dual variables (λ) to provide consistent solutions for the team. Meanwhile, robots incorporate new measurements efficiently using the riSAM algorithm, which handles potential outlier measurements (◔) using M-Estimations and an incremental version of Graduated Non-Convexity, which efficiently updates only the relevant subproblem (■) at each timestep.

***Abstract*—Collaborative Simultaneous Localization and Mapping (C-SLAM) is a fundamental capability for multi-robot teams as it enables downstream tasks like planning and navigation. However, existing C-SLAM back-end algorithms that are required to solve this problem struggle to address the practical realities of real-world deployments (i.e. communication limitations, outlier measurements, and online operation). In this paper we propose *R*obust *I*ncremental *M*anifold *E*dge-based *S*eparable *A*DMM (riMESA) – a robust, incremental, and distributed C-SLAM back-end that is resilient to outliers, reliable in the face of limited communication, and can compute accurate state estimates for a multi-robot team in real-time. Through the development of riMESA, we, more broadly, make an argument for the use of Consensus Alternating Direction Method of Multipliers (C-ADMM) as a theoretical foundation for distributed optimization tasks in robotics like C-SLAM due to its flexibility, accuracy, and fast convergence. We conclude this work with an in-depth evaluation of riMESA on a variety of C-SLAM problem scenarios and communication network conditions using both synthetic and real-world C-SLAM data. These experiments demonstrate that riMESA is able to generalize across conditions, produce accurate state estimates, operate in real-time, and outperform the accuracy of prior works by a factor >7x on real-world datasets.**

***Index Terms*—Simultaneous Localization and Mapping (SLAM), Multi-Robot Systems, Optimization, Robust Perception**

I. Introduction

Collaborative Simultaneous Localization and Mapping (C-SLAM) is a fundamental capability for multi-robot teams [1]. A key component of C-SLAM systems is the back-end algorithm that estimates the state of the team from distributed, noisy measurements [2]. However, existing back-ends struggle with the practical conditions experienced by multi-robot teams in the real world. During field deployments (e.g. search and rescue, forestry inspection, or scientific exploration [3–5]), multi-robot teams cannot assume the presence of communication infrastructure, as existing infrastructure may have been damaged (e.g. disaster response) or may not exist (e.g. remote forest, deep sea, lunar). Rather, teams can only assume access to an ad-hoc network that permits unreliable, sparse communication. Additionally, these teams cannot assume that their sensor measurements are perfect. Instead, teams must accept that due to perceptual aliasing,[1] front-end processes (e.g. loop-closure detection) will produce erroneous outlier measurements. Despite these challenges, multi-robot teams need accurate and timely estimates to support downstream tasks like planning and navigation. Existing C-SLAM back-end algorithms struggle to operate under these conditions, either requiring reliable network connectivity, needing impractical computation time, or failing to provide accurate, robust results.

In this paper, we present Consensus Alternating Direction Method of Multipliers (C-ADMM) as a framework to design C-SLAM back-ends that can handle these challenging conditions and collaboratively produce high-quality state estimates for multi-robot teams operating in the real world. We specifically propose **R**obust **I**ncremental **M**anifold **E**dge-based **S**eparable **A**DMM (riMESA) – a robust, incremental, and distributed C-SLAM back-end designed to meet the challenges of real-world operations that outperforms the accuracy of prior works DDF-SAM2 [6] and DLGBP [7] by a factor of **>7x** on real-world C-SLAM problems (Fig. 1).

The rest of the paper is structured as follows. First, we concretely define the C-SLAM problem that multi-robot teams contend with during real-world deployments and the communications conditions under which it must be solved. Next, we provide an overview of prior works and discuss their ability to solve our target problem. We then introduce C-ADMM, discuss its application to C-SLAM problems, and outline the riMESA algorithm. We conclude with a rigorous evaluation of riMESA on synthetic and real-world data across a variety of C-SLAM scenarios and communication conditions. To aid future research, we make the riMESA code open-source.[2]

The authors are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. {danmcgann, kaess}@cmu.edu

[1] Perceptual Aliasing – The phenomenon where spatially disparate locations display a similar "perceptual" (i.e. visual or geometric) appearance.

[2] https://github.com/rpl-cmu/rimesa

II. Problem Definition

We seek to solve the generic C-SLAM problem in which a team of robots $\mathcal{R}$ estimates state variables Θ using noisy measurements $\mathcal{M}$. In this multi-robot case, each robot takes a subset of the measurements and observes a subset of the total variables. Importantly, some measurements taken by robots will be inter-robot measurements. These measurements may come from direct observation of other robots, joint observations of environment landmarks, or from a distributed loop-closure system [8]. Inter-robot measurements enable the multi-robot team to collaborate. This can improve their individual state estimates and can ensure all robots' solutions remain in a consistent global frame. Due to inter-robot measurements, multiple robots may observe the same variable. Therefore, let $\Theta_i \subset \Theta$ denote the non-disjoint subset of variables observed by robot $i \in \mathcal{R}$. Additionally, let $\mathcal{M}_i \subset \mathcal{M}$ denote the robot's disjoint subset of measurements.

Remark 1 (Generic C-SLAM vs. PGO)**:** *It is common for C-SLAM back-ends to solve only Pose-Graph Optimization (PGO), a subset of C-SLAM in which all variables are poses $\in SE(N)$ and all measurements are relative poses [9–14]. PGO is useful in many applications and provides structure to exploit in algorithm design. However, it limits map representations (e.g. landmarks) as well as measurement types (e.g. bearing and range), which are highly useful in specific domains (e.g. underwater [15] and space [16]). To ensure our method generalizes, we focus on the generic C-SLAM problem.*

A. Batch C-SLAM

The de-facto standard for SLAM is to formulate the problem as a factor-graph and solve for the state variables via Maximum A-Posteriori (MAP) inference [2]:

$$\Theta_{MAP} = \arg\max_{\Theta \in \Omega} P(\Theta) \prod_{i \in \mathcal{R}} P(\mathcal{M}_i | \Theta_i) \tag{1}$$

where Ω is the product manifold constructed by the manifold of each variable in Θ. When we assume that each measurement $m \in \mathcal{M}$ is affected by zero-mean Gaussian noise with covariance Σ_m this problem can be solved by Nonlinear Least Squares (NLS) optimization [17]:

$$\Theta_{MAP} = \arg\min_{\Theta \in \Omega} \sum_{i \in \mathcal{R}} \sum_{m \in \mathcal{M}_i} \left\| h_m(\Theta_i) - m \right\|^2_{\Sigma_m} \tag{2}$$

where $h_m(\Theta_i)$ is the measurement prediction function that computes the expected measurement from the state estimate.

B. Robust Incremental C-SLAM

There are two key challenges when utilizing the C-SLAM problem formulation for real-world deployments. Firstly, we are not interested in just solving this optimization problem once. Rather, for practical robotic applications, we need to solve this problem at every timestep, with the advantage that we have access to the previous solution. Secondly, this NLS formulation is inherently sensitive to measurements that do not exhibit Gaussian noise (i.e. outliers). Due to imperfect processes used to derive our measurements $\mathcal{M}$, such outliers are common [2]. Moreover, due to perceptual aliasing, we expect that outliers caused by incorrect data association are inevitable given sufficient operation time. This inevitability, combined with the impact that outliers have on NLS based C-SLAM solvers, necessitates that we extend our algorithms to handle such outliers. Combining these challenges, we define the robust incremental C-SLAM problem:

$$\begin{aligned} \Theta^t_{MAP} &= \arg\min_{\Theta^t \in \Omega^t} \sum_{i \in \mathcal{R}} \sum_{m \in \mathcal{I}^t_i} \left\| h_m(\Theta^t_i) - m \right\|^2_{\Sigma_m} \\ \text{given:} \quad & \Theta^{t-1}_{MAP} \end{aligned} \tag{3}$$

where $\mathcal{I}_i \subseteq \mathcal{M}_i$ are the "true" inlier measurements taken by robot i and (3) must be solved online to report results with minimal delay to downstream tasks. For completeness, let $\mathcal{O}_i$ represent the set of outliers such that $\mathcal{O}_i = \mathcal{M}_i \setminus \mathcal{I}_i$.

C. Maximum-Consensus

An important nuance in the definition of (3) is that the set of "true" inliers $\mathcal{I}$ is unknowable. An intuitive definition of an inlier is any measurement that matches the true state of the world Θ^* up to a reasonable amount of noise. Given that measurements m are affected by Gaussian noise, we can therefore concretely define an inlier as:

$$\left\| h_m(\Theta^*) - m \right\|_{\Sigma_m} < \chi^2_m(T) \tag{4}$$

where we are using the fact that the residual of the measurement will be distributed according to a χ^2 distribution with dimensionality matching m and T is a probability threshold (e.g. $T = 0.95$). However, the true state of the world Θ^* is fundamentally unknowable, making the true set of inliers likewise unknowable. A knowable and, therefore, more practical definition we can use to define inliers is that of "joint-consistency." We define a set of measurements $\mathcal{J}$ as jointly consistent if there exists some state of the world Θ such that:

$$\left\| h_m(\Theta) - m \right\|_{\Sigma_m} < \chi^2_m(T) \quad \forall \quad m \in \mathcal{J} \tag{5}$$

Under the assumptions that our robots will generate inlier measurements with non-trivial probability and that outlier measurements are random, it is *likely* that the largest set of jointly consistent measurements corresponds to the true set of inliers. Identifying this largest set of measurements is commonly referred to as the Maximum-Consensus problem, which, despite being an approximation of our goal, is NP-Hard and generally intractable. Therefore, to solve (3) our algorithms will need to find (either explicitly or implicitly) approximate solutions to the Maximum-Consensus problem.

D. C-SLAM Communication

In the C-SLAM setting, information is distributed across the team, and robots must communicate to solve the problems discussed above. In real-world applications, the team may not have access to existing communication infrastructure. As such, we can only reliably assume that robots can communicate over an ad-hoc network built from the robots themselves. Due to the scale of the environment, hardware constraints, motion of the robots, and sources of interference, this network will be bandwidth limited, time varying, and frequently disconnected.

Let us represent the communication network at a specific time t by a sparse undirected graph $\mathcal{G}^t = (\mathcal{R}, \mathcal{E}^t)$ with nodes made up of the robots $\mathcal{R}$ and edges $\mathcal{E}^t$ defined by the currently available direct connections between robots. This graph is expected to be both sparsely connected at any given timestep and only sparsely connected over time. We assume that robots

perform pairwise communications with connected teammates at a relatively low rate (e.g. 0.1 - 10 Hz), where the rate is practically specified by the bandwidth and latency of the underlying communication hardware and protocols. Due to limited connectivity, robots will not be able to maintain synchronized communication, and instead communications will occur asynchronously whenever pairs of robots can connect. We further assume that connections may drop out at any time. Additionally, even if using connection-oriented protocols (i.e. TCP), such dropout may be observable to only one of the communicating robots (i.e. Two-Generals Failure).[3] Finally, we assume that latency may be large relative to the rate of measurements, and communications may need to be performed in parallel with a C-SLAM algorithm.

This ad-hoc, sparse, and unreliable communication model is restrictive and represents a challenging scenario. However, it is representative of what multi-robot teams may encounter during remote field deployments. Additionally, an algorithm that can operate effectively under this restrictive model can also operate in more optimistic scenarios. If the team has use of communication infrastructure, then $\mathcal{G}^t$ is simply more densely connected at each timestep. If network bandwidth and robot hardware permits fast communication, then the effective communication rate is simply larger. Finally, if communication links are reliable, then robots simply experience less dropout. Therefore, algorithms that can tolerate this restrictive model will be applicable even when communication is less limited.

E. Problem Scope

To support real-world deployments of multi-robot teams, we require algorithms that can solve the robust incremental C-SLAM problem (3) even when robots have access to only ad-hoc, sparse, and unreliable communication (Sec. II-D).

III. Related Work

Prior works have not provided an effective solution to the incremental, robust C-SLAM problem with ad-hoc, sparse, and unreliable communication. To address this problem, prior works have broadly sought to combine the independent lines of research on C-SLAM optimization and robust optimization. For both lines of research, there are a number of approaches, which we discuss briefly below. We then discuss the variety of works that have looked to combine these methods to solve our target problem.

A. C-SLAM Optimization

Prior researchers have broadly focused on three classes of C-SLAM optimization architectures.

1) Centralized C-SLAM Optimization: The conceptually simplest approach to the C-SLAM task is to use a centralized back-end. In these methods, robots communicate their measurements to a central server that performs all computation and sends solutions back to the team. Centralized methods have been demonstrated successfully under a variety of conditions [19–30]. However, they require that $\mathcal{G}^t$ is connected at each timestep and can support significant network traffic to return solutions to all team members, making them inapplicable to scenarios with ad-hoc, sparse, and unreliable communications. Additionally, a central server introduces a single point of failure into the system, making these systems brittle. Finally, as these methods combine all measurements from all the robots into a joint optimization problem, centralization struggles to scale to large teams or long-term operation [31].

2) Decentralized C-SLAM Optimization: Similar to centralized algorithms are decentralized back-ends. In these approaches each robot holds a copy of the global problem and independently computes the solution [32–38]. While decentralization removes any single point of failure from the system and permits robots to continue operation in the face of sparse communication, these methods require even greater bandwidth than centralized methods to send all measurements to every other robot. Additionally, decentralization necessitates that each robot performs expensive and redundant computation. These issues with bandwidth and computation can be mitigated, but not removed, by sparsifying the global problem [33] and sharing computation within robot clusters [37]. As such, decentralized methods will, like centralized approaches, struggle to scale to large multi-robot teams and long-term operation.

3) Distributed C-SLAM Optimization: The final class of C-SLAM back-ends are distributed algorithms. Rather than aggregating the global problem, distributed methods allow agents to solve only local sub-problems and rely on information passed during communication to converge local solutions to the global optimum. Distributed algorithms hold the greatest potential for C-SLAM, as local sub-problems can be solved efficiently, algorithms can be designed to tolerate sparse, unreliable communication, and these algorithms can scale to even very large multi-robot teams. However, existing works have not yet taken full advantage of this potential.

Due to the diversity of distributed optimization techniques [39, 40], a wide variety of distributed solvers have been proposed. Targeting the batch C-SLAM problem (2), prior works have proposed algorithms based on Multi-Block ADMM (MB-ADMM [41]), Distributed Gauss-Seidel (DGS [9]), Distributed Gradient-Descent (GeoD [10]), Majorization-Minimization (MM-PGO [13, 14]), and Distributed Riemannian Gradient Descent (DC2-PGO [12], ASAPP [11]). As batch algorithms, however, these methods are all limited in their ability to address incremental problems. Naively, batch methods could be applied to the incremental case by solving each timestep as a batch problem. However, these methods all require hundreds to thousands of iterations to converge, significantly exceeding the communication and computation availability for any given timestep at which we need a solution. Further, while some of these methods can tolerate asynchronous communication and thus communication dropout between algorithm iterations, they would all require a connected network during the current timestep, which is not guaranteed in real-world scenarios.

Some prior works have proposed distributed algorithms specifically targeting incremental C-SLAM, namely, Distributed Data Fusion Smoothing and Mapping (DDF-SAM [42] and DDF-SAM2 [6]) and Distributed Loopy Gaus-

[3]Two-Generals Failure – Any communication failure where only one party observes that the communication was not successful. Named after the "Two Generals Problem," a thought experiment on unreliable communication [18].

sian Belief Propagation (DLGBP [7]). While these methods can achieve real-time performance, due to their underlying optimization strategies, they struggle to provide accurate and consistent results in many scenarios as illustrated in our experimental results (Sec. VII).

B. Robust Optimization

Often viewed through the lens of single-robot SLAM (though applicable to C-SLAM), numerous prior works have proposed methods to address outlier measurements by approximately solving the Maximum-Consensus problem.

1) Relaxed Consensus: One method to approximately solve the Maximum-Consensus problem is to relax joint-consistency (5) to measures that are more efficient to compute, like Pairwise Consistency (PCM [43]), Group-k Consistency (GkCM [44]), or heuristic consistency (iRRR [45, 46], IPC [47]). With these relaxed consistency definitions, the largest set of consistent measurements can be estimated with max-clique graph algorithms or repeated optimizations. Despite these approximations, relaxed consensus methods remain computationally expensive, and their consistency approximations limit their ability to derive quality solutions.

2) M-Estimation: Another way to relax the Maximum-Consensus problem is via M-Estimation, which wraps measurement errors with a robust kernel ρ to mitigate the effects of outliers on the solution [48, 49]. In doing so, these methods implicitly approximate a solution to the Maximum-Consensus problem. M-Estimation methods are trivially compatible with any NLS solver, including existing fast and efficient incremental SLAM solvers like iSAM2 [50], making them able to support real-time operation. However, depending on the selection of kernel [48, 51, 52], M-Estimation methods construct a cost function that either remains sensitive to outliers or becomes increasingly sensitive to initialization [53, A6.8].

3) Variable Augmentation: Lying somewhere between the relaxed consensus methods and M-Estimation methods, variable augmentation approaches add additional variables to the optimization problem that attempt to classify measurements and down-weight the effect of outliers. The resulting problem is solved by an alternating optimization over the state and augmented variables [54, 55]. However, depending on the exact formulation used for augmented variables, these methods can either construct, via Black-Rangarajan duality, an algorithm that is equivalent to the use of an M-Estimator [56, 57] or one that is computationally inefficient to solve [58].

