©SHUTTERSTOCK.COM/FLASH VECTOR

Part 1—A Tutorial

Distributed Optimization Methods for Multi-Robot Systems

By Ola Shorinwa, Trevor Halsted, Javier Yu, and Mac Schwager

Distributed optimization provides a framework for deriving distributed algorithms for a variety of multi-robot problems. This tutorial constitutes the first part of a two-part series on distributed optimization applied to multi-robot problems, which seeks to advance the application of distributed optimization in robotics. In this tutorial, we demonstrate that many canonical multi-robot problems can be cast within a distributed optimization framework, such as multi-robot simultaneous localization and mapping (SLAM), multi-robot target tracking, and multi-robot task assignment problems. We identify three broad categories of distributed optimization algorithms: distributed first-order (DFO) methods, distributed sequential convex programming, and the alternating direction method of multipliers (ADMM). We describe the basic algorithmic structure of each category and provide representative algorithms

Digital Object Identifier 10.1109/MRA.2024.3358718
Date of publication 13 February 2024; date of current version 11 September 2024.

within each category. We then work through a simulation case study of multiple drones collaboratively tracking a ground vehicle. We compare solutions to this problem using a number of different distributed optimization algorithms. In addition, we implement a distributed optimization algorithm in hardware on a network of Raspberry Pis communicating with XBee modules to illustrate robustness to the challenges of real-world communication networks.

INTRODUCTION

Distributed optimization is the problem of minimizing a joint objective function subject to constraints using an algorithm implemented on a network of communicating computation nodes. In this tutorial, we specifically consider the computation nodes as robots and the network as a multi-robot mesh network. While distributed optimization has been a long-standing topic of research in the optimization community (e.g., [1] and [2]), its usage in multi-robot systems is limited to only a handful of examples. However, we contend that many problems in multi-robot coordination and collaboration can be formulated and solved within the framework of distributed optimization, yielding a powerful new tool for multi-robot systems. We show in this tutorial that cooperative estimation [3], distributed SLAM, multiagent learning [4], and collaborative motion planning [5] are all amenable to approaches based on distributed optimization.

This tutorial constitutes the first part of a two-part series on distributed optimization methods for multi-robot systems. In the first part (the tutorial), we focus on introducing the concepts of distributed optimization in application to a broad class of multi-robot problems. The second part (the survey) will provide a survey of existing distributed optimization methods and highlight open research problems in distributed optimization for multi-robot systems. This series is directed toward robotics researchers and practitioners interested in learning about distributed optimization techniques and their potential to yield novel solutions to problems in multi-robot coordination.

We consider problems that are separable, meaning that the joint objective function can be expressed as a sum over each robot's local objective functions and that the joint constraints can be expressed as the intersection over the robots' local constraints. Each robot requires knowledge only of its own local objective and constraints and communicates only with one-hop neighbors in a mesh network. The algorithms we discuss are homogeneous, in that each robot executes the same algorithmic steps. There is no specialized leader robot and no hierarchy or differentiated role assignments, and no robot has knowledge of the joint objective or constraints. In general, these algorithms are iterative, with each robot sharing its intermediate decision variables and/or problem gradients with its one-hop neighbors at each iteration. As the iterations proceed, the decision variables of all the robots converge to a common solution of the optimization problem. In convex problems, each robot obtains a globally optimal solution to the joint problem. In nonconvex problems, the robots typically reach consensus on a locally optimal solution. (This is the behavior we often observe in practice, although analytical convergence and consensus guarantees for the nonconvex case remain an open area of research.)

We describe three broad classes of optimization algorithms: DFO methods (in which the update procedure for the iterates requires each robot to compute a gradient of its local objective function), distributed sequential convex methods (in which the update procedure for the iterates requires the robots to compute higher-order derivatives, such as Hessians, in addition to gradients), and ADMM methods (in which each robot optimizes a full subproblem at each iteration). We give key examples from each class and discuss their implementation details. We also implement these algorithms in an example scenario in which multiple aerial robots collaborate to estimate the trajectory of a moving target. Finally, we demonstrate a hardware implementation of an ADMM algorithm on a network of Raspberry Pis communicating with XBee radios.

“MANY PROBLEMS IN MULTI-ROBOT COORDINATION AND COLLABORATION CAN BE FORMULATED AND SOLVED WITHIN THE FRAMEWORK OF DISTRIBUTED OPTIMIZATION.”

In some cases, it may not be obvious that a multi-robot problem is of the appropriate form for a distributed optimization algorithm. One may have to manipulate the problem formulation to express it as a separable optimization. We demonstrate in this tutorial that many core multi-robot problems, namely, multi-robot SLAM, multi-robot target tracking, multi-robot task assignment, collaborative trajectory planning, and multi-robot learning, can be cast in this form. Optimization-based approaches often provide new flexibility, new insights, and new performance guarantees in solving multi-robot problems. For example, multi-robot target tracking problems are typically solved via filtering or smoothing approaches, leading to challenges in managing the cross correlation of local measurements [6]. Formulating multi-robot target tracking problems as optimization problems avoids these drawbacks.

CENTRALIZED VERSUS DISTRIBUTED OPTIMIZATION

In principle, multi-robot problems can be solved through centralized optimization. This could be done by passing all information to a leader robot or a base station to perform the computation centrally. However, such centralized techniques are not scalable to large groups of robots, require large amounts of communication to aggregate the data at one location, and introduce a single point of vulnerability (the leader or base station) to faults and attacks. Instead, distributed optimization algorithms enable each robot to obtain an optimal solution of the joint problem locally through communications with one-hop neighbors, without a leader or single point of failure.

Distributed optimization algorithms also have an inherent data privacy property. The robots co-optimize a joint objective without sharing their local "problem data" with one another. Specifically, while robots communicate the value of their local decision variables and/or gradients, they do not expose the functional form of their objective and constraint functions or directly communicate raw sensor data with one another. This may facilitate cooperation across competing manufacturers or competing service providers without exposing proprietary data or violating data privacy laws.

Despite their many advantages, distributed optimization algorithms do come with some drawbacks compared to centralized methods. Since each robot progressively obtains more information via communication with its neighbors, we observe that distributed optimization algorithms require a greater number of iterations for convergence than their centralized counterparts, and they often require a longer computation time to converge compared to centralized methods, particularly in small-scale problems. However, there seems to be little research comparing the empirical or theoretical performance of distributed versus centralized optimization algorithms, which presents an interesting direction for future research. Some distributed algorithms can also be sensitive to hyperparameter tuning, can have a strong reliance on synchronous algorithmic updates, and can be intolerant of dynamically changing networks. In this tutorial, we highlight which algorithm classes suffer from these challenges and discuss practical ways to accommodate these requirements in robotics problems.

"OPTIMIZATION-BASED APPROACHES OFTEN PROVIDE NEW FLEXIBILITY, NEW INSIGHTS, AND NEW PERFORMANCE GUARANTEES IN SOLVING MULTI-ROBOT PROBLEMS."

CONTRIBUTIONS
This tutorial article has four primary objectives:
1) describe three main classes of distributed optimization algorithms
2) highlight the practical implications of typical assumptions made by distributed optimization algorithms and provide potential strategies for addressing the associated challenges
3) demonstrate the formulation of many canonical multi-robot problems as distributed optimization problems
4) provide a case study comparing multiple different distributed optimization algorithms in a multidrone target tracking scenario, both in simulation and on networking hardware.

ORGANIZATION
We present notation and mathematical preliminaries in the "Notation and Preliminaries" section and formulate the general separable distributed optimization problem in the "Problem Formulation" section. The "Classes of Distributed Optimization Algorithms" section describes the three main categories of distributed optimization algorithms and provides representative algorithms for each category. In the "Multi-robot Problems Posed as Distributed Optimizations" section, we demonstrate that many multi-robot problems can be cast within the framework of distributed optimization. In the "Notes on Implementation, Practical Performance, and Limitations" section, we offer implementation tips and practical performance observations and discuss limitations of these methods. The "Distributed Multidrone Vehicle Tracking: A Case Study" section gives a demonstration of distributed optimization algorithms applied to a multidrone vehicle tracking problem in simulation and hardware, and we give concluding remarks in the "Conclusion" section.

NOTATION AND PRELIMINARIES
In this section, we introduce the notation used in this article and provide the definitions of mathematical concepts relevant to the discussion of the distribution optimization algorithms. We denote the gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ as ∇f and its Hessian as $\nabla^2 f$. We denote the vector containing all ones as $\mathbf{1}_n$, where n represents the number of elements in the vector. We next discuss some relevant notions of the connectivity of a graph.

DEFINITION 1: CONNECTIVITY OF AN UNDIRECTED GRAPH
An undirected graph $\mathcal{G}$ is connected if a path exists between every pair of vertices (i, j), where $i, j \in \mathcal{V}$. Note that such a path might traverse other vertices in $\mathcal{G}$.

DEFINITION 2: CONNECTIVITY OF A DIRECTED GRAPH
A directed graph $\mathcal{G}$ is strongly connected if a directed path exists between every pair of vertices (i, j), where $i, j \in \mathcal{V}$. In addition, a directed graph $\mathcal{G}$ is weakly connected if the underlying undirected graph is connected. The underlying undirected graph $\mathcal{G}_u$ of a directed graph $\mathcal{G}$ refers to a graph with the same set of vertices as $\mathcal{G}$ and a set of edges obtained by considering each edge in $\mathcal{G}$ a bidirectional edge. Consequently, every strongly connected directed graph is weakly connected; however, the converse is not true.