4) Continuation: Currently, the most promising approaches to robust SLAM optimization are continuation-based methods. These methods take the same theoretical approach as M-Estimation but resolve the induced initialization sensitivity using iterative optimization over a continuation of the problem. Graduated Non-Convexity (GNC) was the first continuation method proposed for batch SLAM problems [59]. While GNC can provide accurate results, its repeated batch optimizations make it computationally inefficient. However, Robust Incremental Smoothing and Mapping (riSAM) demonstrated that these same ideas could be incrementalized to achieve robust and real-time SLAM optimization [60].

C. Robust C-SLAM Optimization

Numerous prior works have looked to make C-SLAM optimization robust by combining existing back-ends with methods for robust optimization. In-turn, these approaches inherit the capabilities and the drawbacks of their components.

Numerous centralized and decentralized methods have proposed incorporating existing robust optimization methods. Schuster et al. [34], CVI-SLAM [22], COVINS [24], and RCVI-SLAM [30] propose utilizing M-Estimators in their optimization, in-turn inheriting their efficiency but also their induced sensitivity to initialization. Similarly, MAPLAB 2.0 [29] proposed the use of Switchable-Constraints, a variable augmentation method that will produce near-equivalent results to M-Estimators. LAMP 2.0 [26], Hydra-Multi [28], and Swarm-SLAM [37] proposed the use of batch GNC to perform their optimization robustly, in-turn inheriting its accuracy but also its inefficiency. CVIDS [27] and DiSCO-SLAM [36] proposed using PCM to filter outliers, in-turn inheriting both its inefficiency and potentially poor performance. In addition to the downsides listed above, all of these methods additionally inherit the communication requirements of centralized and decentralized architectures, making their application to real-world deployments difficult and application to our target problem (Sec. II-E) all but infeasible.

Distributed systems have taken similar approaches. DOOR-SLAM [61] proposed using DGS for distributed optimization combined with PCM for outlier-rejection. In doing so, it inherits the computational inefficiency and limited accuracy of DGS as well as the computational inefficiency and limited accuracy of PCM. The same composition and drawbacks can be found in DCL-SLAM [62], and a similar composition can be found in D^2SLAM [63]. Kimera-Multi [64] composes the distributed back-end ASAPP with a GNC-based robust estimation approach. Both methods are computationally complex, and composing them multiplicatively increases these costs. Thus, while it can provide accurate results, the method is nearly impossible to apply to the real-time scenarios [65]. Intentionally targeting real-time performance, Murai et al. proposed combining the use of standard M-Estimators with their proposed DLGBP optimization method [7]. This combination allows for very efficient incremental solving, as DLGBP can be made efficient and the addition of M-Estimators has little impact on computational performance. However, this also combines DLGBP's poor convergence with a sensitivity to initialization induced by the use of M-Estimators which results in generally poor performance across problem scenarios.

D. Related Work Summary

Evident from this review, while some prior works can support multi-robot teams in limited conditions, we lack back-end algorithms that can generalize to provide solutions to the incremental, robust C-SLAM problem with ad-hoc, sparse, and unreliable communication. In this work we argue for an alternative approach to the C-SLAM back-end – distributed consensus ADMM. We outline how C-ADMM can be applied to distributed C-SLAM optimization and, importantly, how it can be extended to effectively address the incremental, robust C-SLAM problem via our algorithm – riMESA.

IV. Consensus ADMM for Collaborative SLAM

In this section we begin with an overview of C-ADMM theory and then provide an overview of our work that proposed to apply C-ADMM to the C-SLAM problem [66].

A. Standard C-ADMM

We begin by reviewing C-ADMM [67], which has long been a popular, fully distributed method for solving consensus optimization problems of the form:

$$\begin{aligned} \underset{\bar{x}\in\mathbb{R}^{|\mathcal{R}|n}}{\arg\min} \quad & \sum_{i\in\mathcal{R}} f_i(x_i) \\ \text{s.t.} \quad & x_i = x_j \ \forall\ (i,j)\in\mathcal{E} \end{aligned} \tag{6}$$

where $\mathcal{R}$ is the set of all agents, f_i is the local objective of agent i, $x_i \in \mathbb{R}^n$ is the copy of decision variables held by agent i, and $\bar{x} \in \mathbb{R}^{|\mathcal{R}|n}$ is the concatenation of all agents' local decision variables. Agents communicate over an undirected network made up of the agents $\mathcal{R}$ and communication links $\mathcal{E}$. The constraints force neighboring agents to agree on a single solution. By induction, this forces all agents to converge to a single joint solution to the problem.

Applying standard ADMM to problem (6) produces an algorithm that requires centralized updates [68]. To produce a fully distributed algorithm, C-ADMM augments problem (6) with additional variables. For each edge in the communication network $(i,j) \in \mathcal{E}$, C-ADMM introduces two new variables $z_{(i,j)}$ and $z_{(j,i)}$. The variable $z_{(i,j)}$ is held by agent i and can be interpreted as agent i's estimate of agent j's local solution. We will refer to these variables as "edge variables" and use them to rewrite (6) as:

$$\begin{aligned} \underset{\bar{x}\in\mathbb{R}^{|\mathcal{R}|n},\ \bar{z}\in\mathcal{Z}}{\arg\min} \quad & \sum_{i\in\mathcal{R}} f_i(x_i) \\ \text{s.t.} \quad & x_i = z_{(i,j)},\ x_j = z_{(j,i)} \ \forall\ (i,j)\in\mathcal{E} \end{aligned} \tag{7}$$

where $\bar{z}$ is the concatenation of all $z_{(i,j)}$ and $\mathcal{Z}$ is the space of $\mathbb{R}^{2|\mathcal{E}|n}$ which is implicitly constrained such that $z_{(i,j)} = z_{(j,i)}$.

This augmentation increases the number of constraints but does not change their meaning. C-ADMM solves (7) according to the update process:

$$\bar{x}^{k+1} = \underset{\bar{x}\in\mathbb{R}^{|\mathcal{R}|n}}{\arg\min}\ \mathcal{L}(\bar{x}, \bar{z}^k, \bar{\lambda}^k, \beta^k) \tag{8}$$

$$\bar{z}^{k+1} = \underset{\bar{z}\in\mathcal{Z}}{\arg\min}\ \mathcal{L}(\bar{x}^{k+1}, \bar{z}, \bar{\lambda}^k, \beta^k) \tag{9}$$

$$\bar{\lambda}^{k+1} = \bar{\lambda}^k + \beta^k(D\bar{x}^{k+1} - \bar{z}^{k+1}) \tag{10}$$

$$\beta^{k+1} = \alpha\beta^k \tag{11}$$

where $\bar{\lambda}$ is the concatenation of all dual variables (one corresponding to each constraint), $D \in \mathbb{R}^{(2|\mathcal{E}|n)\times(|\mathcal{R}|n)}$ maps each x_i in $\bar{x}$ to all corresponding $z_{(i,j)}$ in $\bar{z}$, β is the penalty term, α is a scaling factor hyper-parameter, k is the iteration count, and $\mathcal{L}$ is the problem's Augmented Lagrangian:

$$\sum_{i\in\mathcal{R}} f_i(x_i) + \sum_{j\in\mathcal{N}_i} \left\langle \lambda_{(i,j)}, x_i - z_{(i,j)} \right\rangle + \frac{\beta}{2}\left\| x_i - z_{(i,j)} \right\|^2 \tag{12}$$

where $\mathcal{N}_i$ are the neighbors of agent i. Solving these iterates can be fully distributed, as (8) can be solved by each agent minimizing its local Augmented Lagrangian independently. After this minimization, C-ADMM stipulates that all agents communicate their results to all neighbors to allow each agent to independently solve (9), (10), and (11). When all f_i are strongly convex, C-ADMM converges linearly [69].

Within existing literature, there are two extensions to the base C-ADMM algorithm that will be relevant to its application to C-SLAM problems. Firstly, standard C-ADMM assumes that each agent's objective relies on all optimization parameters. However, there is a large class of problems where each agent's objective relies on only a subset of the variables. We refer to such problems as "separable" consensus optimization problems, a reference to Separable Optimization Variable ADMM (SOVA) [70]. SOVA proposed a simple but effective extension in which agents hold only the variables required by their local objective and we impose constraints on only the shared variables. Separable problems take the form:

$$\begin{aligned} \underset{\{x_0\in\mathbb{R}^{n_0},\ \ldots,\ x_r\in\mathbb{R}^{n_r}\}}{\arg\min} \quad & \sum_{i\in\mathcal{R}} f_i(x_i) \\ \text{s.t.} \quad & A_{(i,j)}x_i = B_{(i,j)}x_j \ \forall\ (i,j)\in\mathcal{E} \end{aligned} \tag{13}$$

where $A_{(i,j)} \in \mathbb{R}^{n_{(i,j)}\times n_i}$ and $B_{(i,j)} \in \mathbb{R}^{n_{(i,j)}\times n_j}$ map the variables shared between agents i and j into a common space.

Secondly, C-ADMM assumes an algorithmic structure in which, at each iteration, all agents communicate with all neighbors. However, C-ADMM has been proven to converge for significantly less restrictive communication models. Edge-based C-ADMM assumes that at each iteration of the algorithm, only a subset of communication edges are active, and each agent, therefore, communicates with only some of its neighbors [71]. Without loss of generality, this model can be simplified to assume that at each iteration only two agents communicate with each other. Under this edge-based model, when the problem is convex and each edge is active with positive probability, C-ADMM is proven to converge with rate $O(1/k)$ where k is the number of iterations [72].

B. C-ADMM for Batch C-SLAM

Next, we show how the C-SLAM problem (2) can be transformed into an instance of a Separable Consensus ADMM problem with on-manifold decision variables.[4]

Starting from the batch C-SLAM problem (2) we can apply a distributed lens knowing that measurements $\mathcal{M}$ and, in-turn the state variables Θ are distributed across the robot team. First, let $S_{(i,j)} \triangleq \Theta_i \cap \Theta_j$ represent the variables shared between robots i and j, where for $\theta_s \in S_{(i,j)}$ we denote θ_{s_i} as the copy of that variable owned by robot i. From this we can rewrite (2) to reflect this distribution of information as:

$$\begin{aligned} \Theta_{MAP} = \underset{\Theta_i\in\Omega_i \forall i\in\mathcal{R}}{\arg\min} \quad & \sum_{i\in\mathcal{R}}\sum_{m\in\mathcal{M}_i} \left\| h_m(\Theta_i) - m \right\|^2_{\Sigma_m} \\ \text{s.t.} \quad & q_s(\theta_{s_i}, \theta_{s_j}) = 0 \\ & \forall\ \theta_s \in S_{(i,j)} \ \forall\ (i,j)\in\mathcal{E} \end{aligned} \tag{14}$$

where q_s compares the equality appropriately for the manifold to which θ_s belongs and returns $0 \in \mathbb{R}^d$ if and only if θ_{s_i} and θ_{s_j} are equal and a "distance" between the variables if they

[4]This section is based largely on our prior work [66], though it uses additional insights from follow-up work [73] and further experience.

are not equal. The generic function q_s is used as there exist many ways to compare equality of on-manifold objects.

From (14) we can see that the general C-SLAM problem is an instance of a separable optimization problem (13) as each robot's cost function affects only a subset of the global variables and robots share sparse sets of variables. However, unlike (13) and all standard ADMM formulations, C-SLAM problems typically include on-manifold decision variables, which are handled with generic constraint functions q_s.

To permit a fully distributed algorithm, we can take the same approach as standard C-ADMM – to augment the problem with edge variables. Specifically, for every shared variable θ_s shared along edge (i,j), we introduce edge variables $z_{(i,j)_s}$ and $z_{(j,i)_s}$. The addition of edge variables allows us to rewrite the constraints of the problem as:

$$\begin{aligned}\Theta_{MAP} = \underset{\Theta_i \in \Omega_i \forall i \in \mathcal{R},\ Z \in \mathcal{Z}}{\arg\min} \quad & \sum_{i \in \mathcal{R}} \sum_{m \in \mathcal{M}_i} \left\| h_m(\Theta_i) - m \right\|^2_{\Sigma_m} \\ \text{s.t.} \quad & q_s(\theta_{s_i}, z_{(i,j)_s}) = 0 \\ & q_s(\theta_{s_j}, z_{(j,i)_s}) = 0 \\ & \forall\ \theta_s \in S_{(i,j)}\ \forall\ (i,j) \in \mathcal{E}\end{aligned} \tag{15}$$

where Z is the set of all $z_{(i,j)_s}$ and $\mathcal{Z}$ is the appropriate product manifold further constrained such that $z_{(i,j)_s} = z_{(j,i)_s}$. We note here that the representation used for edge variables will depend in part on the representation of its corresponding state variable θ_s and the selection of constraint function q_s. Next we use (8, 9, 10, 11, 12) to derive on-manifold, separable, C-ADMM iterates required to solve (15).

$$\begin{aligned}\Theta_i^{k+1} = & \underset{\Theta_i \in \Omega_i}{\arg\min} \quad \sum_{m \in \mathcal{M}_i} \left\| h_m(\Theta_i^k) - m \right\|^2_{\Sigma_m} \\ & + \sum_{j \in \mathcal{N}_i} \sum_{s \in S_{(i,j)}} \left\langle \lambda^k_{(i,j)_s}, q_s\left(\theta_{s_i}, z^k_{(i,j)_s}\right) \right\rangle \\ & + \sum_{j \in \mathcal{N}_i} \sum_{s \in S_{(i,j)}} \frac{\beta^k}{2} \left\| q_s\left(\theta_{s_i}, z^k_{(i,j)_s}\right) \right\|^2\end{aligned} \tag{16}$$

$$\begin{aligned}z^{k+1}_{(i,j)_s} = & \underset{z_s \in \mathcal{Z}_s}{\arg\min} \\ & \left\langle \lambda^k_{(i,j)_s}, q_s\left(\theta^{k+1}_{s_i}, z_s\right) \right\rangle + \frac{\beta^k}{2} \left\| q_s\left(\theta^{k+1}_{s_i}, z_s\right) \right\|^2 \\ & + \left\langle \lambda^k_{(j,i)_s}, q_s\left(\theta^{k+1}_{s_j}, z_s\right) \right\rangle + \frac{\beta^k}{2} \left\| q_s\left(\theta^{k+1}_{s_j}, z_s\right) \right\|^2\end{aligned} \tag{17}$$

$$\lambda^{k+1}_{(i,j)_s} = \lambda^k_{(i,j)_s} + \beta^k \left(q_s\left(\theta^{k+1}_{s_i}, z^{k+1}_{(i,j)_s}\right) \right) \tag{18}$$

$$\beta^{k+1} = \alpha \beta^k \tag{19}$$

where a dual variable $\lambda_{(i,j)_s}$ is introduced for each constraint and the iterates are written for individual variables where they can be solved for independently. However, to actually apply these iterates to real problems, we need methods to solve the various Augmented Lagrangian optimization problems.

1) State Variable Update: We begin by looking at the state variable update (16). To solve this update, we can leverage existing NLS solvers by transforming the problem slightly, combining the dual and penalty terms of the Augmented Lagrangian into a "Biased Prior" (Sec. IV-B5). This reformulation allows us to compute solutions to (16) using existing sparse NLS solvers like `gtsam`, `g2o`, or `ceres` [74–76].

2) Edge Variable Update: We next look at the edge-variable updates (17). Using a similar reformulation used for the Θ update, we could construct and solve an NLS optimization problem for each $z^{k+1}_{(i,j)_s}$. However, as robots may share large numbers of variables, to improve efficiency, we would prefer that (17) be solved in closed form. As we will later show, depending on the selection of constraint function and how we choose to represent our edge variables (Sec. IV-B6), a closed-form solution may exist or may be approximated.

3) Dual Updates: Computing updates to dual variables is straightforward according to (18) once a constraint function and edge variable representation have been selected.

4) Penalty Updates: For applications that require very precise solutions, the penalty parameter β can be scaled over time using parameter α using a straightforward update (19).

5) Biased Priors: Originally identified by Choudhary et al. [41], the Augmented Lagrangian in (16) can be represented as an NLS problem by combining the dual and penalty terms as "Biased Priors". Specifically, the identity that $\arg\min_a \langle b, a \rangle + (\beta/2)\|a\|^2 = \arg\min_a (\beta/2)\|a + b/\beta\|^2$ when b is constant is used to rewrite the terms as (20).

$$\frac{\beta_{(i,j)}}{2} \left\| q_s\left(\theta_s, z_{(i,j)_s}\right) + \frac{\lambda_{(i,j)_s}}{\beta_{(i,j)}} \right\|^2 \tag{20}$$

Representing consensus constraints, biased priors are effectively the mechanism by which all robots' local solutions are constrained to be equal for any shared variables – thus ensuring a single consistent state estimate for the team.

However, there is a practical challenge in applying biased priors. In most C-SLAM problems we are optimizing over poses that live on the SE(N) manifold. These poses are challenging as the translation (t) and rotation (r) components live in different domains with different scales $t \in [-D, D]$ and $r \in (-\pi, \pi]$ where D is the size of the team's operational area. However, a biased prior, as written in (20), treats these components uniformly. The practical effect is that these biased priors under-constrain a pose's rotational components, which in-turn can cause stability and convergence issues when solving for Θ_i^{k+1}. To improve stability, it is best to account for the different scales of these domains by adding a noise model Σ_s. Specifically, we propose using a noise model Σ_s with $\sigma_r = 0.1$ and $\sigma_t = 1$. We refer to biased priors with $\Sigma_s \neq I$ as "Weighted Biased Priors" for clarity.

6) Constraint Functions: In the standard C-ADMM literature (Sec. IV-A), all algorithms assume vector-valued variables and thus that all constraints are linear ($q(a,b) \triangleq a - b$), which permits a closed-form solution to (17) of $z^{k+1}_{(i,j)_s} = \frac{1}{2}(\theta^{k+1}_{s_i} + \theta^{k+1}_{s_j})$. For vector-valued variables in C-SLAM problems (e.g. landmarks), this approach should be taken.

However, for the SE(N) variables typically found in C-SLAM problems, the equality comparison is less straightforward. In Tab. I we outline four possible constraint functions and for each describe the closed-form solution to (17) or an approximate solution if no closed-form solution exists.

TABLE I: Constraint functions for SE(N) objects and their corresponding closed-form solutions to (17). Where SPLIT interpolates the translation component linearly and the rotation component spherically [77], Vec returns a vector of the objects non-constant matrix elements [78], and p is the dimension of the tangent space for SE(N). Manifold object notation is derived from the work of Solà et al. [79]. The geodesic z update is approximated by the case where $\lambda_{(i,j)_s} = \lambda_{(j,i)_s} = \mathbf{0}$.

Function	$z \in$	$q(\theta, z)$	$z^{k+1}_{(i,j)_s}$
Geodesic	SE(N)	$\text{Log}\left(z^{-1} \circ \theta\right)$	$\text{SPLIT}\left(\theta_{s_i}, \theta_{s_j}, 0.5\right)$
Apx.-Geo.	$\mathbb{R}^p$	$\text{Log}\left(\theta\right) - z$	$\frac{1}{2}\left(\text{Log}\left(\theta_{s_i}\right) + \text{Log}\left(\theta_{s_j}\right)\right)$
Split	SE(N)	$\begin{bmatrix}\text{Log}\left(R_z^{-1} R_\theta\right) \\ t_\theta - t_z\end{bmatrix}$	$\text{SPLIT}\left(\theta_{s_i}, \theta_{s_j}, 0.5\right)$
Chordal	$\mathbb{R}^{N^2+N}$	$\text{Vec}(\theta) - z$	$\frac{1}{2}\left(\text{Vec}\left(\theta_{s_i}\right) + \text{Vec}\left(\theta_{s_j}\right)\right)$

While, on paper, all the formulations in Tab. I appear reasonable, our prior investigations [66] reveal that the geodesic formulation provides the best performance when applied to C-SLAM problems. This finding agrees with general findings from the SLAM community, which has broadly converged to use SE(N) representations and geodesic error for optimization over robot poses.

7) Asynchronous Communication: To compute the edge and dual variable updates, robots must communicate their current state estimates Θ_i for any shared variables. Many batch distributed C-SLAM optimization algorithms require that such communication occurs synchronously, with each robot communicating in lockstep. However, this C-ADMM-based approach to C-SLAM permits an edge-based communication model (Sec. IV-A) in which agents perform asynchronous pairwise communications. To support this model, we do need to add unique penalty terms $\beta_{(i,j)}, \beta_{(j,i)}$ for each pair of robots, as update (19) may be run at different effective rates.

In batch scenarios, the use of an edge-based communication model practically just means less downtime, as robots do not have to wait for rounds of communication to complete and can better utilize communication and computational resources. However, we will see later that the capability to support asynchronous communication is a key advantage of C-ADMM-based approaches when applied in incremental scenarios.

8) MESA+: Putting together the ideas from this section (separable and on-manifold C-ADMM iterates, weighted biased prior reformulation, geodesic constraint selection, and an edge-based asynchronous communication model), we derive the Manifold, Edge-Based, Separable ADMM Plus (MESA+) algorithm (Alg. 1), which optimizes the following iterates:

$$\Theta_i^{k+1} = \underset{\Theta_i \in \Omega_i}{\arg\min} \sum_{m \in \mathcal{M}_i} \left\| h_m\left(\Theta_i^k\right) - m \right\|^2_{\Sigma_m} + \sum_{j \in \mathcal{R}} \sum_{s \in S_{(i,j)}} \frac{\beta^k_{(i,j)}}{2} \left\| \text{Log}\left(\theta^k_{s_i} \ominus z^k_{(i,j)_s}\right) + \frac{\lambda^k_{(i,j)_s}}{\beta^k_{(i,j)}} \right\|^2_{\Sigma_s} \tag{21}$$

$$z^{k+1}_{(i,j)_s} = \text{SPLIT}\left(\theta^{k+1}_{s_i}, \theta^{k+1}_{s_j}, 0.5\right) \tag{22}$$

$$\lambda^{k+1}_{(i,j)_s} = \lambda^k_{(i,j)_s} + \beta^k_{(i,j)} \text{Log}\left(\theta^{k+1}_{s_i} \ominus z^{k+1}_{(i,j)_s}\right) \tag{23}$$

$$\beta^{k+1}_{(i,j)} = \alpha \beta^k_{(i,j)} \tag{24}$$

Remark 2 (MESA vs. MESA+)**:** *We highlight that MESA+ bears differences from our previously proposed MESA [66] (prompting the "+" to differentiate). Namely, MESA+ utilizes weighted biased priors, selects the best-performing constraint function, and does not adopt MESA's alternate dual update ([66, Eq.(18)]) as our experience shows that it can sometimes decrease numerical stability.*

Algorithm 1 Manifold, Edge-based, Separable, ADMM Plus (MESA+)

In: Robots $\mathcal{R}$, communication links $\mathcal{E}$, local estimates $\left\{\Theta_0^0, \Theta_1^0, ..., \Theta_r^0\right\}$
Out: Final Variable Estimates $\left\{\Theta_0^{final}, \Theta_1^{final}, ..., \Theta_r^{final}\right\}$
$\lambda_{(i,j)_s}, \lambda_{(j,i)_s} \leftarrow \mathbf{0}\ \forall\ s \in S_{(i,j)}\ \forall\ (i,j) \in \mathcal{E}$
$z_{(i,j)_s} \leftarrow \theta^0_{s_i}, z_{(j,i)_s} \leftarrow \theta^0_{s_j}\ \forall\ s \in S_{(i,j)}\ \forall\ (i,j) \in \mathcal{E}$
while Not Converged **do**
  if Communication available between robot i and robot j **then**
    In parallel update Θ_i and Θ_j with (21)
    Between i and j communicate $\theta_{s_i}, \theta_{s_j}\ \forall\ \theta_s \in S_{(i,j)}$
    In parallel update $z_{(i,j)_s}, z_{(j,i)_s}\ \forall\ \theta_s \in S_{(i,j)}$ with (22)
    In parallel update $\lambda_{(i,j)_s}, \lambda_{(j,i)_s}\ \forall\ \theta_s \in S_{(i,j)}$ with (23)
    In parallel update $\beta^{k+1}_{(i,j)}$ and $\beta^{k+1}_{(j,i)}$ with (24)

V. Robust Incremental MESA

In this section we propose riMESA, an extension of the MESA+ algorithm developed to solve the robust, incremental problem outlined in Sec. II-E. As discussed in Sec. III, batch C-SLAM back-end algorithms cannot be naively applied to the incremental problem. MESA+, however, provides an algorithmic structure that permits us to design an effective, robust, and incremental version.

A. Incrementalization Overview

An iteration of MESA+ can be broadly broken down into two steps – a local optimization step (21) and a communication step (22, 23, 24). In the local optimization step, robots compute their local estimate, pushing this solution to equal that of the other robots via weighted biased priors. In the communication step, robots share relevant portions of their estimates and tighten shared variable constraints to provide better consistency. It is over many iterations that shared variable constraints are fully realized and a final consistent solution is reached.

A key idea of riMESA is to amortize this process (i.e. the tightening of constraints) over time as communication is available between robots. This amortization is possible for three reasons. Firstly, and most importantly, intermediate results (i.e. from iterations in which shared variable constraints are not tight) are still useful. Solutions to (21) are effectively lower bounded by that found by a robot operating independently and improved with even partially tight constraints. Secondly, C-ADMM-based algorithms demonstrate fast convergence, and therefore consistency and solution quality improve quickly even when the algorithm is amortized over time. Finally, MESA+ is fully agnostic to the optimization process used to solve (21). It only requires that we have a solution when communication occurs. This permits us to use existing incremental optimization methods to achieve online performance.

This idea gives rise to an asynchronous, two-step algorithm consisting of a local update step in which a robot incrementally updates their local solution as new measurements are observed and a communication step in which a pair of robots update their edge and dual variables to tighten shared variable constraints whenever communication is available.

Remark 3 (Batch C-SLAM Amortization)**:** *This structure of amortization is not compatible with all batch C-SLAM algorithms. Algorithms based on distributed gradient descent [9–12] do not provide useful results at intermediate steps of the algorithm. C-ADMM-based algorithms provide a unique basis for an incremental back-end.*

B. Robustness Overview

Incremental optimization methods that solve (21), however, remain sensitive to outliers due to their NLS formulation. Due to the success of continuation methods (Sec. III-C), we next look to address this sensitivity by reformulating the problem through the lens of M-Estimation.

Returning to the distributed C-SLAM optimization problem (14), with the knowledge that some measurements may be incorrect, we can apply robust kernels ρ to our measurements to downweight the effects of outliers:

$$\begin{aligned}\Theta_{MAP} = \underset{\Theta_i \in \Omega_i \forall i \in \mathcal{R}}{\arg\min} \quad & \sum_{i \in \mathcal{R}} \sum_{m \in M_i} \rho\left(\left\| h_m(\Theta_i) - m \right\|^2_{\Sigma_m} \right) \\ \text{s.t.} \quad & q_s(\theta_{s_i}, \theta_{s_j}) = 0 \\ & \forall\ \theta_s \in S_{(i,j)}\ \forall\ (i,j) \in \mathcal{E}\end{aligned} \tag{25}$$

From (25), we can augment the problem with edge-variables and derive the C-ADMM iterates as done in Sec. IV-B. Doing so results in nearly the exact same iterates shown above (21, 22, 23, 24), with the only difference being in the Θ update (21) where $\|h_m\left(\Theta_i^k\right) - m\|^2_{\Sigma_m}$ becomes $\rho(\|h_m\left(\Theta_i^k\right) - m\|^2_{\Sigma_m})$.

Though seemingly a small change to the mathematics, this change highlights an exciting capability of C-ADMM-based optimization for distributed C-SLAM – that outliers need only be handled locally. To handle outlier measurements, one only needs to perform the local optimization robustly. Of course, to additionally support the proposed incrementalization, we also require that we can perform this robust local optimization efficiently. Towards this, we propose using riSAM [60]. As a continuation method, riSAM achieves robustness without inducing a sensitivity to initialization, and as an incremental method, riSAM is able to maintain real-time efficiency.

C. Communication Overview

We next turn our attention to the communication step of MESA+ (22, 23, 24). Due to its basis in C-ADMM, MESA+ permits asynchronous communication. This in-turn allows the proposed algorithm riMESA to natively handle sparse asynchronous communication between agents we expect from our communication networks (Sec. II-D). However, there are additional challenges of real-world communication networks that we must address – latency and communication failures.

Due to network latency, the time to complete a communication between two robots could be longer than the delay between new local measurements with which we need to update our robot's state estimate. Additionally, at any point during this communication, the connection between robots could fail. To ensure that communication/measurement updates do not conflict and that mid-communication failures do not break the algorithm, we adopt a "Communication Handler."

The handler holds a cached copy of the algorithm's state and operates in a separate thread to manage communication between robots. By operating in a separate thread, the main thread can continue to perform updates as new measurements arrive. Additionally, by operating on only a cached copy of the algorithm's state, no changes affect the main thread until the communication is complete and successful. Thus, if there is a failure mid-communication, the handler can simply be discarded, and the only effect on the algorithm is that the communication opportunity is not utilized. Notably, this structure makes it trivial to support multiple parallel communications.

However, despite this design, Two-Generals failures can still cause issues. In the case of such a failure, only one robot will incorporate information from a communication. This will technically violate the invariant stipulated by our C-ADMM design that $z_{(i,j)_s} = z_{(j,i)_s}$. While we cannot prevent these types of failures [18], we design our algorithm to ensure that robots are always able to re-synchronize on their next communication.

D. Additional Details

Stemming from interactions between incremental distributed operation and robust optimization, there are a few additional challenges that we now address.

1) Robust Weighted Biased Priors: By pursuing M-Estimation based approaches, during our local optimization we are simultaneously estimating the variable state as well as attempting to correctly classify the measurement as either an inlier or an outlier. From an information-theoretic perspective, the information that could provide a clear classification for an inter-robot measurement could come from either robot. Importantly, in incremental settings this information can be observed at any time. Consider the case of a misclassified measurement to which one robot later observes clarifying information. Over time, communication between the two robots will allow information to "flow" and allow both to correct the misclassification, but this can take a prohibitively long time due to sparse communication between agents.

To allow robots to more quickly respond to information, we additionally wrap all biased priors in robust kernels. We refer to these as "Robust Weighted Biased Priors" (RWBP), and their use results in the state variable update:

$$\begin{aligned}\Theta_i^{k+1} = \underset{\Theta_i \in \Omega_i}{\arg\min} \quad & \sum_{m \in \mathcal{M}_i} \rho\left(\left\| h_m\left(\Theta_i^k\right) - m \right\|^2_{\Sigma_m} \right) \\ + \sum_{j \in \mathcal{R}} \sum_{s \in S_{(i,j)}} & \rho\left(\frac{\beta^k_{(i,j)}}{2} \left\| \mathrm{Log}\left(\theta^k_{s_i} \ominus z^k_{(i,j)_s} \right) + \frac{\lambda^k_{(i,j)_s}}{\beta^k_{(i,j)}} \right\|^2_{\Sigma_s} \right)\end{aligned} \tag{26}$$

This allows robots to effectively consider the constraints that biased priors represent as "out-of-date" in-turn allowing them to better utilize new information to correct misclassifications and maintain higher quality state estimates.

2) Dual Variable Decay: While the use of robust weighted biased priors solves an important challenge, it also induces another. With robots able to reject constraints, we need to address their corresponding variables $z_{(i,j)_s}$ and $\lambda_{(i,j)_s}$. The edge-variable $z_{(i,j)_s}$ will be updated at the next communication opportunity. Out-of-date dual variables, however, will continue to affect the algorithm as they have been aggregated into the current value $\lambda_{(i,j)_s}$. While dual variables are adaptive in both magnitude and direction as local estimates change, jumps from constraint rejection can leave dual variables with large components in arbitrary dimensions. To tackle this challenge, we introduce a decay rate $\mathfrak{d} = 0.9$, which we use to downweight

old dual estimates so that recent information dominates the dual's influence. This is added to (23), resulting in:

$$\lambda_{(i,j)_s}^{k+1} = \mathfrak{d}\lambda_{(i,j)_s}^{k} + \beta^k \left(q_s \left(\theta_{s_i}^{k+1}, z_{(i,j)_s}^{k+1} \right) \right) \tag{27}$$

3) Penalty Parameters & RWBP Initialization: In the incremental case, we add new measurements and new RWBPs as they are observed, which results in a handful of challenges.

First is that after a new shared variable is observed via an inter-robot measurement and its corresponding RWBP is constructed, there may be a period of time before the pair of robots communicates. During this time, the inter-robot measurement and corresponding RWBP provide no information, as the robot lacks a valid estimate of $z_{(i,j)_s}$. We refer to RWBPs during this period as "uninitialized." To ensure that uninitialized RWBPs do not affect the solution, we construct the penalty parameter with a very small value (β_{uninit}). As other RWBPs may have been previously initialized for the pair, this necessitates that we track a unique $\beta_{(i,j)_s}$ for all shared variables s.

When robots successfully communicate for the first time after a new RWBP is constructed and we compute values for its variables ($z_{(i,j)_s}$, $\lambda_{(i,j)_s}$, $\beta_{(i,j)_s}$), we consider the RWBP to be "initialized." As discussed in Sec. IV-B4, after initialization, updating the penalty parameter can be used to tighten the shared variable constraints. However, this can cause challenges when loop-closures are observed or when constraints are rejected, as discussed above. If constraints are enforced with a large penalty parameter, it can prevent a robot's local optimization process from updating its solution with new information. Therefore, we use only dual variable updates to tighten constraints and use a constant penalty parameter value β_{init} once an RWBP is initialized.

4) Shared Variable Initialization: When RWBPs are initialized, we also initialize the corresponding shared variable θ_{s_i}. In general, C-ADMM-based algorithms are not overly sensitive to initial values and can, without any loss of generality, use an initial estimate provided locally or use the current solution from whichever robot "owns" the shared variable.

However, in the presence of outlier measurements, the choice of initialization injects information into the optimization process. Given that we don't know what information will allow for correct measurement classification, we avoid throwing away information. To do so, we require robots to utilize their local solution along with the inter-robot measurement to initialize any new shared variable. One challenge to this invariant is that some measurements do not observe a variable's entire state. For example, a bearing and range measurement observes position directly but not the orientation of another robot. To account for this challenge, we add a robust initialization scheme for shared variables. Robots will use a local estimate when new shared variables are observed.[5] When communication occurs and we initialize the corresponding biased prior, robots will overwrite their estimate using the owner's value for any component that is not observed locally.