DEFINITION 3: STOCHASTIC MATRIX
A nonnegative matrix $W \in \mathbb{R}^{n \times n}$ is referred to as a *row-stochastic matrix* if

$$W\mathbf{1}_n = \mathbf{1}_n. \tag{1}$$

In other words, the sum of all elements in each row of the matrix equals one. We refer to W as a *column-stochastic matrix* if

$$\mathbf{1}_n^\top W = \mathbf{1}_n^\top. \tag{2}$$

Likewise, for a doubly stochastic matrix W,

$$W\mathbf{1}_n = \mathbf{1}_n, \text{ and } \mathbf{1}_n^\top W = \mathbf{1}_n^\top. \tag{3}$$

In distributed optimization in multi-robot systems, robots perform communication and computation steps to minimize some joint objective function. We focus on problems in which the robots' exchange of information must respect the topology of an underlying distributed communication graph, which could possibly change over time. This communication graph, denoted as $\mathcal{G}(t) = (\mathcal{V}(t), \mathcal{E}(t))$, consists of vertices $\mathcal{V}(t) = \{1,\ldots,N\}$ and edges $\mathcal{E}(t) \subseteq \mathcal{V}(t) \times \mathcal{V}(t)$ over which pairwise communication can occur. For undirected graphs, we denote the set of neighbors of robot i as $\mathcal{N}_i(t)$. For directed graphs, we refer to the set of robots that can send information to robot i as the set of *in neighbors* of robot i, denoted by $\mathcal{N}_i^+(t)$. Likewise, for directed graphs, we refer to the set of robots that can receive information from robot i as the *out neighbors* of robot i, denoted by $\mathcal{N}_i^-(t)$.

PROBLEM FORMULATION

We consider a general separable distributed optimization problem of the form

$$\begin{aligned} \min_{x} \quad & \sum_{i \in \mathcal{V}} f_i(x) \\ \text{subject to} \quad & g_i(x) = 0 \quad \forall i \in \mathcal{V} \\ & h_i(x) \leq 0 \quad \forall i \in \mathcal{V} \end{aligned} \tag{4}$$

where $x \in \mathbb{R}^n$ denotes the joint optimization variable, $f_i : \mathbb{R}^n \to \mathbb{R}$ is the local objective function for robot i, $g_i : \mathbb{R}^n \to \mathbb{R}$ is the equality constraint function of robot i, and $h_i : \mathbb{R}^n \to \mathbb{R}$ denotes its inequality constraint function. Each robot $i \in \mathcal{V}$ has access to its local objective constraint functions but has no knowledge of the local objective and constraint functions of the other robots. Such problems arise in many robotics applications where the local objective functions depend on data collected locally by each robot, often in the form of measurements taken by sensors attached to the robots. The robots seek to collectively solve this joint optimization problem without a leader or central coordinator. We note that not all robots need to have a local constraint function. In these cases, the corresponding constraint functions are omitted in (4).

We consider distributed algorithms in which each robot maintains a local copy of the optimization variable, with x_i denoting robot i's local vector of optimization variables. Distributed optimization algorithms solve an equivalent reformulation of the optimization problem (4), given by

$$\begin{aligned} \min_{\{x_i, \forall i \in \mathcal{V}\}} \quad & \sum_{i \in \mathcal{V}} f_i(x_i) \\ \text{subject to} \quad & x_i = x_j \quad \forall (i,j) \in \mathcal{E} \\ & g_i(x_i) = 0 \quad \forall i \in \mathcal{V} \\ & h_i(x_i) \leq 0 \quad \forall i \in \mathcal{V}. \end{aligned} \tag{5}$$

We call $x_i = x_j \quad \forall (i,j) \in \mathcal{E}$ the *consensus constraints*. Under the assumption that the communication graph is connected for undirected graphs and weakly connected for directed graphs, the optimal cost in (5) is equivalent to that in (4), and the minimizing arguments x_i^* in (5) are equal to the minimizing argument x^* of (4) for all robots $i = 1,\ldots,n$. To simplify notation, we introduce the set $\mathcal{X}_i = \{x_i \mid g_i(x_i) = 0, h_i(x_i) \leq 0\}$, representing the feasible set given the constraint functions g_i and h_i. Consequently, we can express the problem in (5) succinctly as follows:

$$\begin{aligned} \min_{\{x_i \in \mathcal{X}_i, \forall i \in \mathcal{V}\}} \quad & \sum_{i \in \mathcal{V}} f_i(x_i) \\ \text{subject to} \quad & x_i = x_j \quad \forall (i,j) \in \mathcal{E}. \end{aligned} \tag{6}$$

CLASSES OF DISTRIBUTED OPTIMIZATION ALGORITHMS

In this section, we categorize distributed optimization algorithms into three broad classes—DFO methods, distributed sequential convex programming, and ADMM methods—based on shared mechanisms for achieving convergence (and not necessarily based on their applicability to multi-robot problems). We provide a brief overview of each category by considering a representative distributed algorithm within each category. In the subsequent discussion, we consider the separable optimization problem in (6).

Before describing the specific algorithms that solve distributed optimization problems, we first consider the general framework that all these approaches share. Each algorithm progresses over discrete iterations $k = 0, 1, \ldots$ until convergence. In general, each iteration consists of a communication step and a computation step. Besides assuming that each robot has the sole capability of evaluating its local objective function f_i, we distinguish between the "internal" variables $\mathcal{P}_i^{(k)}$ that the robot computes at each iteration k and the "communicated" variables $\mathcal{Q}_i^{(k)}$ that the robot communicates to its neighbors. Each algorithm also involves parameters $\mathcal{R}_i^{(k)}$, which generally require coordination among all the robots but can typically be assigned before deployment of the system.

In distributed optimization, all the robots seek to collectively minimize the joint objective function in (6) while achieving consensus on a common set of minimizing optimization variables. Each of the three classes we describe treats the consensus constraints in (6) differently. In DFO methods, from the perspective of a single robot, the update iterations represent a tradeoff between the optimality of a robot's individual solution based on its local objective function versus reaching agreement with its neighbors, either on the decision variable directly or on the gradient of the global objective. Asymptotically, the robots' decision variables or gradients converge to a consensus, leading to global optimality for convex problems. In distributed sequential convex methods, individual robots use communication to build approximate global Hessians and gradients to execute approximate second-order update steps, asymptotically leading each agent to obtain a global minimum in the convex case. Finally, for the ADMM, these consensus constraints are enforced explicitly through an augmented Lagrangian constrained optimization approach. The key insight underlying this approach is that minimizing the local objective functions subject to these additional agreement constraints is equivalent to minimizing the joint objective function over a collective decision variable.

DFO METHODS

Gradient descent methods have been widely applied to solve broad classes of optimization problems, particularly unconstrained problems. To simplify the discussion of these methods, we consider the unconstrained variant of (6), where we retain only the consensus constraints and disregard the constraint functions $g_i(x_i)$ and $h_i(x_i)$. We note that extensions of gradient descent to constrained optimization typically involve a projection of the iterates to the feasible set, a method known as *projected gradient descent*. In the second part of our series [7], we discuss extensions of gradient descent methods to constrained optimization in greater detail. In general, gradient descent methods require only the computation of the gradient (i.e., the first derivative of the objective and constraint functions); hence, these methods are also referred to as *first-order methods*. When applied to the unconstrained joint optimization problem, the updates to the optimization variable take the form

$$x^{(k+1)} = x^{(k)} - \alpha^{(k)} \nabla f(x^{(k)}) \tag{7}$$

where $\alpha^{(k)}$ denotes a diminishing step-size and $\nabla f(x^{(k)})$ denotes the gradient of the objective function, given by

$$\nabla f(x) = \sum_{i \in \mathcal{V}} \nabla f_i(x). \tag{8}$$

From (8), computation of $\nabla f(x)$ requires knowledge of the objective function of all the robots, which is unavailable to any individual robot, and thus requires aggregation of this information at a central node.

DFO algorithms circumvent this underlying challenge by enabling each robot to utilize only its local gradients while communicating with its neighbors to reach consensus on a common solution. In many DFO methods, a robot aggregates the information of its neighbors by taking the weighted combination of the local variables or gradients as specified by a stochastic weighting matrix W. The stochastic matrix W must be compatible with the underlying communication network (i.e., w_{ij} is nonzero only if robot j can send information to robot i).

We begin with a basic distributed gradient descent method, described by the update procedure

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} x_j^{(k)} - \alpha^{(k)} \nabla f_i\left(x_i^{(k)}\right) \tag{9}$$

where each robot mixes its local estimates with those of its neighbors by taking a weighted combination of these local estimates before taking a step in the direction of its local gradient. More generally, a subgradient $\partial f_i(x_i^{(k)})$ (where ∂f_i denotes the subgradient of f_i) can be utilized in place of the gradient of the local objective function, yielding the canonical distributed subgradient method [8]. This paradigm, consisting of taking a weighted combination of local estimates prior to a descent step, is referred to as the *combine-then-adapt* (*CTA*) paradigm. In contrast, in adapt-then-combine (ATC) methods, each robot updates its local optimization variable using its gradient prior to combining its local variable with that of its neighbors, with the update procedure given by

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} \left(x_j^{(k)} - \alpha^{(k)} \nabla f_i\left(x_i^{(k)}\right)\right) \tag{10}$$

where $x_j^{(k)} \in \mathbb{R}^n$ denotes the local variable of neighboring robot j and each robot updates its local variable $x_i^{(k+1)}$ using the local gradient before communicating its local variable with its neighbors and aggregating their respective updates. Consequently, we can further categorize DFO methods into two broad subclasses—ATC methods and CTA methods—based on the relative order of the communication and computation procedures.

In general, the algorithms given by (9) and (10) do not converge to the optimal solution of the joint optimization problem. To see this, consider the case where $x_i = x^\star$, $\forall i \in \mathcal{V}$, where $x^\star$ denotes the optimal solution of the joint optimization problem. In the ATC approach, we can express the update procedure as the difference between two terms: $\Sigma_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} x_j^{(k)}$ and $\Sigma_{j \in \mathcal{N}_i \cup \{i\}} \alpha^{(k)} \nabla f_i(x_i^{(k)})$. Given that W is row stochastic, the first term in ATC and CTA approaches simplifies to $x^\star$. However, in ATC approaches, the second term represents a weighted combination of the local gradients of each robot, which is not necessarily zero. In fact, we have only $\Sigma_{i \in \mathcal{V}} \nabla f_i(x^\star) = 0$ in the general case. Likewise, in CTA methods, the second term represents the local gradient of each agent, which is not necessarily zero. As a result, the iterate $x_i^{(k+1)}$ moves away from the optimal solution $x^\star$.

If $\alpha^{(k)}$ did not asymptotically converge to zero, then the iterates would converge only to a neighborhood of the globally optimal value [observe that substituting the optimal value into (10) or (9) yields a nonzero innovation] [9]. If the step-size satisfies the conditions $\Sigma_{k=0}^{\infty} \alpha(k) = \infty$ and $\Sigma_{k=0}^{\infty} \alpha(k) < \infty$, then convergence of the iterates to an optimal solution is guaranteed [10], [11]. An example of a step-size rule satisfying these conditions is given by $\alpha^{(k)} = \alpha^{(0)}/k$. Although both conditions are sufficient for convergence, only the nonsummable condition is necessary [12]. In practice, an optimal diminishing step-size is given by $\alpha^{(k)} = \alpha^{(0)}/\sqrt{k}$, which is not square summable [13], [12].

In extensions of these basic approaches, we replace the gradient $\nabla f_i(x_i^{(k)})$ with a new variable $y_i^{(k)}$ that uses consensus to aggregate gradient information from the other robots and track the average gradient of the joint objective function. Gradient tracking methods, for example, DIGing [14], employ an estimate of the average gradient computed through dynamic average consensus with

$$y_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} y_j^{(k)} + \left[\nabla f_i\left(x_i^{(k+1)}\right) - \nabla f_i\left(x_i^{(k)}\right)\right]. \tag{11}$$

The iterate $x_i^{(k)}$ of each agent is guaranteed to converge to the optimal solution $x^\star$ under a constant step-size provided the communication network is connected and certain other conditions on the network topology and the objective functions hold [14]. Moreover, the iterate $y_i^{(k)}$ converges to the average gradient of the individual objective functions [15] given convergence of $x_i^{(k)}$ to the limit point $x^\star$. At initialization of the algorithm, all the robots select a common step-size. Further, robot i initializes its local variables with $x_i^{(0)} \in \mathbb{R}^n$ and $y_i^{(0)} = \partial f_i(x_i^{(0)})$. Algorithm 1 summarizes the update procedures in the distributed gradient tracking method DIGing [14]. We note that ATC methods are compatible with uncoordinated step-sizes; i.e., each robot does not have to use the same step-size. Unlike ATC methods, CTA methods require a common step-size among the robots for convergence to an optimal solution.

DISTRIBUTED SEQUENTIAL CONVEX PROGRAMMING

Sequential convex programming entails solving an optimization problem by computing a sequence of iterates representing the solution of a series of approximations of the original problem. Newton's method is a prime example of a sequential convex programming method. In Newton's method, and more generally, quasi-Newton methods, we take a quadratic approximation of the objective function at an operating point $x^{(k)}$, resulting in

$$\tilde{f}(x) = f(x^{(k)}) + \nabla f(x^{(k)})^\top (x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^\top H(x^{(k)})(x - x^{(k)}) \tag{12}$$

where $H(\cdot)$ denotes the Hessian of the objective function, $\nabla^2 f$, or its approximation. Subsequently, we compute a solution to the quadratic program, given by

$$x^{(k+1)} = x^{(k)} - H(x^{(k)})^{-1} \nabla \tilde{f}(x^{(k)}) \tag{13}$$

which requires centralized evaluation of the gradient and Hessian of the objective function. Distributed sequential programming (DSQP) enables each robot to compute a local estimate of the gradient and Hessian of the objective function and thus allows for the local execution of the update procedures. We consider the NEXT algorithm [16] to illustrate this class of distributed optimization algorithms. We assume that each robot uses a quadratic approximation of the optimization problem as its convex surrogate model $U(\cdot)$. In NEXT, each robot maintains an estimate of the average gradient of the objective function as well as an estimate of the gradient of the objective function excluding its local component [e.g., $\Sigma_{j \neq i} f_j(x_i)$ for robot i, which we denote by $\tilde{\pi}_i^{(k)}$]. At a current iterate $x_i^{(k)}$, robot i creates a quadratic approximation of the optimization problem, given by

$$\underset{\tilde{x}_i \in \mathcal{X}_i}{\text{minimize}} \left(\nabla f_i(x_i^{(k)}) + \tilde{\pi}_i^{(k)}\right)^\top (\tilde{x}_i - x_i^{(k)}) + \frac{1}{2}(\tilde{x}_i - x_i^{(k)})^\top H_i(x_i^{(k)})(\tilde{x}_i - x_i^{(k)}) \tag{14}$$

which takes into account the robot's local Hessian H_i or its estimate (e.g., computed using a quasi-Newton update scheme [17], [18], [19]) and can be solved locally. Each robot computes a weighted combination of its current iterate and the solution of (14), given by the procedure

$$z_i^{(k)} = x_i^{(k)} + \alpha^{(k)} (\tilde{x}_i^{(k)} - x_i^{(k)}) \tag{15}$$

where $\alpha^{(k)} \in (0,1)$ denotes a diminishing step-size. Subsequently, robot i computes its next iterate by taking a weighted combination of its local estimate $z_i^{(k)}$ with that of its neighbors via the procedure

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} z_j^{(k)} \tag{16}$$

for consensus on a common solution of the original optimization problem, where the weight $w_{i,j}$ must be compatible with the underlying communication network. In addition, robot i updates its estimates of the average gradient of the objective function, denoted by y_i, using dynamic average consensus in the same form as (11). Updating $\tilde{\pi}_i^{(k)}$ takes a similar form. In the limit that the iterates approach a common value x^*, y_i approaches the average gradient of the joint objective function at x^*, and so does $\tilde{\pi}_i^{(k)} + \nabla f_i(x_i^{(k)})$. Thus, NEXT reasons that an appropriate update for $\tilde{\pi}_i$ takes the following form:

$$\tilde{\pi}_i^{(k+1)} = N \cdot y_i^{(k+1)} - \nabla f_i(x_i^{(k+1)}). \tag{17}$$

Each agent initializes its local variables with $x_i^{(0)} \in \mathbb{R}^n$, $y_i^{(0)} = \nabla f_i(x_i^{(0)})$, and $\tilde{\pi}_i^{(k+1)} = N y_i^{(0)} - \nabla f_i(x_i^{(0)})$ prior to executing the above update procedures. We note that NEXT is guaranteed to converge to a stationary point of the optimization problem [16]. Algorithm 2 summarizes the update procedures in NEXT [16].

Other algorithms that use distributed sequential convex programming include methods that perform a distributed Newton's method [20] and distributed quasi-Newton methods [21]. Furthermore, algorithms that use consensus on local Hessians exist [22], often at the expense of greater communication overhead.

ALGORITHM 1: DIGing

```
Initialization: $k \leftarrow 0, x_i^{(0)} \in \mathbb{R}^n, y_i^{(0)} = \nabla f_i(x_i^{(0)})$
Internal variables: $\mathcal{P}_i^{(k)} = \varnothing$
Communicated variables: $\mathcal{Q}_i^{(k)} = (x_i^{(k)}, y_i^{k})$
Parameters: $\mathcal{R}_i^{(k)} = (\alpha, w_i)$
do in parallel $\forall i \in \mathcal{V}$
    Communicate $\mathcal{Q}_i^{(k)}$ to all $j \in \mathcal{N}_i$
    Receive $\mathcal{Q}_j^{(k)}$ from all $j \in \mathcal{N}_i$
    $x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} x_j^{(k)} - \alpha y_i^{(k)}$
    $y_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} y_j^{(k)} + \nabla f_i(x_i^{(k+1)}) - \nabla f_i(x_i^{(k)})$
    $k \leftarrow k+1$
while stopping criterion is not satisfied
```

ADMM

The ADMM belongs to the class of optimization algorithms referred to as the *method of multipliers* (or *augmented Lagrangian methods*), which compute a primal–dual solution pair of a given optimization problem. The method of multipliers proceeds in an alternating fashion: the primal iterates are updated as minimizers of the augmented Lagrangian, and subsequently, the dual iterates are updated via dual (gradient) ascent on the augmented Lagrangian. The procedure continues iteratively until convergence or termination. The augmented Lagrangian of the problem in (6) (with only the consensus constraints) is given by

$$\mathcal{L}_a(\mathbf{x}, q) = \sum_{i=1}^{N} f_i(x_i) + \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} \left(q_{i,j}^\top (x_i - x_j) + \frac{\rho}{2} \| x_i - x_j \|_2^2 \right) \tag{18}$$

where $q_{i,j}$ represents a dual variable for the consensus constraints between robots i and j, $q = [q_{i,j}^\top, \forall (i,j) \in \mathcal{E}]^\top$, and $\mathbf{x} = [x_1^\top, x_2^\top, \ldots, x_N^\top]^\top$. The parameter $\rho > 0$ represents a penalty term on violations of the consensus constraints. Generally, the method of multipliers computes the minimizer of the augmented Lagrangian with respect to the joint set of optimization variables, which hinders distributed computation. In contrast, in the ADMM, the minimization procedure is performed block component-wise, enabling parallel distributed computation of the minimization subproblem in the consensus problem. However, many ADMM algorithms still require some centralized computation, rendering them not fully distributed in the multi-robot mesh network sense that we consider in this article.