E. riMESA Implementation Details

To realize the ideas proposed above, we must handle the practicalities of implementing the algorithm. Firstly, let each robot i maintain the following internal state:

`risam`	An instance of an riSAM optimizer [60].
Θ	The local state estimate.
$S_j \ \forall \ j \in \mathcal{R}$	Sets variables shared with other robots.
$\mathcal{Y}_j \ \ \forall \ j \in \mathcal{R}$	Variables needing initialization from other robots.
E	The set of observed environment variables.
$\Lambda = \{\lambda_{(i,j)_s}\}$	The set of all dual variables.
$Z = \{z_{(i,j)_s}\}$	The set of all edge variables.
$B = \{\beta_{(i,j)_s}\}$	The set of all penalty parameters.
$\mathcal{C}$	Cache of biased priors to be added the optimizer.
$\mathcal{K}$	Cache of variables affected by new communication.
$\mathcal{W}$	Cache of variables marked for re-convexification.
$\mathcal{Q}$	Mapping from shared variables to the local variables to which they are connected.
$\mathcal{A}$	Mapping from shared variables to their local observability.

Secondly, let each robot i be configured with the following hyperparameter values:

β_{uninit}	$1e^{-4}$	Penalty parameter for uninitialized biased priors.
β_{init}	1.0	Penalty parameter for biased priors.

The exact use of these state variables will be illustrated concretely in the following sections.

1) Bookkeeping: Unlike batch settings, in incremental C-SLAM, the underlying problem is changing at every timestep as robots take new measurements. We must, therefore, efficiently track information required to perform local updates, identify shared variables, and communicate information.

To facilitate communication, each robot i tracks $S_j \ \forall \ j \in \mathcal{R}$, which holds the set of variables locally known to be shared between the pair (i, j), and E, which holds the set of environment variables observed by robot i. When a robot i makes a new inter-robot measurement that observes a shared variable not already in S_j for the respective robot j, it is added to S_j and marked for initialization in $\mathcal{Y}_j$. When a robot observes environmental state (e.g. landmarks), it does not know what other robots with which it shares this state. Therefore, we add environment variables to a separate set E handled specially.[6]

Remark 4 (Shared Variable Identification)**:** *Identification of shared state from indirect measurements will be unambiguous, as the distributed loop-closure system that derived the measurement will know the exact data used to derive the measurement, from which robot that data was received, and the identifier that robot uses for the data. Identification via direct measurements is more challenging, as identification depends on the measurement front-end. However, in general we propose that they can be identified by timestamp, given that robots' clocks are sufficiently synchronized and that the measurement front-end is able to identify which teammate is being observed. Identification of shared environment variables like landmarks is more difficult. Practically, robots will likely need to compare some form of descriptor to identify any shared state. We assume that E holds both variable identifiers and descriptors. We further assume that comparison between such sets (i.e. $E \cup E'$) evaluates descriptor similarity to find shared environment state.*

For each variable s shared with another robot j, each robot i must also maintain a corresponding dual variable $\lambda_{(i,j)_s}$, edge variable $z_{(i,j)_s}$, penalty parameter $\beta_{(i,j)_s}$, and robust weighted biased prior (Sec. V-D1). Thus, when observing a new shared variable, a robot must extend Λ, Z, and B as well as construct a corresponding biased prior that references these new values. Additionally, to support robust optimization, we track in $\mathcal{Q}$

[5]Note that default values (e.g. $\mathbf{0}$) are needed for unobserved components.

[6]Environment Variable – Named to make clear that these variables do not represent the state of any robot but rather represent the state of an element of the local environment.

which local variables are connected to any shared variable s, and to support robust initialization, we track in $\mathcal{A}$ the local observability of all shared variables. All together the bookkeeping process is summarized in Alg. 2.

Algorithm 2 riMESA: Bookkeep (Local to robot i)

In: New environment variables, E_{new}, new shared variables S_{new}, their initial estimates Φ, and a flag L indicating local observation
Extend E for e in E_{new}
for each variable s in S_{new} shared with robot j **do**
  Extend S_j for new variable s
  Extend $\mathcal{Y}_j$ for new variable s if owned by j
  Extend Λ for j, s with value $\mathbf{0}$
  Extend Z for j, s with value Φ_s
  Extend B for j, s with value β_{uninit}
  Extend C with a RWBP on s referencing $\lambda_{(i,j)_s}$, $z_{(i,j)_s}$, and $\beta_{(i,j)_s}$
  Extend $\mathcal{Q}$ for s with its local connections
  Extend $\mathcal{A}$ for s with the variable's observability if L

2) Update: When new factors are observed, we must not only bookkeep the new information but also update our local solution. The process of doing so is simple due to our use of the robust incremental optimizer riSAM [60]. Encapsulating riSAM, riMESA's update is summarized in Alg. 3.

Algorithm 3 riMESA: Update (Local to robot i)

In: New factors $\mathcal{F}$ and new initial estimates Φ
$E_{new} \leftarrow$ any new environment variable observed in $\mathcal{F}$
$S_{new} \leftarrow$ any new shared variable observed in $\mathcal{F}$
Bookkeep(E_{new}, S_{new}, Φ, `true`) ▷ Alg. 2
$\Theta \leftarrow$ `risam.update`($\mathcal{F} \cup C, \Phi$, `reelim`=$\mathcal{K}$, `cvx`=$\mathcal{W}$) ▷ [60, Alg. 4]
Reset $C \leftarrow \varnothing$ and $\mathcal{K} \leftarrow \varnothing$ and $\mathcal{W} \leftarrow \varnothing$

While simple, there are a number of nuances in this procedure. Firstly, by default, riSAM will only re-linearize factors for variables that have changed by more than a set threshold. However, since edge and dual variables are not considered variables by riSAM (as they are not optimized by riSAM), we must specifically indicate that factors affecting shared variables whose edge and dual variables have been modified since the last update (cached in $\mathcal{K}$) must be re-eliminated.

Secondly, riSAM will only treat factors with a robust kernel as potential outliers. We assume that users will mark such factors appropriately but that RWBPs are constructed automatically with robust kernels (Sec. V-D1). Notably, riSAM will only re-convexify such factors when their variables are involved in the update as defined by the topological structure of the underlying factor-graph. In the distributed case, however, a robot does not know the topological structure of the other robots' graphs. Therefore, we take a conservative approach. When a shared variable is involved in an update (defined as when it is initialized), we mark for re-convexification all variables shared with that robot as well as the local variables connected with these shared variables. This set of variables for re-convexification is cached in $\mathcal{W}$.

Finally, we note that the decision to cache $C, \mathcal{K}, \mathcal{W}$ and force their update during Alg. 3 was made to avoid repeated work that would be required by updating riSAM with this information explicitly after communication. This delays the information from appearing in the local solution, but the delay is minor given we expect updates from new measurements to be frequent. If effects from communication are required to be seen immediately, one can call the update step immediately after any communication at a computational cost.

Remark 5 (riSAM Implementation Details)**:** *We use an implementation of riSAM [60] provided by the original authors.*[7] *We use riSAM's DogLeg Line Search to compute updates [60, Alg. 3]. For our M-Estimator, we use the SIG Kernel [60, Eq. 2] with a control parameter update sequence* $\mu = [0.0, 0.5, 0.9, 0.95, 1.0]$ *and a shape parameter computed such that for* $\mu = 1$*, factors with residual* $r \geq \chi^2(0.95)$ *will have an influence* $\left(\partial/\partial r^2\right) \rho_\mu(r^2) \leq 0.1$ *to ensure that factors classified as outliers are unable to negatively affect the solution.*

3) Communication: When communication is available between two robots, C-ADMM only requires that the pair share their current solution for all shared variables. This requires minimal bandwidth on the communication link between the pair. However, the set of shared variables may have changed since the pair's last communication, and each may require initialization for some shared variables, requiring additional (though small) communication overhead. Importantly, we must perform all of this communication in a way that is resilient to latency and communication failures (Sec. V-C).

Therefore, riMESA makes use of a two-stage communication process orchestrated by a communication handler in a separate thread. When communication is initialized between two robots, each constructs a cached version of the algorithm's internal state $\mathfrak{H}$. We denote any component of a cache with a hat ($\hat{}$) to differentiate it from the algorithm state. First, robots update each-other on their set of shared variables, which variables they need to initialize, and their local observability for any such variable. Robots then share their local estimates for the, now jointly known, set of shared variables. This two-stage communication between robots is summarized in Alg. 4.

Algorithm 4 riMESA: Communication Handler (Robots $i \leftrightarrow j$)

In: State cache $\mathfrak{H} = \left\{\hat{\Theta}, \hat{S}_j, \hat{E}, \hat{\mathcal{Y}}_j, \hat{\mathcal{A}}_j \triangleq \mathcal{A}\left[\hat{\mathcal{Y}}_j\right]\right\}$
Out: Communication result $\mathfrak{R}$
Send $\hat{S}_j$, $\hat{E}$, $\hat{\mathcal{Y}}_j$, $\hat{\mathcal{A}}_j$ to j and receive $\hat{S}_i$, $\hat{E}'$, $\hat{\mathcal{Y}}_i$, $\hat{\mathcal{A}}_i$ from j ▷ Stage 1
Send $\hat{\theta}_{s_j}$ and receive $\hat{\theta}_{s_j}$ $\forall\ s \in \hat{S}_j \cup \hat{S}_i \cup (\hat{E} \cap \hat{E}')$ ▷ Stage 2
$\mathfrak{R} \leftarrow$ all data sent and received

Remark 6 (Two-Stage Communication)**:** *A two stage communication process is required for efficient communication by any incremental C-SLAM back-end that shares state data (e.g. DDF-SAM2, DLGBP). In incremental scenarios, shared state may always change between communications. Therefore, before communicating state data, robots must first agree upon a joint set of shared variables. Alternatively, robots could simply send their entire solution, but doing so would induce a much greater communication cost.*

When a communication is successfully completed, a communication result $\mathfrak{R}$ containing all the data transferred in the communication is passed back to the main thread, and the information is incorporated by Alg. 5 to initialize new variables and update S_j, Λ, Z, and B. While this process is still a blocking process with respect to measurement updates (Alg. 3), it is very efficient, only requiring writing of data and computing closed-form solutions to edge variable updates.

Remark 7 (Use of Cached Data in Communication Incorporation)**:** *We specifically call out the fact that in Alg. 5 all new estimates (i.e.* $z_{(i,j)_s}$ *and* $\lambda_{(i,j)_s}$*) are computed using cached data in the communication result* $\mathfrak{R}$*. As the actual communication (Alg. 4) occurs in parallel, additional measurement updates may have changed the algorithm's state by the time a communication is completed. To ensure we compute edge variables that match for both robots involved in the communication, riMESA must use the values that were transmitted even if their local values have since been updated.*

[7] https://github.com/rpl-cmu/risam-v2

Algorithm 5 riMESA: Incorporate Communication (Local to robot i)

In: Comm. Result $\mathfrak{R} = \{\{\hat{\theta}_{s_i}\}, \hat{S}_j, \hat{E}, \hat{\mathcal{Y}}_j, \hat{\mathcal{A}}_j \text{ and } \{\hat{\theta}_{s_j}\}, \hat{S}_i, \hat{E}', \hat{\mathcal{Y}}_i, \hat{\mathcal{A}}_i\}$
$\mathfrak{S} \leftarrow \hat{S}_j \cup \hat{S}_i \cup (\hat{E} \cap \hat{E}')$ ▷ All jointly known shared variables
$S_{new} \leftarrow \mathfrak{S} \setminus \hat{S}_j$
Bookkeep($\varnothing$, S_{new}, Θ, `false`) ▷ Alg. 2
Update θ_{s_j} and $\hat{\theta}_{s_i}$ with Init$\left(\hat{\theta}_{s_i}, \hat{\theta}_{s_j}, \hat{\mathcal{A}}_j[s]\right)$ $\forall\, s \in \hat{\mathcal{Y}}_j$ ▷ i Robust Init.
Update $\hat{\theta}_{s_j}$ with Init$\left(\hat{\theta}_{s_i}, \hat{\theta}_{s_j}, \hat{\mathcal{A}}_i[s]\right)$ $\forall\, s \in \hat{\mathcal{Y}}_i$ ▷ j Robust Init.
$\mathcal{Y}_j \leftarrow \mathcal{Y}_j \setminus \hat{\mathcal{Y}}_i$
for s in $\mathfrak{S}$ **do**
  Update $z_{(i,j)_s} \in Z$ using $\hat{\theta}_{s_i}$, $\hat{\theta}_{s_j}$ and Eq. (22)
  Update $\lambda_{(i,j)_s} \in \Lambda$ using $\hat{\theta}_{s_j}$, $z_{(i,j)_s}$ and Eq. (27)
  If $\beta_{(i,j)_s} == \beta_{uninit}$ **then** update B using $\beta_{(i,j)_s} \leftarrow \beta_{init}$
Extend $\mathcal{K}$ with $\mathfrak{S}$
Extend $\mathcal{W}$ with $\mathfrak{S} \cup \mathcal{Q}\left[\mathfrak{S}\right]$ if $\hat{\mathcal{Y}}_j \neq \varnothing$

4) riMESA Summary: The riMESA algorithm consists entirely of running Alg. 3 when new measurements are added, Alg. 4 when communications are initialized, and Alg. 5 when communications are successfully completed. If communications fail for any reason (e.g. timeout, loss of connection), the thread running Alg. 4 can simply be stopped and the attempt to communicate effectively "discarded." Together these steps allow for riMESA to compute accurate solutions to C-SLAM problems in real-time even when measurements are affected by outliers and communication is ad-hoc, sparse, and unreliable.

Remark 8 (Relationship to Prior Work)**:** *iDFGO [80] is an incremental factor-graph optimizer that can be applied to C-SLAM. iDFGO, however, assumes agents iterate multiple times per-step each requiring synchronized communication on a connected network, violating the communication requirements outlined in Sec. II-D. Additionally, iDFGO focuses on convex problems using chordal constraints for nonlinear problems, degrading performance on C-SLAM compared to geodesic constraints [66]. Finally, iDFGO utilizes Huber kernels for robustness, leaving it sensitive to outliers.*

riMESA is a direct extension of our prior work, iMESA [73]. riMESA surpasses iMESA by handling outlier measurements, supporting non-blocking communications, addressing environment variables, and providing a mechanism for shared variable initialization.

VI. Convergence Guarantees

We next discuss the theoretical convergence provided by C-ADMM-based C-SLAM optimizers like riMESA.

For batch optimization, variants of C-ADMM are proven to converge for non-convex problems [81, 82], on-manifold problems [83], problems with nonlinear constraints [84], asynchronous problems [72], and separable problems [70]. However, to the best of our knowledge, C-ADMM has not been shown to converge for problems like C-SLAM that exhibit all of these traits and utilize coupled nonlinear constraints. These recent works on C-ADMM do provide promising indications that convergence could be proven for algorithms like MESA+ on batch C-SLAM. However, it also indicates that such convergence will be provable only under restrictive conditions to which real-world C-SLAM problems may not adhere.

Moreover, to operate incrementally, riMESA introduces a number of relaxations, namely, amortizing communication, adding RWBPs, and decaying dual variables. These relaxations are necessary for riMESA to achieve quality performance in the incremental setting where the underlying problem is continuously changing, though they diverge from standard C-ADMM that assumes a fixed problem. Taking their limit (i.e. communications at time $t \to \infty$, RWBP $\to$ Weighted Biased Prior, and $\mathfrak{d} \to 1$), we recover a robust version of MESA+. This is aligned with standard C-ADMM but would be unable to operate incrementally, making the relaxations analogous to the wildfire threshold used to incrementalize standard NLS for single-robot SLAM in the iSAM2 algorithm [50]. However, like in iSAM2, these relaxations likely preclude any formal convergence guarantees for the riMESA algorithm even if it were shown for the batch MESA+ algorithm.

While the lack of convergence guarantees is a notable limitation of riMESA, it is also standard among incremental distributed C-SLAM algorithms. DDF-SAM2 does not constrain robots to maintain equal linearization points for shared variables and in turn is guaranteed not to converge to a single consistent solution [6]. Additionally, being an extension of Loopy Belief Propagation, DLGBP is not guaranteed to converge for even some convex problems, let alone non-convex C-SLAM problems [85]. Under these conditions we opt to focus on comparing the empirical performance of incremental distributed C-SLAM algorithms in the following section.

VII. Experiments

In this section we evaluate the performance of riMESA on a variety of real and synthetic C-SLAM problems. We demonstrate that riMESA is able to achieve superior performance compared to state-of-the-art robust, incremental, distributed C-SLAM back-ends and, through ablations, validate its design.

A. Experiment Design

We begin by discussing the design of the experiments – outlining the prior works and baselines to which we compare performance, the metrics for performance evaluation, the method used to generate synthetic datasets, and the model for simulated communications between robots.

1) Prior Works & Baselines: A single prior work, DLGBP [7], specifically targets robust incremental distributed C-SLAM. As discussed in Sec. III, DLGBP is based on Loopy Belief Propagation and takes an M-Estimation approach to outliers, wrapping potential outlier measurements in robust kernels. We use an implementation of DLGBP provided by the authors that uses kernel parameters proposed in the original work and a window size $w = 30$.

To provide a holistic comparison, we additionally compare to a version of DDF-SAM2 [6] that utilizes M-Estimation for robustness. Though not proposed in the original work, the use of M-Estimators is a standard approach to enable robustness, and we argue that their use is a reasonable extension to the method. DDF-SAM2 was implemented by the authors of this work using the Naive-Bayes approximation proposed in the original paper, and potential outliers are wrapped with Geman-McClure kernels that use a shape parameter $c = 3$.