We focus here on ADMM algorithms that are distributed over robots in a mesh network, with each robot executing the same set of distributed steps. We specifically consider the consensus ADMM (C-ADMM) [23] a representative algorithm within this category. The C-ADMM introduces auxiliary optimization variables into the consensus constraints in (6) to enable fully distributed update procedures. The primal update procedure of robot i takes the form

$$x_i^{(k+1)} = \operatorname*{argmin}_{x_i \in \mathcal{X}_i} \left\{ f_i(x_i) + x_i^\top y_i^{(k)} + \rho \sum_{j \in \mathcal{N}_i} \left\| x_i - \frac{1}{2}(x_i^{(k)} + x_j^{(k)}) \right\|_2^2 \right\} \tag{19}$$

ALGORITHM 2: NEXT

```
Initialization: $k \leftarrow 0, x_i^{(0)} \in \mathbb{R}^n, y_i^{(0)} = \nabla f_i(x_i^{(0)}), \tilde{\pi}_i^{(0)} = N y_i^{(0)} - \nabla f_i(x_i^{(0)})$
Internal variables: $\mathcal{P}_i = (x_i^{(k)}, \tilde{x}_i^{(k)}, \tilde{\pi}_i^{(k)})$
Communicated variables: $\mathcal{Q}_i^{(k)} = (z_i^{(k)}, y_i^{(k)})$
Parameters: $\mathcal{R}_i^{(k)} = (\alpha^{(k)}, w_i, U(\cdot), \mathcal{X}_i)$
do in parallel $\forall i \in \mathcal{V}$
    $\tilde{x}_i^{(k)} = \operatorname*{argmin}_{x \in \mathcal{X}_i} U(x; x_i^{(k)}, \tilde{\pi}_i^{(k)})$
    $z_i^{(k)} = x_i^{(k)} + \alpha^{(k)} (\tilde{x}_i^{(k)} - x_i^{(k)})$
    Communicate $\mathcal{Q}_i^{(k)}$ to all $j \in \mathcal{N}_i$
    Receive $\mathcal{Q}_j^{(k)}$ from all $j \in \mathcal{N}_i$
    $x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} z_j^{(k)}$
    $y_i^{(k+1)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} y_j^{(k)}$
            $+ [\nabla f_i(x_i^{(k+1)}) - \nabla f_i(x_i^{(k)})]$
    $\tilde{\pi}_i^{(k+1)} = N \cdot y_i^{(k+1)} - \nabla f_i(x_i^{(k+1)})$
    $k \leftarrow k+1$
while stopping criterion is not satisfied
```

which requires only information locally available to robot i, including information received from its neighbors (i.e., $x_j^k, \forall j \in \mathcal{N}_i$). As a result, this procedure can be executed locally by each agent in parallel. After communicating with its neighbors, each robot updates its local dual variable using the procedure

$$y_i^{(k+1)} = y_i^{(k)} + \rho \sum_{j \in \mathcal{N}_i} (x_i^{(k+1)} - x_j^{(k+1)}) \tag{20}$$

where y_i denotes the composite dual variable of robot i, corresponding to the consensus constraints between robot i and its neighbors, which is initialized to zero. Algorithm 3 summarizes the update procedures in the C-ADMM [23].

SYNOPSIS

We summarize the notable features of each category of distributed algorithms in Table 1, which should be considered when

ALGORITHM 3: C-ADMM

```
Initialization: $k \leftarrow 0, x_i^{(0)} \in \mathbb{R}^n, y_i^{(0)} = 0$
Internal variables: $\mathcal{P}_i^{(k)} = y_i^{(k)}$
Communicated variables: $\mathcal{Q}_i^{(k)} = x_i^{(k)}$
Parameters: $\mathcal{R}_i^{(k)} = \rho$
do in parallel $\forall i \in \mathcal{V}$
    $x_i^{(k+1)} = \operatorname*{argmin}_{x_i \in \mathcal{X}_i} \Big\{ f_i(x_i) + x_i^\top y_i^{(k)} \ldots$
            $+ \rho \sum_{j \in \mathcal{N}_i} \left\| x_i - \frac{1}{2}(x_i^{(k)} + x_j^{(k)}) \right\|_2^2 \Big\}$
    Communicate $\mathcal{Q}_i^{(k)}$ to all $j \in \mathcal{N}_i$
    Receive $\mathcal{Q}_j^{(k)}$ from all $j \in \mathcal{N}_i$
    $y_i^{(k+1)} = y_i^{(k)} + \rho \sum_{j \in \mathcal{N}_i} (x_i^{(k+1)} - x_j^{(k+1)})$
    $k \leftarrow k+1$
while stopping criterion is not satisfied
```

TABLE 1. Suitable distributed optimization algorithms for different complicating attributes common in multi-robot problems.

ATTRIBUTE	DFO (E.G., [14])	DSCP (E.G., [16])	ADMM (E.G., [23])
Dynamic communication networks	✓	✓	×
Lossy communication	✓	✓	×
Unidirectional communication networks	✓	×	×
Bidirectional communication networks	✓	✓	✓
Constrained problems	×	×	✓
Robustness to step-size/ penalty parameter	×	×	✓

The information displayed is based on the representative algorithm (indicated by the citation) considered in each algorithm class. DSCP: distributed sequential convex programming.

selecting a distributed algorithm for a multi-robot problem. In general, the update procedures in DFO algorithms require lower-complexity computational operations, which makes them suitable for problems where each robot has limited access to computational resources [9], [14], [24]. Further, DFO algorithms accommodate dynamic unidirectional and bidirectional communication networks. However, DFO algorithms are generally not amenable to constrained problems, limiting their applications in some multi-robot problems. On the other hand, while DSQP algorithms are suitable for problems with dynamic bidirectional communication networks, these algorithms do not generally extend to unidirectional networks [20], [21]. In addition, while some DSQP algorithms [16], [25] are suitable for constrained optimization, this is not the case for all methods of this class. In contrast, although distributed algorithms based on the ADMM do not address dynamic unidirectional communication networks, ADMM-based algorithms apply to constrained optimization [23], [26]. Moreover, ADMM-based algorithms show better robustness to the selection of algorithm parameters, such as the step-size or penalty parameter. However, ADMM-based methods incur a greater computational overhead, as the optimization subproblems arising in the update procedures do not necessarily have closed-form solutions.

MULTI-ROBOT PROBLEMS POSED AS DISTRIBUTED OPTIMIZATIONS

Many robotics problems have a distributed structure, although this structure might not be immediately apparent. In many cases, applying distributed optimization methods requires reformulating the original problem into a separable form that allows for distributed computation of the problem variables locally by each robot. In this section, we consider five general problem categories that can be solved using distributed optimization tools: multi-robot SLAM, multi-robot target tracking, multi-robot task assignment, collaborative planning, and multi-robot learning. We note that an optimization-based approach to solving some of these problems might not be immediately obvious. However, we show that many of these problems can be quite easily formulated as distributed optimization problems through the introduction of auxiliary optimization variables in addition to an appropriate set of consensus constraints.

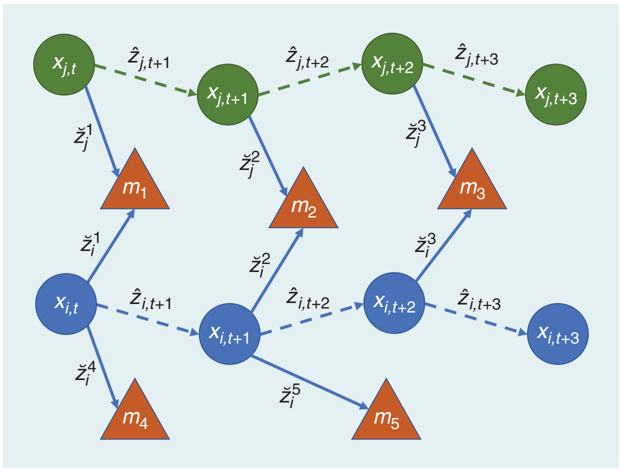

FIGURE 1. A factor graph representation of a multi-robot SLAM problem, where two robots, robot i (blue circles) and robot j (green circles), seek to jointly estimate a set of map features $\{m_1, m_2, \cdots\}$ (orange triangles) in addition to their own pose trajectory $\{x_{i,t}, x_{j,t}, \forall t\}$ from the set of odometry measurements $\{\hat{z}_{i,t}, \hat{z}_{j,t}\}$ and observations of each map feature k $\{\breve{z}_i^k, \ \breve{z}_j^k\}$.

MULTI-ROBOT SLAM

In multi-robot SLAM problems, a group of robots seek to estimate their position and orientation (pose) within a consistent representation of their environment (Figure 1). In a full landmark-based SLAM approach, we consider optimizing over both M map features $m_1, \ldots, m_M$ as well as N robot poses $x_1, \ldots, x_N$ over a duration of $T+1$ time steps:

$$\begin{aligned}\underset{\mathbf{x}, m}{\text{minimize}} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \left\| \bar{z}_{i,t}(x_{i,t}, x_{i,t+1}) - \hat{z}_{i,t+1} \right\|_{\Omega_{i,t}}^2 \\ + \sum_{i=1}^{N} \sum_{k=1}^{M} \left\| \tilde{z}_i^k(x_i, m_k) - \breve{z}_i^k \right\|_{\Lambda_{i,t}}^2.\end{aligned} \tag{21}$$

The z terms denote measurements $(\hat{z}, \breve{z})$ and measurement functions $(\bar{z}, \tilde{z})$: the expected relative poses $\bar{z}_{i,t}$ are functions of two adjacent poses of robot i derived from robot odometry measurements, and the expected relative pose $\tilde{z}_i^k$ is a function of the pose of robot i and the position of map feature k. We have concatenated the problem variables in (21), with $x_i = [x_{i,0}^\top, x_{i,1}^\top, \ldots, x_{i,T}^\top]^\top$, $\mathbf{x} = [x_1^\top, x_2^\top, \ldots, x_N^\top]^\top$, and $m = [m_1^\top, m_2^\top, \ldots, m_M^\top]^\top$. The error terms in the objective function are weighted by the information matrices $\Omega_{i,t}$ and $\Lambda_{i,t}$ associated with the measurements collected by robot i.

Although the first set of terms in the objective function of the optimization problem (21) is separable among the robots, the second set of terms is not. Consequently, the optimization problem must be reformulated. Nonseparability of the objective function arises from the coupling between the map features and the robot poses. To achieve separability of the objective function, we can introduce local copies of the variables corresponding to each feature, with an associated set of consensus (equality) constraints to ensure that the resulting problem remains equivalent to the original problem (21). The resulting problem takes the form

$$\begin{aligned}\underset{\mathbf{x}, \hat{m}_1, \hat{m}_2, \ldots, \hat{m}_N}{\text{minimize}} \quad & \sum_{i=1}^{N} \sum_{t=0}^{T-1} \left\| \bar{z}_{i,t}(x_{i,t}, x_{i,t+1}) - \hat{z}_{i,t+1} \right\|_{\Omega_{i,t}}^2 \\ & + \sum_{i=1}^{N} \sum_{k=1}^{M} \left\| \tilde{z}_i^k(x_i, \hat{m}_{i,k}) - \breve{z}_i^k \right\|_{\Lambda_{i,t}}^2 \\ \text{subject to} \quad & \hat{m}_i = \hat{m}_j \quad \forall (i, j) \in \mathcal{E}\end{aligned} \tag{22}$$

where robot i maintains $\hat{m}_i$, its local copy of the map m. We note that x_i is the trajectory of robot i and is estimated only by robot i. The problem (22) is separable among the robots, which enforce consensus among their representations of the map; in other words, the objective function can be expressed in the form

$$f(\mathbf{x}, \hat{m}_1, \hat{m}_2, \ldots, \hat{m}_N) = \sum_{i=1}^{N} f_i(x_i, \hat{m}_i) \tag{23}$$

where

$$f_i(x_i, \hat{m}_i) = \sum_{t=0}^{T-1} \left\| \bar{z}_{i,t}(x_{i,t}, x_{i,t+1}) - \hat{z}_{i,t+1} \right\|^2_{\Omega_{i,t}} + \sum_{k=1}^{M} \left\| \tilde{z}_i^k(x_i, \hat{m}_{i,k}) - \check{z}_i^k \right\|^2_{\Lambda_{i,t}}. \quad (24)$$

Note that the consensus constraints involve only a subset of the local variables of each robot. Distributed optimization algorithms are amenable to problems of this form without any significant modifications. In methods requiring a weighting matrix, considering robot i, only variables involved in the consensus constraints are combined (mixed) with those of its neighbors. Likewise, variants of the ADMM, such as the separable optimization variable ADMM [26], can be applied to this problem. We can interpret the bundle adjustment problem similarly. In this case, the map features represent the scene geometry, and the robot poses include the optical characteristics of the respective cameras. However, a challenge in applying this approach in unstructured environments is ensuring that multiple robots agree on the labels of the map landmarks.

An alternative approach is pose graph optimization (PGO), which avoids explicitly estimating the map by representing the robots' trajectories as a graph in which the edges represent the estimated transformation among poses. A pose i consists of a position (which we represent by the vector τ_i) and an orientation (which we represent by the rotation matrix R_i). In this perspective, the task of determining robot trajectories consists of two stages performed sequentially. In the "front end," the robots process raw sensor measurements to estimate relative poses consisting of a relative rotation ($\tilde{R}_{ij} \approx R_i^{-1} R_j$) and relative translation ($\tilde{\tau}_{ij} \approx \tau_j - \tau_i$). The second stage is the "back end," in which robots find optimal robot poses given those relative pose measurements. Under the assumption that the robots can perform the front-end optimization locally [finding ($\tilde{R}_{ij}, \tilde{\tau}_{ij}$) for each edge ($i, j$) in their trajectories], PGO addresses the back-end stage of SLAM. The objective function of PGO, in which the robots determine the set of poses (consisting of a rotation R_i and translation τ_i for each pose i) that best explains the relative pose estimates ($\tilde{R}_{ij}, \tilde{\tau}_{ij}$), is separable and, therefore, amenable to distributed optimization techniques:

$$\min_{\{(R_i,\tau_i)\}_{i=1}^n} \sum_{(i,j)\in\mathcal{E}} \frac{\omega_{ij}}{2} \left\| R_j - R_i \tilde{R}_{ij} \right\|_F^2 + \frac{w_{ij}}{2} \left\| \tau_j - \tau_i - R_i \tilde{\tau}_{ij} \right\|_2^2.$$

While PGO specifically addresses solving the back end of SLAM, some existing distributed techniques that do not rely on distributed optimization have also been proposed for the front end, e.g., [27]. We refer to [28], [29], [30], and [31] for additional details on SLAM and multi-robot SLAM.

Distributed optimization algorithms can be readily applied to the graph-based SLAM problem in (22). Moreover, we note that a number of related robotics problems, including rotation averaging/synchronization and shape registration/alignment, can be similarly reformulated into a separable form and subsequently solved using distributed optimization algorithms [32], [33], [34], [35], [36], [37].

MULTI-ROBOT TARGET TRACKING

In the multi-robot target tracking problem, a group of robots collect measurements of an agent of interest (referred to as a *target*) and seek to collectively estimate the trajectory of the target. Multi-robot target tracking problems arise in many robotics applications ranging from environmental monitoring and surveillance to autonomous robotics applications, such as autonomous driving, where the estimated trajectory of the target can be leveraged for scene prediction to enable safe operation. Figure 2 illustrates the multi-robot target tracking problem where a group of four quadrotors make noisy observations of a flagged ground vehicle (the target). Each colored cone represents the region where each quadrotor can observe the vehicle, given the limited measurement range of the sensors onboard the quadrotor.

Multi-robot target tracking problems can be posed as maximum a posteriori (MAP) optimization problems where the robots seek to compute an estimate that maximizes the posterior distribution of the target's trajectory given the set of all observations of the target made by the robots. When a model of the dynamics of target is available, denoted by $g : \mathbb{R}^n \to \mathbb{R}^n$, the resulting optimization problem takes the form

$$\underset{x}{\text{minimize}} \sum_{t=0}^{T-1} \left\| x_{t+1} - g(x_t) \right\|^2_{\Omega_t} + \sum_{i=1}^{N} \sum_{t=0}^{T-1} \left\| y_{i,t} - h_i(x_t) \right\|^2_{\Lambda_{i,t}} \quad (25)$$

where $x_t \in \mathbb{R}^n$ denotes the pose of the target at time t and $y_{i,t} \in \mathbb{R}^m$ denotes robot i's observation of the target at time t over a duration of $T+1$ time steps. We represent the trajectory of the target with $x = [x_0^\top, x_1^\top, \ldots, x_T^\top]^\top$. While the first term in the objective function corresponds to the error between the estimated state of the target at a subsequent time

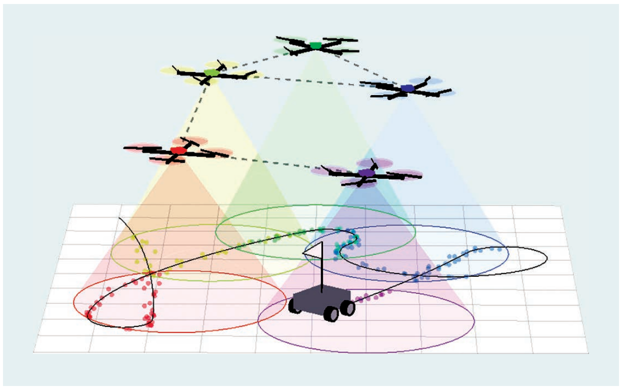

FIGURE 2. A multi-robot target tracking scenario with four quadrotors (the robots) making noisy observations of a flagged ground vehicle (the target). The colored cones represent the regions where each quadrotor can observe the vehicle, given the limited measurement range of the sensors onboard each quadrotor.

step and its expected state based on a model of its dynamics, the second term corresponds to the error between the observations collected by each robot and the expected measurement computed from the estimated state of the target, where the function $h_i : \mathbb{R}^n \to \mathbb{R}^m$ denotes the measurement model of robot i. Further, the information matrices $\Omega_t \in \mathbb{R}^{n \times n}$ and $\Lambda_{i,t} \in \mathbb{R}^{m \times m}$ for the dynamics and measurement models, respectively, weight the contribution of each term in the objective function appropriately, reflecting prior confidence in the dynamics and measurement models. The MAP optimization problem in (25) is not separable and, hence, not amenable to distributed optimization in its current form, due to coupling in the objective function arising from x. Nonetheless, we can arrive at a separable optimization problem through a fairly straightforward reformulation [3]. We can assign a local copy of x to each robot, with $\hat{x}_i$ denoting robot i's local copy of x. The reformulated problem becomes

$$\begin{aligned} \underset{\hat{x}}{\text{minimize}} \quad & \sum_{i=1}^{N} \sum_{t=0}^{T-1} \frac{1}{N} \| \hat{x}_{i,t+1} - g(\hat{x}_{i,t}) \|_{\Omega_t}^2 \\ & + \sum_{i=1}^{N} \sum_{t=0}^{T-1} \| y_{i,t} - h_i(\hat{x}_{i,t}) \|_{\Lambda_{i,t}}^2 \\ \text{subject to} \quad & \hat{x}_i = \hat{x}_j \quad \forall (i, j) \in \mathcal{E} \end{aligned} \tag{26}$$

where $\hat{x} = [\hat{x}_1^\top, \hat{x}_2^\top, \ldots, \hat{x}_N^\top]^\top$. Following this reformulation, distributed optimization algorithms can be applied to compute an estimate of the trajectory of the target from (26).

MULTI-ROBOT TASK ASSIGNMENT

In the multi-robot task assignment problem, we seek an optimal assignment of N robots to M tasks such that the total cost incurred in completing the specified tasks is minimized. However, we note that many task assignment problems consist of an equal number of tasks and robots. The standard task assignment problem has been studied extensively and is typically solved using the Hungarian method [38]. However, optimization-based methods have emerged as a competitive approach due to their amenability to task assignment problems with a diverse set of additional constraints, encoding individual preferences or other relevant problem information, making them a general-purpose approach.