In a similar vein, we also compare to a variant of riMESA that utilizes iSAM2 and M-Estimators rather than riSAM for local optimization. We refer to this variant as "kiMESA" due to its use of "kernels." Comparison to this method is used to validate the use of riSAM for robust local optimization,

making it, effectively, an ablative comparison. kiMESA makes use of Geman McClure kernels with a shape parameter $c = 6$.[8]

In addition to these prior works, we also validate riMESA against a number of baselines. First is "Centralized Oracle" – a centralized algorithm that uses batch Levenberg-Marquardt optimization and oracle outlier information (i.e. incorporates only ground-truth inlier measurements). By utilizing oracle information, this baseline provides an upper-bound on performance given that it is not constrained by limited communication and has perfect measurement classifications.

Second is "iMESA" – a non-robust version of riMESA that likewise uses oracle outlier information [73]. This provides a measure of the performance achievable when a method must tolerate limited communication but is unaffected by outliers.

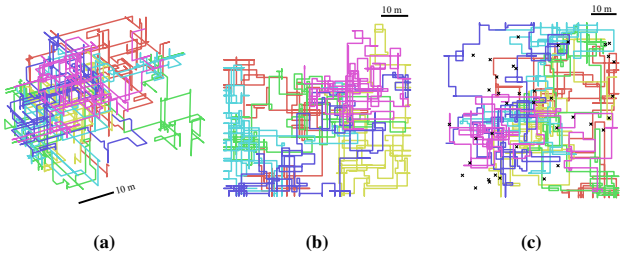
We additionally compare against two centralized baselines that utilize state-of-the-art outlier rejection techniques. "Centralized GNC," which utilizes Graduated Non-Convexity [59], and "Centralized PCM," which utilizes Pairwise Consistency Maximization [43]. These baselines provide a measure of the performance achievable when a method must handle outliers but is not constrained by communication limits. For these methods we use GNC provided by GTSAM,[9] which uses default parameters, and PCM provided by Kimera,[10] which uses a consistency threshold equal to $\chi^2(0.95)$. Due to their computational expense, these baselines are only used for the real-world data experiments, as running them for all synthetic dataset experiments was infeasible.

Finally, we compare against an "Independent" baseline in which all robots use iSAM2 [86] to solve their local factor-graph without inter-robot collaboration. Local outlier measurements are handled by adding a Geman-McClure robust kernel with a fixed shape parameter ($c = 3$). This baseline provides a lower-bound on the performance expected by a collaborative method, given it ignores all inter-robot information.

2) Synthetic Datasets: Synthetic datasets are used to explore algorithm performance across problem conditions.

These datasets are generated for multi-robot teams containing $R = 6$ robots operating in 3D obstacle-free environments. Robot motion is generated by randomly sampling odometry from a categorical distribution with options of $1m$ forward motion as well as $\pm 90°$ rotation around each available axis. We generally focus on "planar" scenarios in which robots are constrained to traverse a planar environment and remain upright. This mobility structure is the most common for robotic applications in which robots traverse a planar (e.g. warehouse) or near-planar (e.g. outdoor) environment. However, this model can also simulate other mobility scenarios.

To construct these datasets, we simulate various types of loop-closure measurements, including intra-robot loop-closures, direct inter-robot loop-closures (i.e. robot-to-robot observations), indirect inter-robot loop-closures (i.e. via a distributed loop-closure system), and landmark observations (i.e. observations of environment features). Each type of measurement is added probabilistically when a robot is within a limited observation range, and outliers are injected such that approximately 10% to 25% of loop-closure measurements are outliers. When generating datasets, each type of measurement can be enabled or disabled, allowing us to simulate the types of measurements generated by different C-SLAM system designs (i.e. sensor selection and front-end algorithm design).



(a) (b) (c)

Fig. 2: Example groundtruth synthetic datasets. (a) An unconstrained 3D dataset. (b) A planar-constrained 3D C-PGO dataset. (c) A planar-constrained 3D landmark C-SLAM dataset. Each color represents a different robot with a trajectory length of 1000 poses.

All measurements are constructed with a fixed Gaussian noise model, which defines both how noise is added to simulated measurement and how algorithms model noise in their optimization. The noise models for all measurements investigated contain a rotational component, a translational component, or both. We define the noise models using a standard deviation σ_r for any rotational components and σ_t for any translational components. For planar-constrained scenarios, we separately specify a value for the rotational yaw axis σ_{rz} since, in many applications, this uncertainty is larger as observations of gravity or the ground constrain pitch and roll. In the experiments below, we often explore various noise models, as large noise results in problems that are generally challenging to solve, not only because of the noise but also because large noise results in odometry drift and, in-turn poor initial estimates. Small noise, on the other hand, constructs problems that are easier to solve and have better initialization.

Examples of the synthetic datasets that can be generated by this method can be seen in Fig. 2.

3) Communication: To model the communication conditions we expect in real-world deployments, we adopt the following general communication model for our experiments. We simulate communication between robots by initiating new communications at a fixed rate r_c between any available robots within a distance of d_c of each other. We assume that these communications are completed within a fixed period of time (resulting in a constant delay b_c) and are successful with a probability p_c. Finally, we randomly sample 5% of otherwise successful communications and fail to incorporate the results for only one of the robots to simulate Two-Generals failures.

One challenge of this model is that synthetic datasets are divorced from any understanding of time. This makes it difficult to define communication parameters, which are often measured in seconds and Hz. For experiments utilizing synthetic datasets, we adopt an abstract unit based on the dataset's odometry measurements. Specifically, we define the period between odometry measurement updates as $\mathfrak{s}$ and use this to also define a rate unit $\mathfrak{r} = 1/\mathfrak{s}$, which we use to define communication model parameters.

Remark 9 (Synthetic Dataset Timing)**:** *Using real-world data, we can use the definitions of $\mathfrak{s}$ to derive approximate timing values*

[8]We also evaluated kiMESA using the same shape parameter as riMESA (Remark 5). However, it was consistently outperformed by the fixed value, and we opt to omit its results for improved readability.

[9]https://github.com/borglab/gtsam

[10]https://github.com/MIT-SPARK/Kimera-RPGO

to ground our synthetic datasets. An analysis of the COSMO-Bench Datasets [87] shows that they generate keyframes about every $1.5s$*. Applying this to our synthetic datasets, defines* $\mathfrak{r} \approx 0.67Hz$ *and a 1000 pose-long trajectory as* ≈ 25 *minutes of robot operation.*

4) Metrics: We evaluate all algorithms on their capacity to produce state estimates that match the true state of the world, as this is most impactful on downstream tasks like planning and navigation. Therefore, we evaluate the performance of all methods using Average Trajectory Error (ATE) to inspect the practical quality of results [88]. We additionally evaluate the ability of an algorithm to properly classify measurements as inliers or outliers. To do so, we use the F1 score (a summary of precision and recall) computed treating "inlier" as the positive class. We compute ATE jointly across all robots after Umeyama alignment to the reference solution and compute the F1 score across the classifications of all robots in the team.

However, we look to evaluate these metrics not only for the final timestep, but for all intermediate timesteps during which a real multi-robot team will need accurate state estimates. To measure this performance, we use an incremental variant of these metrics [60]. For a metric "METRIC" this is defined as:

$$\text{i}\{\text{METRIC}\} = \sum_{k \in K} \left[\frac{k}{\sum_{k \in K} k} \{\text{METRIC}\} \left(\Theta^k, \Theta^*\right) \right] \quad (28)$$

where Θ^k is the solution produced at iteration k and Θ^* is the reference solution. As a weighted average over all timesteps, incremental metrics reward methods that maintain consistently low errors. Compared to point metrics (e.g. metric value at the final timestep), incremental metrics provide a better measure of the performance that can be expected by downstream algorithms throughout robot operation.

For experiments with multiple trials, the incremental metrics are summarized into box-and whisker plots. For experiments with a single trial, the incremental metric is reported directly. For better readability, we report only the translation component of iATE, as the rotation and translation components are highly correlated. Finally, there are cases in the experiments where algorithms fail.[11] If trials fail, the summary box-and-whisker plot is rendered with dashed lines, and the percentage of successful trials summarized in the plot is written above.

B. C-SLAM Generalization Experiment

In our first experiment, we evaluate the ability of riMESA to generalize to different problem scenarios. We design each scenario by enabling and disabling different types of measurements (Sec. VII-A2). This is used to simulate the measurements produced by various C-SLAM system designs.

For each of these scenarios, we additionally evaluate across different levels of measurement noise to simulate different difficulties of problems. For each noise level and scenario combination, we generate 50 random datasets that simulate a team of 6 robots traversing trajectories 1000 poses long.

For all scenarios, robots can initiate new communications at a rate of $r_c = 1\mathfrak{r}$ with any robot within a communication range of $d_c = 30m$. We assume that communications are successful with $p_c = 0.9$ and are completed before the next measurement update is received (effective delay $b_c < 1\mathfrak{s}$).

[11] Failures include when an algorithm diverges to an unreasonable solution or when the algorithm crashes from numerical instability issues.

1) Collaborate PGO (C-PGO) (Fig.3a): We first look at a PGO scenario in which robots generate relative-pose intra-robot loop-closures as well as relative-pose indirect inter-robot loop-closures. C-PGO systems are currently particularly popular as they are mathematically well constrained and the geometric sensors used to derive relative-pose measurements (i.e. LiDARs) are readily available and highly accurate.

2) Range-Aided PGO (Fig.3b): We next look at a scenario in which agents locally perform PGO and derive only direct inter-robot ranging measurements (e.g. from Ultra-Wide-Band (UWB) radios) to support collaboration.

3) Range-Only C-SLAM (Fig.3c): Next we look at a scenario like that above but where, due to sensor or computational limitations, robots utilize only their local odometry and direct inter-robot ranging measurements to localize the team.

4) Bearing-Range-Only C-SLAM (Fig.3d): Next we look at a scenario like that above but in which robots can also observe the bearing to the target teammate when ranging.

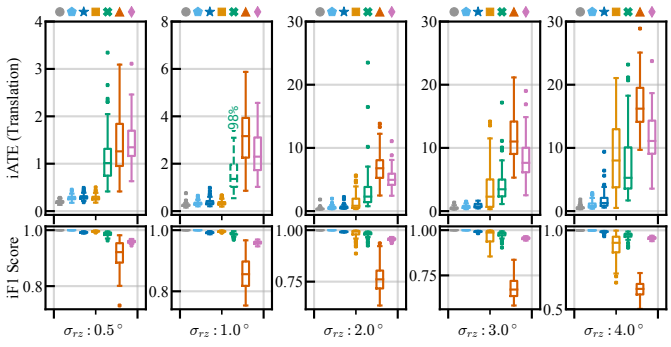
5) Landmark C-SLAM (Fig.3e): Next we look at a scenario in which robots utilize only odometry and measurements to landmarks in their environment. Such system design is commonly used when visual sensors are the primary sensor onboard each robot. Note that for this scenario, the prior work DLGBP is omitted as its distributed formulation is unable to support passive landmarks and instead requires landmarks to support active computation and communication.

6) Landmark+Direct C-SLAM (Fig.3f): Next, we look at a landmark C-SLAM scenario in which robots additionally make direct inter-robot bearing-range measurements to teammates. Again, DLGBP is omitted as it requires active landmarks.

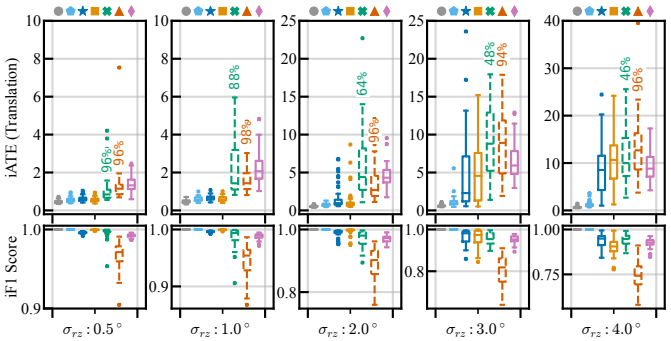
7) Unconstrained 3D C-PGO (Fig. 4.): Finally, we look at a scenario in which robots are not constrained to a planar environment but rather can move arbitrarily in 3D. Such scenarios are extremely rare in robotics (i.e. zero-gravity) but are included for theoretical completeness.

8) Generalization Experiment Analysis: In this experiment we can see that riMESA significantly outperforms prior works, producing solutions that are closest to the baselines that utilize oracle outlier information. This provides us confidence that riMESA can generalize across C-SLAM problem scenarios to adapt to the sensors and front-end algorithms that are necessary for the application. We can additionally see some scenarios where riMESA's performance degrades. Notably, riMESA struggles on the Range-Aided PGO (3b), Range-Only C-SLAM (3c), and Bearing-Range-Only C-SLAM (3d) scenarios in cases of large measurement noise. The commonality among these scenarios is that they are scenarios where the inter-robot measurements are low-rank. Therefore, in cases of large noise, these are scenarios where the effective signal-to-noise ratio is low. Interestingly, in these cases the iMESA baseline also struggles while the Centralized Oracle performs well. This indicates that it is the limited communication rather than the noisy measurements or outliers that is causing the poor performance. We, therefore, hypothesize that with more communication, riMESA can succeed in these conditions. This hypothesis is explored below in Sec. VII-D.

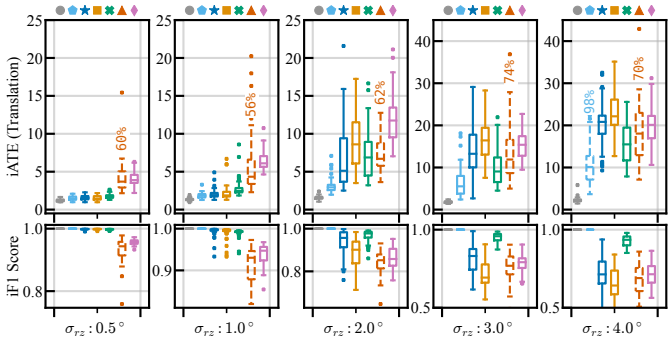
In this experiment we can also see that DDF-SAM2 and DLGBP struggle across most problem conditions. Both prior



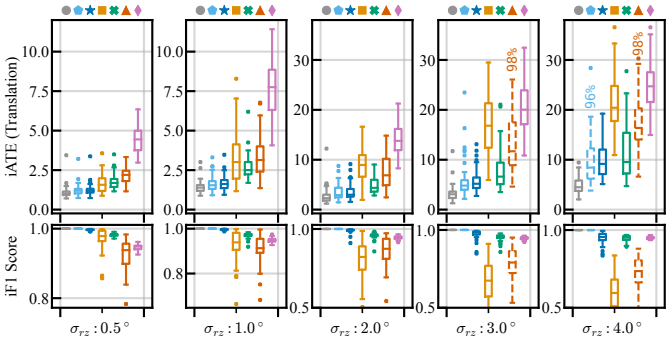
(a) Collaborate Pose-Graph Optimization (C-PGO)



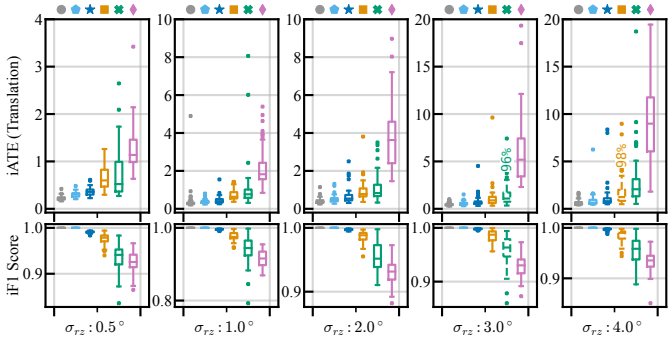
(b) Range-Aided PGO



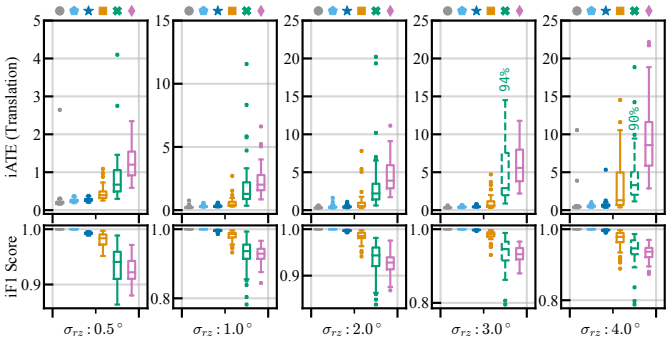
(c) Range-Only C-SLAM



(d) Bearing-Range-Only C-SLAM



(e) Landmark C-SLAM



(f) Landmark+Direct C-SLAM

Fig. 3: Metric performance for Centralized Oracle (●), iMESA (⬟), riMESA (★), kiMESA (■), DDF-SAM2 (✖), DLGBP (▲), and Independent (♦) on various planar-constrained datasets for different levels of measurement noise. Only the yaw axis noise (σ_{rz}) is changed, and datasets use fixed $\sigma_r = 0.25°$ and $\sigma_t = 0.05m$. Across all problem scenarios, our proposed method, riMESA, outperforms prior works and achieves the closest solution to that of the baselines that exploit oracle outlier information. However, there are some problem conditions for which riMESA struggles at high levels of measurement noise. Note: DLGBP is omitted from landmark scenarios as it does not support passive landmarks.