The task assignment problem can be represented as a weighted bipartite graph: a graph whose vertices can be divide into two sets where no two nodes within a given set share an edge. Further, each edge in the graph has an associated weight. In task assignment problems, the edge weight $c_{i,j}$ represents the cost of assigning robot i to task j. Figure 3 depicts a task assignment problem represented by a weighted bipartite graph, with three robots and three tasks. Each robot knows its task preferences only and does not know the task preferences of other robots. Equivalently, the task assignment problem can be formulated as an integer optimization problem. Many optimization-based methods solve a relaxation of the integer optimization problem. Generally, in problems with linear objective functions and affine constraints, these optimization-based methods are guaranteed to yield an optimal task assignment. The associated relaxed optimization problem takes the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i=1}^{N} c_i^\top x_i \\ \text{subject to} \quad & \sum_{i=1}^{N} x_i = 1_M \\ & 1_M^\top x_i = 1 \\ & 0 \leq x \leq 1 \end{aligned} \tag{27}$$

where $x_i \in \mathbb{R}^M$ denotes the optimization variable of robot i, representing its task assignment, and $x = [x_1, x_2, \ldots, x_N]$. Although the objective function of (27) is separable, the optimization problem is not separable, due to coupling of the optimization variables arising in the first constraint. We can obtain a separable problem, amenable to distributed optimization, by assigning a local copy of x to each robot, resulting in the problem

$$\begin{aligned} \underset{\hat{x}}{\text{minimize}} \quad & \sum_{i=1}^{N} c_i^\top \hat{x}_{i,i} \\ \text{subject to} \quad & \sum_{i=1}^{N} \hat{x}_{i,i} = 1_M \\ & 1_M^\top \hat{x}_{i,i} = 1 \\ & 0 \leq \hat{x}_i \leq 1 \quad \forall i \in \mathcal{V} \\ & \hat{x}_i = \hat{x}_j \quad \forall (i,j) \in \mathcal{E} \end{aligned} \tag{28}$$

where $\hat{x}_i \in \mathbb{R}^{M \times N}$ denotes robot i's local copy of x and $\hat{x} = [\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_N]$. Although the reformulation in (28) is simple, it does not scale efficiently with the number of robots and tasks. A more efficient reformulation can be obtained by considering the dual formulation of the task assignment problem. For brevity, we omit a discussion of this approach in this article and refer readers to [39], [40], and [41], where this reformulation scheme is discussed in detail.

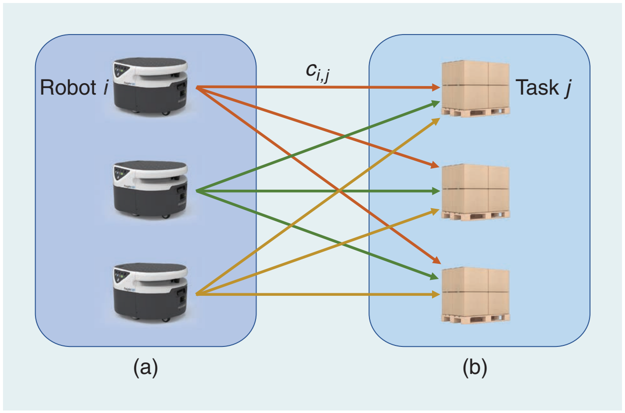

FIGURE 3. A multi-robot task assignment problem represented as a bipartite graph, with (a) three (Fetch) robots and (b) three tasks. An edge with weight $c_{i,j}$ between robot i and task j signifies the cost incurred by robot i if it performs task j. In many problems, each robot's task preferences (edge weights) are neither known by other robots nor accessible to these robots.

COLLABORATIVE PLANNING, CONTROL, AND MANIPULATION

Generally, in collaborative planning problems, we seek to compute state and control input trajectories that enable a group of robots to reach a desired state configuration from a specified initial state while minimizing a trajectory cost and without colliding with other agents. The related multi-robot control problem involves computing a sequence of control inputs that enable a group of robots to track a desired reference trajectory or achieve some specified task, such as manipulating an object collaboratively. Figure 4 presents a collaborative manipulation problem where three quadrotors move an object collaboratively. The dashed line represents the reference trajectory for manipulating the load.

Collaborative multi-robot planning, control, and manipulation problems have been well studied, with a broad variety of methods devised for these problems. Among these methods, receding horizon control, or model predictive control (MPC), approaches have received notable attention due to their flexibility in encoding complex problem constraints and objectives. In MPC approaches, these multi-robot problems are formulated as optimization problems over a finite time duration at each time step. The resulting optimization problem is solved to obtain a sequence of control inputs over the specified time duration; however, only the initial control input is applied by each robot at the current time step. At the next time step, a new optimization problem is formulated, from which a new sequence of control inputs is computed to obtain a new control input for that time step. This process is repeated until completion of the task. At time t, the associated MPC optimization problem has the form

$$\begin{aligned} \underset{x,u}{\text{minimize}} \ & \sum_{i=1}^{N} f_i(x,u) \\ \text{subject to} \ & g(x,u) = 0 \\ & h(x,u) \le 0 \\ & x_{i,0} = \bar{x}_i \quad \forall i \in \mathcal{V} \end{aligned} \tag{29}$$

where $x_i \in \mathbb{R}^{n_i}$ denotes robot i's state trajectory, $u_i \in \mathbb{R}^{m_i}$ denotes its control input trajectory, and $x = [x_1^\top, x_2^\top, \dots, x_N^\top]^\top$, with $u = [u_1^\top, u_2^\top, \dots, u_N^\top]^\top$. The objective function of robot i, $f_i : \mathbb{R}^{\bar{n}} \times \mathbb{R}^{\bar{m}} \to \mathbb{R}$, is often quadratic, given by

$$f_i(x,u) = (x_i - \tilde{x}_i)^\top Q_i (x_i - \tilde{x}_i) + (u_i - \tilde{u}_i)^\top R_i (u_i - \tilde{u}_i) \tag{30}$$

where $\tilde{x}_i$ and $\tilde{u}_i$ denote the reference state and control input trajectory, respectively; $Q_i \in \mathbb{R}^{n_i \times n_i}$ and $R_i \in \mathbb{R}^{m_i \times m_i}$ denote the associated weight matrices for the terms in the objective function; $\bar{n} = \Sigma_{i=1}^{N} n_i$; and $\bar{m} = \Sigma_{i=1}^{N} m_i$. The dynamics function of the robots is encoded in $g : \mathbb{R}^{\bar{n}} \times \mathbb{R}^{\bar{m}} \to \mathbb{R}^{\bar{n}}$. Further, other equality constraints can be encoded in g. Inequality constraints, such as collision avoidance constraints and other state or control input feasibility constraints, are encoded in $h : \mathbb{R}^{\bar{n}} \times \mathbb{R}^{\bar{m}} \to \mathbb{R}^{l}$. In addition, the first state variable of each agent is constrained to be equal to its initial state, denoted by $\bar{x}_i$. In each instance of the MPC optimization problem, the initial state $\bar{x}_i$ of robot i is specified as its current state at that time step. Note that the MPC optimization problem in (29) is not generally separable, depending on the equality and inequality constraints. However, a separable form of the problem can always be obtained by introducing local copies of the optimization variables that are coupled in (29). The functions g and h can also encode complementarity constraints for manipulation and locomotion problems that involve making and breaking rigid body contact [42]. In the extreme case, where the optimization variables are coupled in the objective function and equality and inequality constraints in (29), a suitable reformulation takes the form

$$\begin{aligned} \underset{\hat{x},\hat{u}}{\text{minimize}} \ & \sum_{i=1}^{N} f_i(\hat{x}_i,\hat{u}_i) \\ \text{subject to} \ & g(\hat{x}_i,\hat{u}_i) = 0 \quad \forall i \in \mathcal{V} \\ & h(\hat{x}_i,\hat{u}_i) \le 0 \quad \forall i \in \mathcal{V} \\ & \phi_i(\hat{x}_i) = \bar{x}_i \quad \forall i \in \mathcal{V} \\ & \hat{x}_i = \hat{x}_j \quad \forall (i,j) \in \mathcal{E} \end{aligned} \tag{31}$$

where the function ϕ_i outputs the first state variable corresponding to robot i, given the input $\hat{x}_i$, which denotes robot i's local copy of x. Similarly, $\hat{u}_i$ denotes robot i's local copy of u, with $\hat{x} = [\hat{x}_1^\top, \hat{x}_2^\top, \dots, \hat{x}_N^\top]^\top$ and $\hat{u} = [\hat{u}_1^\top, \hat{u}_2^\top, \dots, \hat{u}_N^\top]^\top$. Distributed optimization algorithms [5], [43], [44] can be employed to solve the resulting MPC optimization problem in (31).

MULTI-ROBOT LEARNING

Multi-robot learning entails the application of deep learning methods to approximate functions from data to solve multi-robot tasks, such as object detection, visual place recognition, monocular depth estimation, 3D mapping, and multi-robot reinforcement learning. Consider a general multi-robot supervised learning problem where we aim to minimize a loss function over labeled data collected by all the robots. We can write this as

$$\min_{\theta} \sum_{i=1}^{N} \sum_{(x_{ij},y_{ij}) \in D_i} l(y_i, f(x_i;\theta))$$

where $l(\cdot,\cdot)$ is the loss function, (x_{ij},y_{ij}) is data point j collected by robot i with feature vector x_{ij} and label y_{ij}, D_i is the set of data collected by robot i, θ are the neural network weights, and $f(x;\theta)$ is the neural network parameterized function we desire to learn. By creating local copies of the

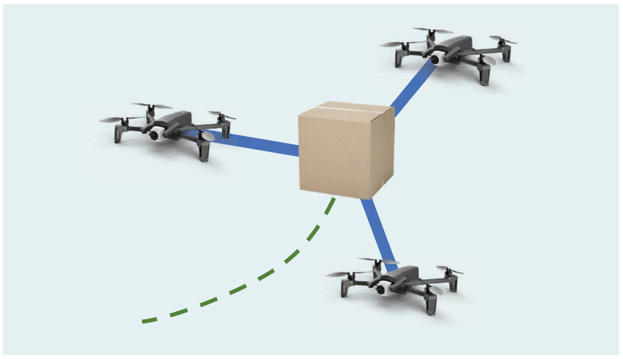
FIGURE 4. A multi-robot manipulation problem with three quadrotors collaboratively manipulating a load rigidly attached to each. The dashed line represents the reference trajectory for manipulating the load.

neural network weights θ_i and adding consensus constraints $\theta_i = \theta_j$, we can put the problem in the form of (6), so it is amenable to distributed optimization. We stress that this problem encompasses a large majority of problems in supervised learning. See [45] for an ADMM-based distributed optimization approach to solving this problem.

Beyond supervised learning, many multi-robot learning problems are formulated within the framework of reinforcement learning. In these problems, the robots learn a control policy by interacting with their environments by making sequential decisions. The underlying control policy, which drives these sequential decisions, is iteratively updated to optimize the performance of all the agents on a specified objective using the information gathered by each robot during its interaction with its environment. Figure 5 describes the reinforcement learning paradigm, where a group of robots learn from experience. Each robot takes an action and receives an observation (and a reward), which provides information on the performance of its current control policy in achieving its specified objective.

Reinforcement learning approaches can be broadly categorized into value-based methods and policy-based methods. Value-based methods seek to compute an estimate of the optimal action value function—the Q function—which represents the expected discounted reward when starting from a given state and taking a given action. An optimal policy can be extracted from the estimated Q function by selecting the action that maximizes the value of the Q function at a specified state. In deep value-based methods, deep neural networks are utilized in approximating the Q function. In contrast, policy-based methods seek to find an optimal policy by directly searching over the space of policies. In deep policy-based methods, the control policy is parameterized using deep neural networks. In general, the agents seek to maximize

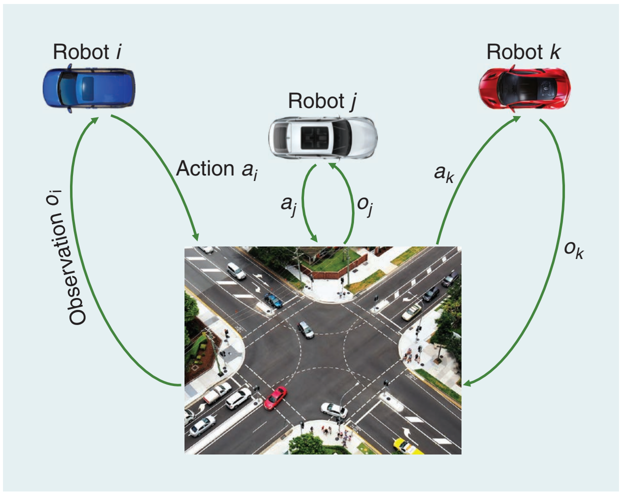

FIGURE 5. In multi-robot reinforcement learning problems, a group of robots compute a control policy from experience by making sequential decisions while interacting with the environment. Each robot takes an action and receives an observation (and a reward), which provides information on its performance in accomplishing a specified task.

the expected infinite-horizon discounted cumulative reward, which is posed as the optimization problem

$$\underset{\theta}{\text{maximize}}\ \mathbb{E}_{\pi_\theta}\left[\sum_{t\geq 0}\gamma^t \sum_{i=1}^{N} R_i(s_{i,t},a_{i,t}) \mid s_{i,0}=\bar{s}_i\right] \tag{32}$$

where π_θ denotes the control policy parameterized by θ, $\gamma \in \mathbb{R}$ denotes the discount factor [$\gamma \in (0,1)$], $s_{i,t}$ denotes the state of robot i at time t, $a_{i,t}$ denotes its action at time t, $\bar{s}_i$ denotes its initial state, $R_i : \mathcal{S}_i \times \mathcal{A}_i \to \mathbb{R}$ denotes the reward function of robot i, and N denotes the number of robots. The optimization problem in (32) is not separable in its current form. However, due to the linearity of the expectation operator, the optimization problem in (32) can be equivalently expressed as

$$\begin{aligned}&\underset{\hat{\theta}_1,\ldots,\hat{\theta}_N}{\text{maximize}} \sum_{i=1}^{N} \mathbb{E}_{\pi_{\hat{\theta}_i}}\left[\sum_{t\geq 0}\gamma^t R_i(s_{i,t},a_{i,t}) \mid s_{i,0}=\bar{s}_i\right]\\ &\text{subject to } \hat{\theta}_i = \hat{\theta}_j \quad \forall (i,j)\in\mathcal{E}\end{aligned} \tag{33}$$

which is separable among the N robots. Hence, the resulting problem can be readily solved using distributed optimization algorithms for reinforcement learning problems, such as distributed Q learning and distributed actor–critic methods [46], [47], [48].

NOTES ON IMPLEMENTATION, PRACTICAL PERFORMANCE, AND LIMITATIONS

Here, we highlight some relevant issues that arise in the application of distributed optimization algorithms in robotics problems. In Table 1, we highlight a few characteristics of the algorithms in each class of distributed optimization problems. We note that the properties of each algorithm class displayed in Table 1 are based on the representative algorithm considered in the algorithm class. We emphasize that subsequent research efforts have been devoted to the derivation of algorithms that address the practical issues faced by many of the existing algorithms. In this section, we describe alternative distributed algorithms that address these issues, often at the expense of convergence speed.

SELECTION OF A STOCHASTIC MATRIX

DFO algorithms and distributed sequential convex programming algorithms require the specification of a stochastic matrix, which must be compatible with the underlying communication network. In general, generating compatible row-stochastic and column-stochastic matrices for directed communication networks does not pose a significant challenge. To obtain a row-stochastic matrix, each robot assigns a weight to all its in neighbors such that the sum of all its weights equals one. Similarly, to obtain a column-stochastic matrix, each robot assigns a weight to all its out neighbors such that the sum of all its weights equals one. In contrast, generating doubly stochastic matrices for directed communication networks is nontrivial if each robot does not know the global network topology. Consequently, in general, algorithms that require doubly stochastic matrices are unsuitable for problems with directed communication networks.

A number of DFO algorithms allow for the specification of row-stochastic or column-stochastic matrices, making this class of algorithms appropriate for problems with directed communication networks, unlike distributed sequential convex programming algorithms, which generally require the specification of a doubly stochastic weighting matrix. Furthermore, a number of distributed sequential convex programming algorithms require symmetry of the doubly stochastic weighting matrix [20], [49], [50], [51], posing an even greater challenge in problems with directed networks.

The specific choice of a doubly stochastic weighing matrix may vary depending on the assumptions made on what global knowledge is available to the robots in the network. The problem of choosing an optimal weight matrix is discussed thoroughly in [52], in which the authors show that achieving the fastest possible consensus can be posed as a semidefinite program, which a computer with global knowledge of the network can solve efficiently. However, we cannot always assume that global knowledge of the network is available, especially in the case of a time-varying topology. In most cases, Metropolis weights facilitate fast mixing without requiring global knowledge, with the assumption that the communication network is undirected with bidirectional communication links. Each robot can generate its own weight vector after a single communication round with its neighbors. In fact, Metropolis weights perform only slightly suboptimally compared to centralized optimization-based methods [53]:

$$w_{ij} = \begin{cases} \dfrac{1}{\max\{|\mathcal{N}_i|, |\mathcal{N}_j|\}} & j \in \mathcal{N}_i \\ 1 - \sum\limits_{j' \in \mathcal{N}_i} w_{ij'} & i = j \\ 0 & \text{else} \end{cases}. \tag{34}$$

Distributed algorithms based on the ADMM do not require the specification of a stochastic weighting matrix. However, the C-ADMM [23] and other distributed variants assume that the communication network among all robots is bidirectional, which makes these algorithms unsuitable for problems with directed communication networks. A number of distributed ADMM algorithms for problems with directed communication networks have been developed [54], [55], [56]. Owing to the absence of bidirectional communication links among the robots, these algorithms utilize a dynamic average consensus scheme to update the slack variables at each iteration, which merges information from a robot and its neighbors using a stochastic weighting matrix. However, some of these distributed algorithms require the specification of a doubly stochastic weighting matrix [56], which introduces notable challenges in problems with directed communication networks, while others allow for the specification of a column-stochastic weighting matrix [55].

INITIALIZATION

In general, in convex problems, distributed optimization algorithms allow for an arbitrary initialization of the initial solution of each robot. However, these algorithms often place stringent requirements on the initialization of the algorithms' parameters. DFO methods require initialization of the step-size and often place conditions on the value of the step-size to guarantee convergence. Some distributed gradient tracking algorithms [14], [57] assume all robots use a common step-size, requiring coordination among all robots. Selecting a common step-size might involve the execution of a consensus procedure by all robots, with additional computation and communication overhead. In algorithms that utilize a fixed step-size, this procedure needs to be executed only once, at the beginning of the optimization algorithm. The ADMM and its distributed variants require the selection of a common penalty parameter ρ. Consequently, all robots must coordinate among themselves in selecting a value for ρ, introducing some challenges, particularly in problems where the convergence rate depends strongly on the value of ρ. Initialization of these algorithm-specific parameters has a significant impact on the performance of each algorithm.

In general, the performance of each distributed algorithm that we consider is sensitive to the choice of parameters, especially when local objective functions are poorly conditioned. For instance, in DFO methods, choosing α too large leads to divergence of the individual variables, while too small a value of α causes slow convergence. Similarly, the C-ADMM (Algorithm 3) has a convergence rate that is highly sensitive to the choice of ρ, though convergence is guaranteed for all $\rho > 0$. We study the sensitivity of the convergence rate to the parameter choice in each simulation in the "Distributed Multidrone Vehicle Tracking: A Case Study" section. However, the optimal parameter choice for a particular example is not prescriptive for the tuning of other implementations. The optimal step-size for a particular algorithm depends on many factors, including the network size, the network connectivity, and the underlying problem. For instance, the size of the network affects the value of the step-size that achieves optimal convergence as well as the maximum rate of convergence itself. Furthermore, while analytical results for optimal parameter selection are available for many of these algorithms, a practical parameter tuning procedure is useful if an implementation does not exactly adhere to the assumptions in the literature.