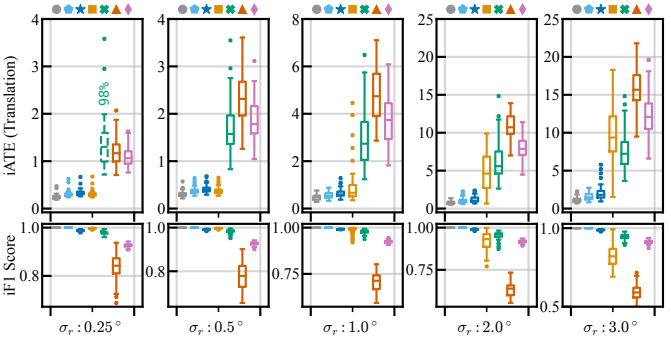
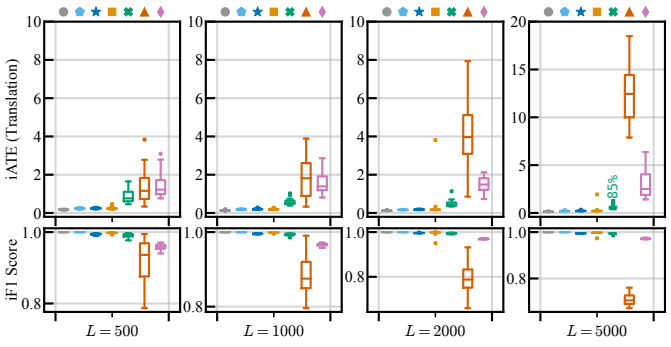


Fig. 4: Metric performance for Centralized Oracle (●), iMESA (⬟), riMESA (★), kiMESA (■), DDF-SAM2 (✖), DLGBP (▲), and Independent (♦) on a unconstrained 3D C-PGO datasets for different levels of measurement noise. Only σ_r is changed and datasets use fixed $\sigma_t = 0.05m$.

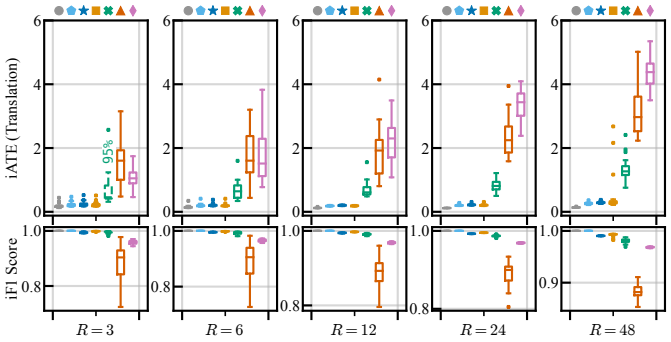
works can sometimes produce quality results but are just as likely to compute poor estimates or diverge entirely. kiMESA, our ablative method, often outperforms these prior works. However, as expected from findings in robust optimization literature, the use of a Geman McClure robust kernel makes it sensitive to initialization, and kiMESA struggles to converge at large noise levels even in high signal-to-noise scenarios. It is also worth noting that in these experiments there are some instances of iMESA diverging. This is likely due to its use of Gauss-Newton optimization steps. This differs from riMESA, which uses a trust-region optimizer. The use of trust-region optimization steps in iMESA would likely resolve this issue.

C. Scale Experiment

In our next experiment, we evaluate how riMESA scales with long-term operation and large teams of robots. In both cases we evaluate on 20 planar-constrained 3D C-PGO datasets. For the long-term operation experiment, we simulate a team with 6 robots over various trajectory lengths (L). For the team-size experiment, we simulate different robot team sizes (R), where each robot traverses a trajectory of 1000 poses. For both, we use the same communication model as in the Generalization Experiment (Sec.VII-B) and a noise model with $\sigma_{rz} = 1.0°$, $\sigma_r = 0.25°$, and $\sigma_t = 0.05m$. Results from the long-term operation experiment can be found in Fig. 5a, and



(a) Long Term Operation



(b) Team-Size

Fig. 5: Metric performance for Centralized Oracle (●), iMESA (⬟), riMESA (★), kiMESA (■), DDF-SAM2 (✖), DLGBP (▲), and Independent (♦) across different (a) operation lengths L and (b) different team-sizes R. riMESA provides quality performance across all problem scales.

results from the team-size experiment can be seen in Fig. 5b.

In this experiment we can see that riMESA is able to scale to both large teams and long-term operation, providing consistent, high-quality results across scales. Both prior works, DDF-SAM2 and DLGBP, see degraded performance at large scales. However, this is most pronounced for DLGBP, which performs particularly poorly during long-term operation. This is likely caused as DLGBP struggles to incorporate long-term loop-closure information due to windowing, which is necessary to maintain online efficiency.

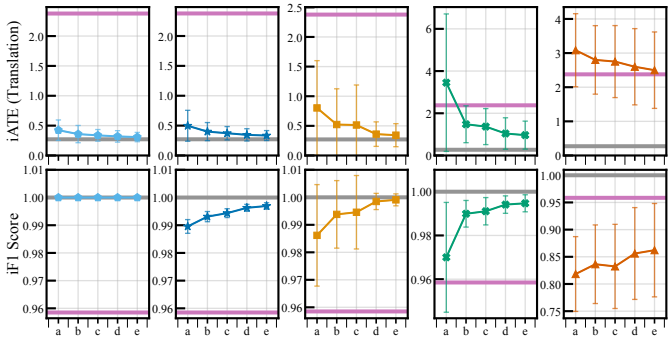
D. Communication Experiments

In our next experiment, we seek to evaluate how the quality of communication affects riMESA. We define "poor" communication quality as when robots have a limited communication range r_c and can only communicate with long delays b_c due to latency and limited bandwidth. Likewise, we define "good" communication quality as when robots can communicate over long ranges at high effective rates.

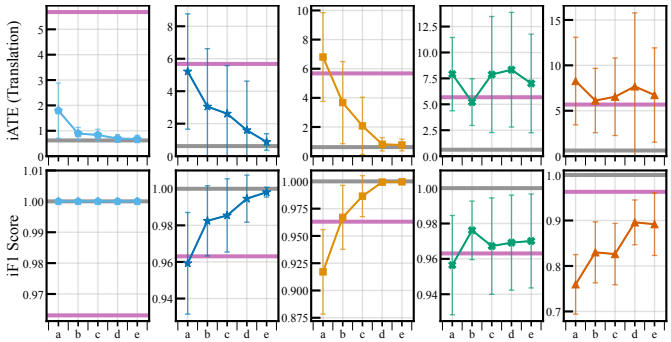
For "poor" communication scenarios, we explicitly simulate delays to test algorithms' abilities to tolerate receiving stale data. In these scenarios robots still attempt to initiate new communications at a faster rate, and we permit robots to maintain communications with multiple robots at once.[12]

We look at the effect of communication under three different scenarios: a standard C-PGO scenario, a Range-Aided PGO scenario, and a Range-Only C-SLAM scenario. C-PGO was selected as a representative generic scenario. The others

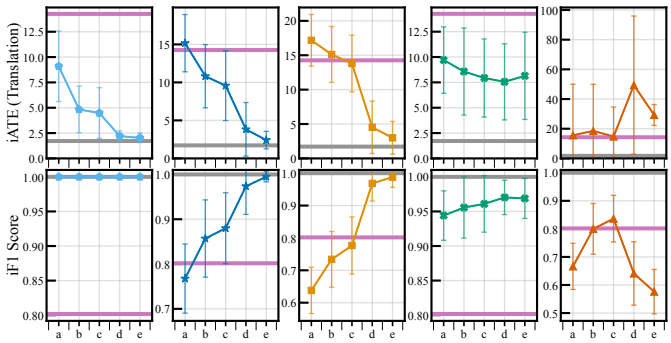
[12]The implementation of DLGBP provided by the original authors does not permit us to simulate delays. Thus, while we affect the rate of communication for DLGBP, we are unable to simulate incorporation of stale data.



(a) Collaborate Pose-Graph Optimization ($\sigma_r = 0.25°$, $\sigma_{rz} = 1°$, $\sigma_t = 0.05m$)



(b) Range-Aided PGO ($\sigma_r = 0.25°$, $\sigma_{rz} = 2.5°$, $\sigma_t = 0.05m$)



(c) Range-Only C-SLAM ($\sigma_r = 0.25°$, $\sigma_{rz} = 2.5°$, $\sigma_t = 0.05m$)

Fig. 6: Metric performance for iMESA (⬟), riMESA (★), kiMESA (■), DDF-SAM2 (✖), and DLGBP (▲) on various planar-constrained problem scenarios for different communication conditions. The baselines Centralized Oracle (●) and Independent (♦) are not dependent on communication and are plotted as horizontal lines. Communication quality, defined by [delay (b_c), rate (r_c), max-distance (d_c)], improves from a to e, where a=[10𝔰, 1𝔯, 30m], b=[2𝔰, 1𝔯, 35m], c=[0𝔰, 1𝔯, 40m], d=[0𝔰, 5𝔯, 45m], e=[0𝔰, 10𝔯, 50m]. Higher quality communication enables riMESA to provide better results across all scenarios, but particularly for low signal-to-noise problems like (c) Range-Only SLAM.

were selected due to our observations in the Generalization Experiment (Sec. VII-B) that – at larger noise levels, riMESA's performance on these scenarios significantly degrades. We explore these scenarios to investigate if higher quality communication can enable quality performance in these challenging scenarios. For these scenarios, we specifically select noise levels at which riMESA's performance begins to degrade. For each combination of communication quality and problem scenario, we evaluate 20 random synthetic datasets. Results from this experiment can be seen in Fig. 6.

In this experiment we can see that, for high signal-to-noise problems (Fig. 6a), riMESA is robust to communication conditions, producing quality results even with very limited communication. We can further see that for low signal-to-noise problem scenarios (Fig. 6b and Fig. 6c), improvements

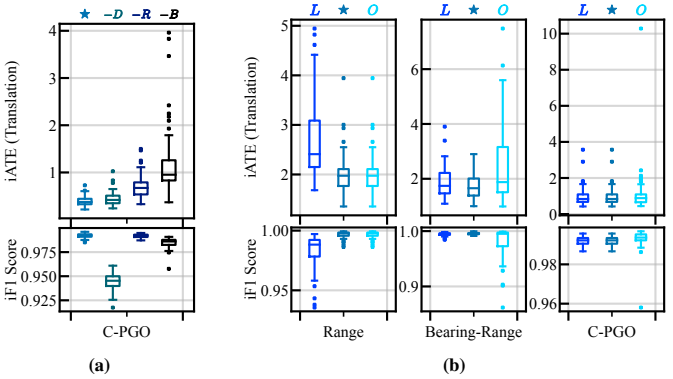


Fig. 7: riMESA ablation studies. In (a) we compare riMESA (★) to a version without dual-decay (***–D***), a version without robust kernels on weighted biased priors (***–R***), and a version without both of these components (***–B***). In (b) we compare riMESA (★) to a version that uses only local estimates to initialize shared variables (***L***) and a version that uses only estimates from owner robots to initialize shared variables (***O***). Both ablation studies validate that the components improve the performance of riMESA.

to communication quality significantly improve results. This supports the hypothesis introduced in Sec. VII-B8 – that riMESA can perform well under low signal-to-noise conditions but requires more communication to achieve quality results.

We can also see that iMESA performs better with higher quality communication, though, compared to riMESA, the improvements are far less drastic. This indicates that while our C-ADMM-based design allows outliers to be handled locally from a mathematical perspective, practically the handling of outliers still imposes some communication burden onto the back-end. Retrospectively, this makes intuitive sense. The robust problem is significantly more complex, and it is reasonable that a multi-robot team will require more communication to converge to a quality solution on a more complex problem.

Additionally, this experiment shows that while iMESA, riMESA, and kiMESA all improve their performance with higher quality communication, the prior works DLGBP and DDF-SAM2 do not see the same improvements. For some problem scenarios, more communication does little to affect the results of these prior works at all. Finally, it is worth noting that while higher quality communication allows kiMESA to improve performance, we still expect this method to struggle in scenarios with large noise, as seen in Fig. 3.

E. Ablation Experiments

We next perform some ablative tests to validate our design decisions for riMESA's robust optimization technique (Sec. V-B) and robust initialization scheme (Sec. V-D4).

1) Robust Optimization Ablation: In this experiment we look at the removal of two components of our robust optimization scheme. Namely, we individually remove the wrapping of weighted biased priors in robust kernels ("–R"), remove use of a decay rate for dual variables ("–D"), and remove both ('–B"). We compare these variants against riMESA on 50 planar 3D C-PGO problems and use the same communication model as the Generalization Experiment (Sec.VII-B) and a noise model with $\sigma_{rz} = 1.0°$, $\sigma_r = 0.25°$, and $\sigma_t = 0.05m$. The results from this experiment can be seen in Fig. 7a and show that the use of dual-decay and robust weighted biased priors significantly improves the performance of riMESA.

2) Robust Initialization Ablation: In this experiment we analyze the effect of our robust initialization scheme. We compare the robust initialization scheme against a variant that uses only local information and default values to initialize shared variables ("L") and a variant that uses the estimate of the owner robot to initialize shared variables ("O"). We compare these variants on three scenarios with different local observability – a Range-Only C-SLAM scenario in which robots do not directly observe any shared variable state locally, a Bearing-Range-Only C-SLAM scenario in which robots locally observe shared variable locations, and a C-PGO scenario in which robots locally observe the entire shared variable state.

For each scenario, we evaluate 50 planar-constrained problems and use the same communication model as the Generalization Experiment (Sec.VII-B). For all scenarios, we use noise model parameters $\sigma_r = 0.25°$ and $\sigma_t = 0.05m$. For the C-PGO scenario, we define $\sigma_{rz} = 3.0°$, and for the range and bearing-range scenarios, we define $\sigma_{rz} = 1.0°$.

The results from this ablation study can be seen in Fig. 7b and show that the robust initialization scheme always results in the best overall performance. It is worth explicitly noting that the robust initialization scheme is equivalent to other approaches in some scenarios. For example, in the range-only scenarios, the robust initialization approach is equivalent to using the owner robot's estimate for initialization.

F. Real-World Experiments

In our final experiment, we seek to evaluate the performance of riMESA on data that (as closely as possible) resembles real-world data. To achieve this goal while supporting reproduction of our results, we adopt COSMO-Bench [87]. This benchmark consists of 24 datasets generated from real-world LiDAR data [89, 90], a baseline LiDAR-based C-SLAM front-end, and a communication model derived from real-world data. It also provides four additional datasets from the Nebula dataset [26].

For this experiment, we adopt communication models based on those proposed by COSMO-Bench. Specifically, we assume that robots initiate new communications at a rate of $r_c = 5Hz$, that the time required to complete each communication is $b_c \leq 200ms$, and that the network bandwidth is always sufficient to transmit the data required by the algorithms. We model communication success p_c and max communication range d_c using the connectivity model (i.e. Wi-Fi or Pro-Radio) applicable to the dataset [87, Sec.IV.C].[13] We additionally continue to sample a small fraction (5%) of otherwise successful communications to simulate Two-Generals failures by failing to incorporate the results for one of the robots.

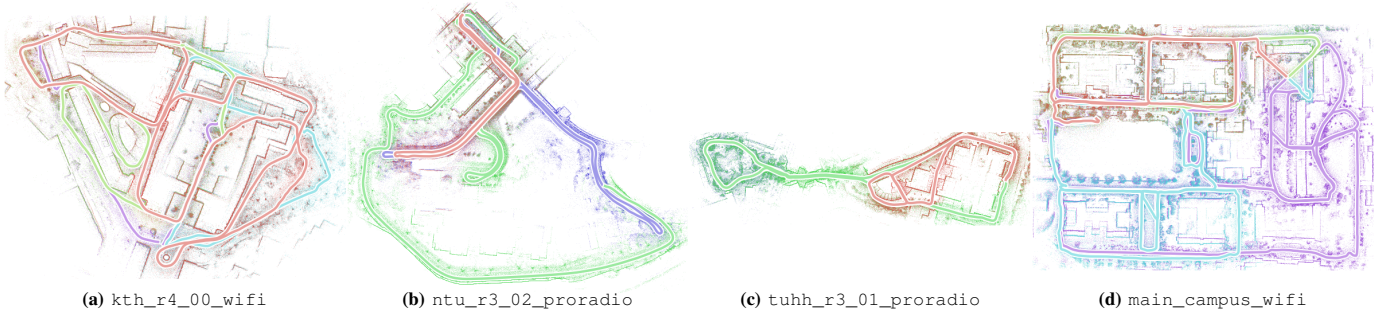
1) Real-World Accuracy Results: Comprehensive results for the COSMO-Bench datasets can be found in Tab. II, and results from the Nebula datasets can be found in Tab. III. For these results, we also provide the ATE for the final step of the dataset for both completeness and to provide better context to iATE values. Qualitative performance of riMESA can be seen in Fig. 8 for a sample of the datasets, and an example plot of ATE over a single dataset can be seen in Fig. 9.

Remark 10 (Real-World Experiment Nuances)**:** *While analyzing the real-world results keep in mind that our reference solutions, which are used to compute metrics, are not perfect. While derived using accurate additional sensors, these reference solutions do not*

[13]For the Nebula datasets, we use the Pro-Radio connectivity model.

TABLE II: iATE and Final ATE performance for all methods and baselines on the COSMO-Bench Datasets [87]. The best-performing method (excluding baselines) for each dataset is bolded. The proposed algorithm, riMESA (highlighted in gray), performs consistently across all datasets.

		Metric	iATE (m)												Final ATE (m)											
		Trial	kth r3_00	kth r3_01	kth r4_00	ntu r3_00	ntu r3_01	ntu r3_02	ntu r4_00	ntu r5_00	tuhh r3_00	tuhh r3_01	kittredge loop	main campus	kth r3_00	kth r3_01	kth r4_00	ntu r3_00	ntu r3_01	ntu r3_02	ntu r4_00	ntu r5_00	tuhh r3_00	tuhh r3_01	kittredge loop	main campus
Wi-Fi Datasets	Baselines	Independent	8.24	11.5	8.76	6.99	7.51	5.93	7.94	7.50	17.1	15.4	33.2	18.2	9.52	13.1	8.67	7.05	8.45	7.98	8.25	7.73	19.0	19.4	47.4	16.9
		Cent. Oracle	2.83	4.51	2.88	2.28	2.01	4.44	1.79	1.94	1.96	4.42	5.47	8.53	1.91	1.52	1.19	1.65	2.08	1.81	1.37	1.39	1.10	1.15	4.58	3.83
		iMESA	5.51	5.10	4.72	2.72	3.34	5.24	2.58	2.52	6.31	3.99	5.93	9.36	2.54	2.96	4.12	1.62	3.20	2.46	1.63	1.53	2.80	1.04	4.90	3.95
		Cent. GNC	5.42	5.25	3.23	2.53	2.36	4.53	2.10	2.32	1.86	5.19	7.41	9.98	2.16	1.51	1.30	2.06	2.51	3.67	1.85	1.80	0.85	1.23	5.06	3.83
		Cent. PCM	28.7	43.3	80.1	2.56	2.27	4.49	2.15	2.19	3.14	4.74	25.9	13.1	79.7	80.1	77.9	1.84	2.39	1.81	1.70	1.52	1.43	1.46	87.1	5.09
	Methods	DDF-SAM2	11.4	53.3	26.1	**3.27**	23.8	4.81	**2.60**	21.9	98.9	4.81	**10.8**	12.8	9.27	86.4	55.7	**1.90**	22.7	**2.26**	**1.58**	25.2	33.7	5.37	**5.38**	**3.74**
		DLGBP	6.80	13.0	11.9	11.6	8.27	4.88	10.9	10.9	12.2	17.8	71.9	58.1	7.46	15.1	13.0	13.5	12.0	6.41	13.3	13.0	21.2	28.1	99.3	76.4
		kiMESA	6.60	11.8	9.65	5.14	4.56	5.56	3.10	5.63	10.5	29.7	34.9	15.3	6.36	12.0	9.09	2.10	4.48	7.07	2.19	5.30	7.21	62.2	8.06	3.77
		riMESA	**4.46**	**4.56**	**4.30**	3.84	**3.27**	**4.50**	4.72	**3.92**	**2.66**	**3.91**	16.8	**10.3**	**2.58**	**2.28**	**3.56**	2.95	**3.04**	3.22	2.96	**2.95**	**1.59**	**1.27**	7.34	4.29
Pro-Radio Datasets	Baselines	Independent	7.33	12.5	11.2	7.14	7.22	5.76	7.69	7.81	16.5	10.9	29.9	35.5	8.19	14.3	12.0	7.45	7.82	7.99	7.42	8.09	18.7	15.7	36.8	23.9
		Cent. Oracle	2.39	3.03	2.58	1.59	1.53	1.97	1.92	1.86	1.71	1.37	4.72	4.40	1.13	1.42	1.25	1.64	1.42	1.78	1.46	1.50	1.24	1.26	4.28	3.20
		iMESA	6.68	3.81	4.01	1.63	1.66	2.02	2.51	2.20	5.36	2.17	5.01	5.03	3.61	3.07	1.90	1.47	1.50	1.89	1.75	1.68	3.49	1.07	4.56	3.47
		Cent. GNC	3.73	2.87	2.77	2.04	2.07	2.50	2.28	1.99	1.59	1.67	5.56	4.69	1.25	1.53	1.33	2.16	2.14	2.97	2.10	1.81	1.27	1.32	4.67	3.25
		Cent. PCM	72.5	57.4	72.0	2.05	14.7	1.98	2.17	2.12	79.8	3.86	10.9	6.74	60.4	60.2	47.0	1.77	5.39	1.75	1.68	1.63	105.7	1.38	6.05	5.26
	Methods	DDF-SAM2	45.2	98.8	40.4	5.38	1.77	**1.89**	5.73	3.48	53.8	36.5	32.6	12.8	54.4	101.5	42.3	**1.67**	**1.64**	**1.65**	**1.69**	1.83	30.1	22.5	13.4	**3.50**
		DLGBP	6.32	13.0	11.6	11.3	8.40	4.78	12.0	11.3	8.06	12.5	72.4	57.7	6.88	15.1	12.8	13.4	12.5	6.27	14.7	13.5	17.0	25.5	100.3	75.9
		kiMESA	10.7	**3.55**	12.1	**1.89**	**1.75**	2.59	5.19	**2.29**	7.62	5.29	32.8	34.3	11.5	**2.58**	12.3	1.81	1.66	2.76	4.95	**1.78**	3.96	3.72	31.2	22.9
		riMESA	**4.97**	3.67	**4.33**	3.14	4.85	2.99	**4.43**	3.25	**5.36**	**2.20**	**6.96**	**8.15**	**3.51**	3.17	**2.58**	2.96	3.44	3.69	4.09	3.16	**3.58**	**1.34**	**5.77**	3.84



(a) kth_r4_00_wifi **(b)** ntu_r3_02_proradio **(c)** tuhh_r3_01_proradio **(d)** main_campus_wifi

Fig. 8: Qualitative results from riMESA on a sample of COSMO-Bench datasets. Colors represent different robots. The pointcloud map is constructed by transforming LiDAR scans according to the riMESA solution. Visible by the alignment between each robot's map, riMESA is able to provide accurate and consistent state estimates.

TABLE III: iATE and Final ATE performance for all methods and baselines on the Nebula Datasets [26] released by COSMO-Bench [87]. The best-performing method (excluding baselines) for each dataset is bolded. The proposed algorithm is able to closely match the performance of the Centralized GNC baseline (expected performance upper-bound), though all methods do generally well on these datasets.

	Metric	iATE (m)				Final ATE (m)			
	Dataset	finals	ku	tunnel	urban	finals	ku	tunnel	urban
Baselines	Independent	0.61	3.14	3.36	1.03	0.65	1.95	3.96	1.47
	Cent. Oracle	0.55	2.47	1.49	0.70	0.74	2.08	1.54	0.84
	iMESA	0.49	2.26	0.88	0.64	0.78	1.57	0.93	0.76
	Cent. GNC	1.65	1.48	0.91	1.17	0.52	1.26	1.05	1.11
	Cent. PCM	5.07	1.51	2.49	3.13	6.77	1.91	2.48	5.46
Methods	DDF-SAM2	6.74	1.43	2.78	7.03	17.1	1.30	2.88	12.0
	DLGBP	**0.69**	2.90	**0.79**	1.00	0.93	3.64	0.89	1.13
	kiMESA	0.73	1.44	0.83	**0.94**	**0.49**	1.26	0.89	**1.04**
	riMESA	1.71	**1.37**	0.85	1.06	0.66	**1.20**	**0.87**	1.11

necessarily represent true "groundtruth." Secondly, due to real-world measurement noise, the global optimum of the C-SLAM problem does not necessarily correspond to either the groundtruth nor the reference solution. Combined, these realities mean that even our theoretical upper-bound methods (i.e. Centralized Oracle) may report metrics that appear worse than other methods. Despite this, these metrics still represent the practical usefulness of a solution; the trends, however, are noisier than those seen in synthetic data experiments.

In these results we can see that riMESA is most frequently the best-performing method. There are some datasets for which riMESA is beaten by the ablative method kiMESA (e.g. ntu_r3_01_proradio) or the prior works DDF-SAM2 or DLGBP (e.g. ntu_r4_00_wifi and finals). However, unlike these methods, which also frequently produce poor results (e.g. ntu_r4_00_proradio), riMESA is the only method that is able to perform consistently across all the datasets. Additionally, due to our use of incremental metrics, and as highlighted in Fig. 9, we can see that riMESA performs consistently throughout robot operations.

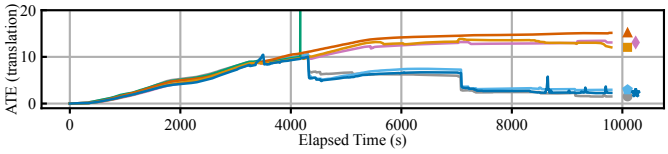


Fig. 9: ATE plotted over time for kth_r3_01_wifi for Centralized Oracle (●), iMESA (⬟), riMESA (★), kiMESA (■), DDF-SAM2 (✖), DLGBP (▲), and Independent (♦). Unlike prior works that struggle to converge or diverge entirely (DDF-SAM2 at $s \approx 4200$), riMESA closely matches the Centralized Oracle across the entire trial.

To validate this high-level takeaway, we look at the average performance gap of the tested methods relative to the Centralized GNC baseline. Centralized GNC was selected as it is the best-performing baseline that could (in limited settings with reliable communication) be practically deployed (e.g. LAMP 2.0 [26]). The performance gap is computed between a method (m) and this baseline (GNC) as $[(\text{iATE}_m - \text{iATE}_{\text{GNC}})/\text{iATE}_{\text{GNC}}] \times 100$. This measure is averaged across all the real-world datasets, and for each method is:

riMESA ★	kiMESA ■	DDF-SAM2 ✖	DLGBP ▲
45.09%	157.70%	787.30%	352.81%

where we can see that riMESA's optimality gap is **>7x** lower than DLGBP and **>17x** lower than DDF-SAM2. Performance improvement over kiMESA is less significant, but recall this method is an ablative comparison for riMESA's robust local optimization. The performance gap relative to the baseline is non-trivial at 45%; however, unlike the Centralized GNC baseline, riMESA must contend with limited communication between robots, and a performance drop is expected.

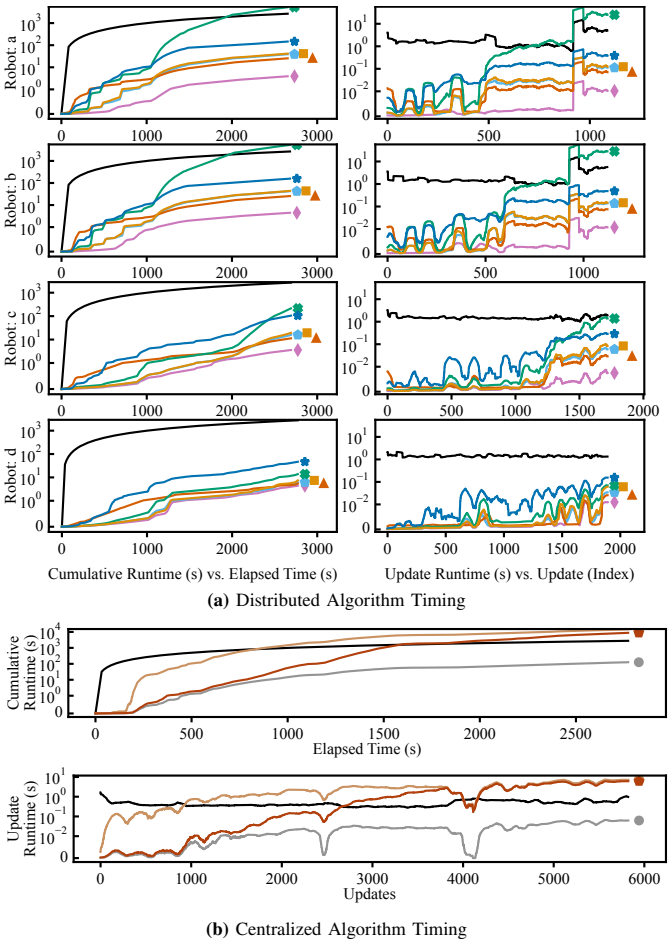
From this experiment we can derive some additional insights. Interestingly, we can see that Centralized PCM can struggle to produce quality results on some datasets. This poor performance is likely caused by PCM misclassifying outlier measurements, which are then incorporated into a non-robust optimizer producing unusable solutions. This provides additional evidence for the use of continuation-based approaches like riMESA. We can also see that the most challenging datasets appear to be `kittredge_loop` and `main_campus` (Tab. II). Notably, for these datasets there is a larger gap between the performance of the "oracle" baselines and any robust method. This indicates that these datasets are challenging due to the presence of outliers, though their long duration is likely a compounding factor. On the other hand, the "easiest" datasets appear to be the Nebula datasets (Tab. III) on which most methods perform reasonably. This is likely due to these datasets containing accurate odometry from a system that was fine-tuned for the operation environment [26].

2) Real-World Timing Results: A key challenge of our target problem is that robots operate online and require up-to-date state estimates for downstream tasks. We next evaluate the runtime of riMESA using the real-world COSMO-Bench datasets. We focus on `main_campus_wifi` as its long duration provides a quality stress test for runtime performance.

We use a machine with an Intel Core i9-13900K processor and 128 GB of RAM. Each method was run independently, and care was taken to minimize background processes. We look at both update runtime and total cumulative runtime. To support practical deployments, a method must remain below real-time bounds for both of these measures. We plot the runtimes for all methods in Fig. 10. For distributed algorithms that support parallel computation, we plot the runtime per-robot, for the centralized algorithms, we plot results for the central server.

In Fig. 10 we can see that almost all the distributed algorithms are able to achieve real-time performance. The exception is DDF-SAM2, which struggles with long update runtimes towards the end of the dataset. These slow updates are likely caused by the algorithm's computation of marginals. On the other hand, while the Centralized Oracle is able to maintain real-time performance, both robust centralized algorithms exceed real-time bounds. While there are efficiency improvements that could be implemented to reduce the runtimes of these algorithms (Remark 11), the aggregation of all data simply results in an intractably large problem for these robust centralized methods, even for the relatively small team size of 4 robots that was tested in this experiment.

Remark 11 (Centralized Baseline Operation)**:** *Centralized algorithms are updated for every new measurement. Odometry is integrated into existing solutions, and loop-closures cause batch re-optimization using the previous solution as a warm start. Due to asynchronous measurements from each robot, centralized algorithms perform more updates than individual robots in the distributed case. Practical implementations could aggregate measurements and perform fewer optimization processes. While this would reduce the cumulative runtime of the method, it would increase the per-update runtime and delay new information from being utilized by the team.*



(a) Distributed Algorithm Timing

(b) Centralized Algorithm Timing

Fig. 10: The runtime performance (cumulative and per-update) on `main_campus_wifi`. (a) Distributed algorithm performance – iMESA (⬟), riMESA (★), kiMESA (■), DDF-SAM2 (✖), DLGBP (▲), and Independent (♦). (b) Centralized baseline performance – Centralized Oracle (●), Centralized GNC (●), and Centralized PCM (●). The real-time bounds for the dataset are shown by black lines (—) and is not constant for updates as time between measurements varies. Apart from DDF-SAM2, all distributed methods are able to maintain real-time performance. While the Centralized Oracle can maintain real-time performance, the robust centralized methods required for real-world operation are too computationally expensive to operate in real-time. *Note: The y-axis scales are linear up to 0.01s and 1s for update/cumulative runtime plots, respectively, and log-scale above these thresholds.*

3) Real-World Bandwidth: As noted, across these experiments we simulate unreliable connectivity, limited communication range, and two-general's failures between robots. However, we did not limit communications based on bandwidth. In this section we evaluate the validity of this assumption. For all trials, we record the total bandwidth usage for the entire team each second. Across all trials using the Wi-Fi model, riMESA had a peak bandwidth usage of 625.3 KB/s and an average usage of 54.9 KB/s. Across all trials using the Pro-Radio model, including the Nebula dataset trials, riMESA had a peak usage of 825.3 KB/s and an average usage of 57.6 KB/s.

While these results are conditioned on the communication model and robot trajectories, they do validate our experimental

assumption, as riMESA's bandwidth usage remains below the limits of the Wi-Fi (2000 KB/s) and Pro-Radio (1000 KB/s) models. It is also worth noting that these are conservative numbers, as communication pairs sufficiently far from each other will not interfere, making the practical bandwidth higher. Finally, for applications where bandwidth is extremely limited, communications could be throttled to operate within available bounds. Moreover, our results support riMESA's efficacy under such conditions, like the `ntu_r3_02_wifi` trial in which riMESA had a peak bandwidth usage of only 3.8 KB/s and an average usage of only 0.12 KB/s and still outperformed all prior works and even some centralized methods.

VIII. Conclusion

In this paper, we presented Consensus Alternating Direction Method of Multipliers as a framework to design effective C-SLAM back-ends. We then proposed riMESA, a robust, incremental, and fully distributed C-SLAM back-end that can handle the challenging conditions of real-world deployments and collaboratively produce high quality state estimates for multi-robot teams. riMESA incrementalizes an edge-based C-ADMM process to allow robots to operate incrementally and asynchronously, leverages a robust incremental local solver (riSAM) to address outlier measurements and compute updates in real-time, and utilizes a robust communication procedure to tolerate unreliable communication with latency. We evaluate the performance of riMESA under a variety of conditions on a total of 2430 unique synthetic datasets and on a total of 28 real-world datasets – validating that riMESA consistently produces high-quality state estimates for a multi-robot team in real-time with only limited communication.

IX. Acknowledgments

This work was partially supported by NASA award 80NSSC24CA020 and the NSF Graduate Research Fellowship Program.

References

[1] P. Y. Lajoie, B. Ramtoula, F. Wu, and G. Beltrame, "Towards collaborative simultaneous localization and mapping: a survey of the current research landscape," *Journal of Field Robotics*, vol. 2, no. 1, 2022.

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. on Robotics (TRO)*, vol. 32, no. 6, 2016.

[3] D. S. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, 2021.

[4] C. Yuan, Y. Zhang, and Z. Liu, "A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques," *Canadian Journal of Forest Research*, vol. 45, no. 7, 2015.

[5] NASA Jet Propulsion Laboratory, "Cooperative Autonomous Distributed Robotic Exploration (CADRE)," https://www.jpl.nasa.gov/missions/cadre.

[6] A. Cunningham, V. Indelman, and F. Dellaert, "DDF-SAM 2.0: Consistent distributed smoothing and mapping," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, DE, May 2013.

[7] R. Murai, J. Ortiz, S. Saeedi, P. H. J. Kelly, and A. J. Davison, "A robot web for distributed many-device localization," *IEEE Trans. on Robotics (TRO)*, vol. 40, 2024.

[8] Y. Tian, K. Khosoussi, and J. P. How, "Resource-aware algorithms for distributed loop closure detection with provable performance guarantees," in *Proc. Workshop on the Algorithmic Foundations of Robotics*. Springer, 2020.

[9] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. Christensen, and F. Dellaert, "Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models," *Intl. J. of Robotics Research (IJRR)*, vol. 36, no. 12, 2017.

[10] E. Cristofalo, E. Montijano, and M. Schwager, "GeoD: Consensus-based geodesic distributed pose graph optimization," arXiv preprint, arXiv:2010.00156 [cs.RO], 2020.

[11] Y. Tian, A. Koppel, A. S. Bedi, and J. P. How, "Asynchronous and parallel distributed pose graph optimization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 4, 2020.

[12] Y. Tian, K. Khosoussi, D. M. Rosen, and J. P. How, "Distributed certifiably correct pose-graph optimization," *IEEE Trans. on Robotics (TRO)*, vol. 37, no. 6, 2021.

[13] T. Fan and T. D. Murphey, "Majorization minimization methods for distributed pose graph optimization with convergence guarantees," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[14] ——, "Majorization minimization methods for distributed pose graph optimization," *IEEE Trans. on Robotics (TRO)*, vol. 40, 2024.

[15] L. Paull, S. Saeedi, M. Seto, and H. Li, "AUV navigation and localization: A review," *IEEE J. of Oceanic Engineering (JOE)*, vol. 39, no. 1, 2014.

[16] E. R. Boroson, R. Hewitt, N. Ayanian, and J. P. de la Croix, "Inter-robot range measurements in pose graph optimization," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[17] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Foundations and Trends in Robotics (FNT)*, vol. 6, no. 1-2, 2017.

[18] E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber, "Some constraints and tradeoffs in the design of network communications," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 1975.

[19] T. Bailey, M. Bryson, H. Mu, J. Vial, L. McCalman, and H. Durrant-Whyte, "Decentralised cooperative localisation for heterogeneous teams of mobile robots," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

[20] R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "An online multi-robot SLAM system for 3d LiDARs," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[21] P. Schmuck and M. Chli, "Multi-UAV collaborative monocular SLAM," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2017.

[22] M. Karrer, P. Schmuck, and M. Chli, "CVI-SLAM—collaborative visual-inertial SLAM," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, 2018.

[23] P. Schmuck and M. Chli, "CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams," *Journal of Field Robotics*, vol. 36, no. 4, 2019.

[24] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli, "COVINS: Visual-inertial SLAM for centralized collaboration," in *Proc. IEEE Intl. Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2021.

[25] Y. Zhang, M. Hsiao, Y. Zhao, J. Dong, and J. J. Engel, "Distributed client-server optimization for SLAM with limited on-device resources," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2021.

[26] Y. Chang, K. Ebadi, C. E. Denniston, M. F. Ginting, A. Rosinol, A. Reinke, M. Palieri, J. Shi, A. Chatterjee, B. Morrell, A. Agha-mohammadi, and L. Carlone, "LAMP 2.0: A robust multi-robot SLAM system for operation in challenging large-scale underground environments," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 4, 2022.

[27] T. Zhang, L. Zhang, Y. Chen, and Y. Zhou, "CVIDS: A collaborative localization and dense mapping framework for multi-agent based visual-inertial SLAM," *IEEE Transactions on Image Processing*, vol. 31, 2022.

[28] Y. Chang, N. Hughes, A. Ray, and L. Carlone, "Hydra-multi: Collaborative online construction of 3d scene graphs with multi-robot teams," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2023.

[29] A. Cramariuc, L. Bernreiter, F. Tschopp, M. Fehr, V. Reijgwart, J. Nieto, R. Siegwart, and C. Cadena, "maplab 2.0 – a modular and multi-modal mapping framework," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 2, 2023.

[30] X. Pan, G. Huang, Z. Zhang, J. Li, H. Bao, and G. Zhang, "Robust collaborative visual-inertial SLAM for mobile augmented reality," *IEEE Trans. on Visualization and Computer Graphics (TVCG)*, vol. 30, no. 11, 2024.

[31] K. Ebadi, L. Bernreiter, H. Biggie, G. Catt, Y. Chang, A. Chatterjee, C. E. Denniston, S. P. Deschênes, K. Harlow, S. Khattak, L. Nogueira, M. Palieri, P. Petráček, M. Petrlík, A. Reinke, V. Krátký, S. Zhao, A. Agha-mohammadi, K. Alexis, C. Heckman, K. Khosoussi, N. Kottege, B. Morrell, M. Hutter, F. Pauling, F. Pomerleau, M. Saska, S. Scherer, R. Siegwart, J. L. Williams, and L. Carlone, "Present and future of SLAM in extreme environments: The DARPA SubT challenge," *IEEE Trans. on Robotics (TRO)*, 2023.

[32] L. Paull, M. Seto, and J. Leonard, "Decentralized cooperative trajectory estimation for autonomous underwater vehicles," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[33] M. J. Schuster, C. Brand, H. Hirschmüller, M. Suppa, and M. Beetz, "Multi-robot 6d graph SLAM connecting decoupled local reference filters," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[34] M. J. Schuster, K. Schmid, C. Brand, and M. Beetz, "Distributed stereo vision-based 6d localization and mapping for multi-robot teams," *Journal of Field Robotics*, vol. 36, no. 2, 2019.

[35] R. Dubois, A. Eudes, J. Moras, and V. Frémont, "Dense decentralized multi-robot SLAM based on locally consistent TSDF submaps," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[36] Y. Huang, T. Shan, F. Chen, and B. Englot, "DiSCo-SLAM: Distributed scan context-enabled multi-robot LiDAR SLAM with two-stage global-local graph optimization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 2, 2022.

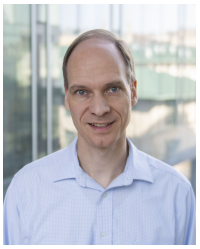
[37] P. Y. Lajoie and G. Beltrame, "Swarm-SLAM: Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems," *IEEE Robotics and Automation Letters (RA-L)*, vol. 9, no. 1, 2024.

[38] X. Liu, J. Lei, A. Prabhu, Y. Tao, I. Spasojevic, P. Chaudhari, N. Atanasov, and V. Kumar, "SlideSLAM: Sparse, lightweight, decentralized metric-semantic SLAM for multi-robot navigation," arXiv preprint, arXiv:2406.17249 [cs.RO], 2025.

[39] O. Shorinwa, T. Halsted, J. Yu, and M. Schwager, "Distributed optimization methods for multi-robot systems: Part 1—a tutorial," *IEEE Robotics & Automation*

Magazine, vol. 31, no. 3, 2024.
[40] ——, "Distributed optimization methods for multi-robot systems: Part 2—a survey," *IEEE Robotics & Automation Magazine*, vol. 31, no. 3, 2024.
[41] 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 *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Hamburg, DE, Oct. 2015.
[42] A. Cunningham, M. Paluri, and F. Dellaert, "DDF-SAM: Fully distributed slam using constrained factor graphs," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, TW, Oct. 2010.
[43] J. Mangelson, D. Dominic, R. Eustice, and R. Vasudevan, "Pairwise consistent measurement set maximization for robust multi-robot map merging," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Brisbane, AU, May 2018.
[44] B. Forsgren, M. Kaess, R. Vasudevan, T. W. McLain, and J. G. Mangelson, "Group-k consistent measurement set maximization via maximum clique over k-uniform hypergraphs for robust multi-robot map merging," *Intl. J. of Robotics Research (IJRR)*, vol. 43, no. 14, 2024.
[45] Y. Latif, C. Cadena, and J. Neira, "Realizing, reversing, recovering: Incremental robust loop closing over time using the iRRR algorithm," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Vilamoura, PT, 2012.
[46] ——, "Robust loop closing over time for pose graph SLAM," *Intl. J. of Robotics Research (IJRR)*, vol. 32, no. 14, 2013.
[47] E. Olivastri and A. Pretto, "IPC: Incremental probabilistic consensus-based consistent set maximization for slam backends," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Yokohama, JP, May 2024.
[48] Z. Zhang, "Parameter estimation techniques: a tutorial with application to conic fitting," *Image and Vision Computing*, vol. 15, no. 1, 1997.
[49] K. Aftab and R. Hartley, "Convergence of iteratively re-weighted least squares to robust m-estimators," in *IEEE Winter Conference on Applications of Computer Vision*, Waikoloa, USA, 2015.
[50] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Intl. J. of Robotics Research (IJRR)*, vol. 31, no. 2, 2012.
[51] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard, "Robust map optimization using dynamic covariance scaling," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, DE, May 2013.
[52] E. Olson and P. Agarwal, "Inference on networks of mixtures for robust robot mapping," *Intl. J. of Robotics Research (IJRR)*, vol. 32, no. 7, 2013.
[53] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*, 2nd ed. Cambridge University Press, 2003.
[54] N. Sünderhauf and P. Protzel, "Switchable constraints for robust pose graph SLAM," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Vilamoura, PT, 2012.
[55] K. J. Doherty, Z. Lu, K. Singh, and J. J. Leonard, "Discrete-continuous smoothing and mapping," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 4, 2022.
[56] M. Black and A. Rangarajan, "On the unification of line processes, outlier rejection, and robust statistics with applications in early vision," *Intl. J. of Computer Vision*, vol. 19, no. 1, 1996.
[57] N. Chebrolu, T. Läbe, O. Vysotska, J. Behley, and C. Stachniss, "Adaptive robust kernels for non-linear least squares problems," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, 2021.
[58] P. Y. Lajoie, S. Hu, G. Beltrame, and L. Carlone, "Modeling perceptual aliasing in SLAM via discrete continuous graphical models," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 2, 2019.
[59] H. Yang, P. Antonante, V. Tzoumas, and L. Carlone, "Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 2, 2020.
[60] D. McGann, J. G. Rogers III, and M. Kaess, "Robust incremental smoothing and mapping (riSAM)," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, London, GB, May 2023.
[61] 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 (RA-L)*, vol. 5, no. 2, 2020.
[62] S. Zhong, Y. Qi, Z. Chen, J. Wu, H. Chen, and M. Liu, "DCL-SLAM: A distributed collaborative LiDAR SLAM framework for a robotic swarm," *IEEE Sensors Journal*, 2023.
[63] H. Xu, P. Liu, X. Chen, and S. Shen, "d^2slam: Decentralized and distributed collaborative visual-inertial SLAM system for aerial swarm," *IEEE Trans. on Robotics (TRO)*, vol. 40, 2024.
[64] Y. Tian, Y. Chang, F. Herrera-Arias, C. Nieto-Granda, J. P. How, and L. Carlone, "Kimera-multi: Robust, distributed, dense metric-semantic SLAM for multi-robot systems," *IEEE Trans. on Robotics (TRO)*, vol. 38, no. 4, 2022.
[65] Y. Tian, Y. Chang, L. Quang, A. Schang, C. Nieto-Granda, J. P. How, and L. Carlone, "Resilient and distributed multi-robot visual SLAM: Datasets, experiments, and lessons learned," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2023.
[66] D. McGann, K. Lassak, and M. Kaess, "Asynchronous distributed smoothing and mapping via on-manifold consensus ADMM," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Yokohama, JP, May 2024.
[67] G. Mateos, J. A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, 2010.
[68] S. Boyd, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, 2010.
[69] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the ADMM in dellized consensus optimization," *IEEE Transactions on Signal Processing*, vol. 62, no. 7, 2014.
[70] O. Shorinwa, T. Halsted, and M. Schwager, "Scalable distributed optimization with separable variables in multi-agent networks," in *Proc. American Control Conference (ACC)*, Denver, USA, Jul. 2020.
[71] E. Wei, "Distributed optimization and market analysis of networked systems," PhD thesis, Massachusetts Institute of Technology, Boston, MA, September 2014.
[72] E. Wei and A. Ozdaglar, "On the O(1/k) convergence of asynchronous distributed alternating direction method of multipliers," in *Proc. IEEE Global Conference on Signal and Information Processing*, Austin, USA, 2013.
[73] D. McGann and M. Kaess, "iMESA: Incremental distributed optimization for collaborative simultaneous localization and mapping," in *Proc. Robotics: Science and Systems (RSS)*, Delft, NL, Jun. 2024.
[74] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Institute of Technology, Technical Report, 2012.
[75] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G^2o: A general framework for graph optimization," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, CN, May 2011.
[76] S. Agarwal, K. Mierle, and The Ceres Solver Team, "Ceres Solver," 10 2023. [Online]. Available: https://github.com/ceres-solver/ceres-solver
[77] A. Haarbach, T. Birdal, and S. Ilic, "Survey of higher order rigid body motion interpolation methods for keyframe animation and continuous-time trajectory estimation," in *Proc. International Conference on 3D Vision (3DV)*, Verona, IT, 2018.
[78] A. Kovnatsky, K. Glashoff, and M. M. Bronstein, "MADMM: a generic algorithm for non-smooth optimization on manifolds," in *Proc. Eur. Conf. on Computer Vision (ECCV)*, Amsterdam, NL, 2016.
[79] J. Solà, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," arXiv preprint, arXiv:1812.01537 [cs.RO], 2021.
[80] K. Matsuka and S. J. Chung, "Localized and incremental probabilistic inference for large-scale networked dynamical systems," *IEEE Trans. on Robotics (TRO)*, vol. 39, no. 5, 2023.
[81] M. Hong, Z. Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM Journal on Optimization*, vol. 26, no. 1, 2016.
[82] Y. Wang, W. Yin, and J. Zeng, "Global convergence of ADMM in nonconvex nonsmooth optimization," *Journal of Scientific Computing*, vol. 78, no. 1, Jan. 2019.
[83] J. Li, S. Ma, and T. Srivastava, "A riemannian ADMM," arXiv preprint, arXiv:2211.02163 [math.OC], 2023.
[84] K. Sun and X. A. Sun, "A two-level distributed algorithm for nonconvex constrained optimization," *Computational Optimization and Applications*, vol. 84, no. 2, Mar. 2023.
[85] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proc. Fifteenth Conference on Uncertainty in Artificial Intelligence*, Stockholm, SE, 1999.
[86] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Rome, Italy, Apr. 2007.
[87] D. McGann, E. R. Potokar, and M. Kaess, "COSMO-Bench: A benchmark for collaborative SLAM optimization," arXiv preprint, arXiv:2508.16731 [cs.RO], 2025.
[88] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry," in *Proc. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Matrid, ES, Oct. 2018.
[89] T. M. Nguyen, S. Yuan, T. H. Nguyen, P. Yin, H. Cao, L. Xie, M. Wozniak, P. Jensfelt, M. Thiel, J. Ziegenbein, and N. Blunder, "MCD: Diverse large-scale multi-campus dataset for robot perception," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Seattle, US, 2024.
[90] D. Albin, D. McGann, M. Mena, A. Thomas, H. Biggie, X. Sun, S. McGuire, J. How, and C. Heckman, "CU-Multi: A dataset for multi-robot collaborative perception," arXiv preprint, arXiv:2509.19463 [cs.RO], 2025.

Daniel McGann received his B.S. in Computer Science from Northeastern University, Boston, MA, USA, in 2020 and his Ph.D. in Robotics from Carnegie Mellon University, Pittsburgh, PA, USA, in 2025. His research to date has focused on distributed optimization to enable Simultaneous Localization and Mapping (SLAM) for multi-robot teams. He is broadly interested in research to develop reliable state estimation and perception for robots and multi-robot teams operating in the wild.



Michael Kaess is a Professor in the Robotics Institute at the School of Computer Science, Carnegie Mellon University (CMU). Before joining CMU in 2013, he was a Research Scientist and Postdoctoral Associate at the Computer Science and Artificial Intelligence Laboratory (CSAIL) at the Massachusetts Institute of Technology (MIT). He earned his PhD in Computer Science from the Georgia Institute of Technology in 2008. His research focuses on probabilistic methods for robot perception, including navigation, mapping, localization, and inference.