In the case that parameter tuning is essential to performance, it can be reasonable to select suitable parameters for an implementation before deploying a system, either using analytical results or simulation. The most general (centralized) procedure for parameter tuning involves comparing the convergence performance of the system on a known problem for different parameter values. While a uniform sweep of the parameter space may be effective for small problems or parameter-insensitive methods, it is not computationally efficient. Given the convergence rate of a distributed method at particular choices of parameter, bracketing methods provide parameter selections to more efficiently find the convergence rate-minimizing parameter. For instance, golden section search (GSS) provides a versatile approach for tuning a scalar parameter [58]. Finding the optimal step-size in one instance of a problem often provides reasonable parameter choices for a problem of similar size, connectivity, and structure.

DYNAMIC OR LOSSY COMMUNICATION

In practical situations, the communication network among robots changes over time as the robots move, giving rise to a time-varying communication graph. Networked robots in the real world can also suffer from dropped message packets as well as failed hardware or software components. Lossy communication can be a result of networks where there are many robots and communication signal interfere and in situations where robots have unstable communication links (e.g., wireless connections close to range limits). Generally, DFO optimization algorithms are amenable to problems with dynamic communication networks and are guaranteed to converge to the optimal solution provided that the communication graph is B connected for undirected communication graphs or B strongly connected for directed communication graphs [14], which implies that the union of the communication graphs over B consecutive time steps is connected or strongly connected, respectively. This property is also referred to as *bounded connectivity*. This assumption ensures the diffusion of information among all robots. Unlike DFO algorithms, many distributed sequential convex programming algorithms assume that the communication network remains static. Nevertheless, a few DSQP algorithms are amenable to problems with dynamic communication networks [16], [59] and converge to the optimal solution of the problem under the assumption that the sequence of communication graphs is B strongly connected. Some distributed ADMM algorithms are not amenable to problems with dynamic communication networks. This is an interesting avenue for future research.

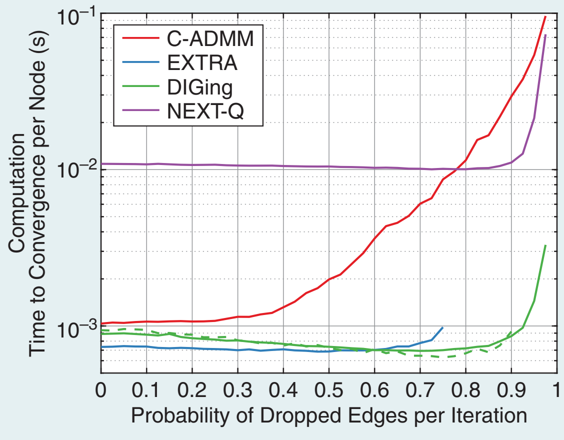

FIGURE 6. The computation time to convergence as a function of the probability of dropped edges in a mesh network, averaged over 50 trials using a geometric random graph with $N = 20$. The stopping condition for each trial is a normalized mean square error of 10^{-6}. Each undirected edge is dropped with the given probability at every iteration. DIGing is the only method considered that can handle directional lost edges (the dashed line). The implementations use optimal hyperparameters, which vary according to the probability of dropped edges.

Similarly, dropped messages or packets can be modeled as changes to edges in the communication graph where an edge temporarily becomes directed. In modern mesh networking protocols, dropped packets can be detected through packet acknowledgment, and the data can be resent or the robots can choose to ignore that communication link during the given iteration of distributed optimization. We explore the effect of dropping edges from the communication network in Figure 6.

SYNCHRONIZATION

Synchronization, in the context of distributed optimization, is the assumption that robots compute their local updates and communicate at the same time, and it ensures that each robot has up-to-date communicated variables from its neighbors. Many distributed optimization algorithms require synchronous execution for guaranteed convergence to an optimal solution [14], [16], [20], [23], [24], [26]. In practice, when networks have many agents or heterogeneous computation capability, it is unlikely that all robots will finish their local computation/communication at exactly the same time, and therefore, some practical synchronization scheme is required. Fortunately, one simple solution is to have each robot wait to receive updates from each of its neighbors before proceeding with its next iteration of distributed optimization. This is the decentralized version of a barrier algorithm [60] in parallel computing. When all robots require roughly the same amount of time to perform each iteration, this simple barrier approach has a negligible impact on the time to convergence of a distributed optimization algorithm. However, if some subset of the robots is much slower than the others, then this barrier approach can result in long idle times for some robots and a longer time to convergence.

Alternatively, DFO algorithms (DIGing, EXTRA, and so on) are generally fairly amenable to asynchronous execution, and some other methods are explicitly designed for asynchronous execution [61].

DISTRIBUTED MULTIDRONE VEHICLE TRACKING: A CASE STUDY

We illustrate the implementation of distributed optimization methods using a simulation of a multidrone vehicle target tracking problem as a case study. We emphasize that the same principles apply to a broad class of robotics problems that we have outlined in the "Multi-robot Problems Posed as Distributed Optimizations" section. In addition, we implement the C-ADMM distributed optimization algorithm on a network of Raspberry Pis communicating with XBee modules to demonstrate a distributed optimization algorithm on hardware.

SIMULATION STUDY

In this simulation, we consider a distributed multidrone vehicle target tracking problem in which robots connected by a communication graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, each record range-limited linear measurements of a moving target and seek to collectively estimate the target's entire trajectory. We assume that each drone can communicate locally with nearby drones over the undirected communication graph $\mathcal{G}$. The drones all share a linear model of the target's dynamics as

$$x_{t+1} = A_t x_t + w_t \tag{35}$$

where $x_t \in \mathbb{R}^4$ represents the position and velocity of the target in some global frame at time t, A_t is the dynamics matrix associated with a linear model of the target's dynamics, and $w_t \sim \mathcal{N}(0, Q_t)$ represents process noise (including the unknown control inputs to the target). Restricting our case study to a linear target model in this tutorial ensures that the underlying optimization problem is convex, leading to strong convergence guarantees and robust numerical properties for our algorithm. A more expressive nonlinear model can also be used, but this requires a more sophisticated distributed optimization algorithm with more challenging numerical properties. At every time step when the target is sufficiently close to a drone i (which we denote by $t \in \mathcal{T}_i$), that robot collects an observation according to the linear measurement model

$$y_{i,t} = C_{i,t} x_t + v_{i,t} \tag{36}$$

where $y_{i,t} \in \mathbb{R}^2$ is a positional measurement, $C_{i,t}$ is the measurement matrix of drone i, and $v_{i,t} \sim \mathcal{N}(0, R_{i,t})$ is measurement noise. We again assume a linear measurement model to keep this case study as simple as possible. A nonlinear model can also be used.

All the drones have the same model for the prior distribution of the initial state of the target $\mathcal{N}(\bar{x}_0, \bar{P}_0)$, where $\bar{x}_0 \in \mathbb{R}^4$ denotes the mean and $\bar{P}_0 \in \mathbb{R}^{4\times 4}$ denotes the covariance. The global cost function is of the form

$$f(x) = \left\| x_0 - \bar{x}_0 \right\|^2_{\bar{P}_0^{-1}} + \sum_{t=1}^{T-1} \left\| x_{t+1} - A_t x_t \right\|^2_{Q_t^{-1}} + \sum_{i \in \mathcal{V}} \sum_{t \in \mathcal{T}_i} \left\| y_{i,t} - C_{i,t} x_t \right\|^2_{R^{-1}} \tag{37}$$

while the local cost function for drone i is

$$f_i(x) = \frac{1}{N} \left\| x_0 - \bar{x}_0 \right\|^2_{\bar{P}_0^{-1}} + \sum_{t=1}^{T-1} \frac{1}{N} \left\| x_{t+1} - A_t x_t \right\|^2_{Q_t^{-1}} + \sum_{t \in \mathcal{T}_i} \left\| y_{i,t} - C_{i,t} x_t \right\|^2_{R^{-1}}. \tag{38}$$

In our results, we consider only a batch solution to the problem (finding the full trajectory of the target given each robot's full set of measurements). Methods for performing the estimate in real time through filtering and smoothing steps have been well studied, both in the centralized and distributed case [62]. An extended version of this multi-robot tracking problem is solved with distributed optimization in [3]. A rendering of a representative instance of this multi-robot tracking problem is shown in Figure 2.

In Figures 7 and 8, several distributed optimization algorithms are compared in an instance of the distributed multi-drone vehicle tracking problem. For this problem instance, 10 simulated drones seek to estimate the target's trajectory over 16 time steps, resulting in a decision variable dimension of $n = 64$. We compare four distributed optimization methods that we consider representative of the taxonomic classes outlined in the preceding sections: C-ADMM [23], EXTRA [24], DIGing [14], and NEXT-Q [16]. Figure 7 shows that the C-ADMM and EXTRA have similar fast convergence rates per iteration, while DIGing and NEXT-Q are four and 15 times slower, respectively, to converge below a mean square error of 10^{-6}. The step-size hyperparameters for each method are computed by GSS (for NEXT-Q, which uses a two-parameter decreasing step-size, we fix one according to the values recommended in [16]).

We note that tuning is essential for achieving robust and efficient convergence with most distributed optimization

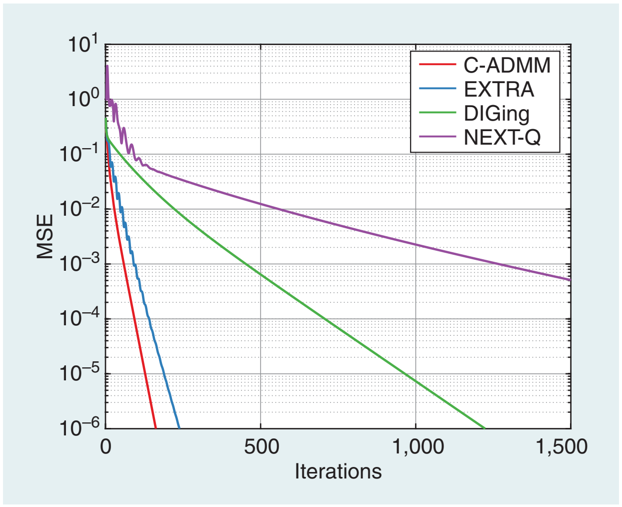

FIGURE 7. The mean square error (MSE) per iteration in a distributed multidrone vehicle target tracking problem with $N = 10$ and $n = 64$.

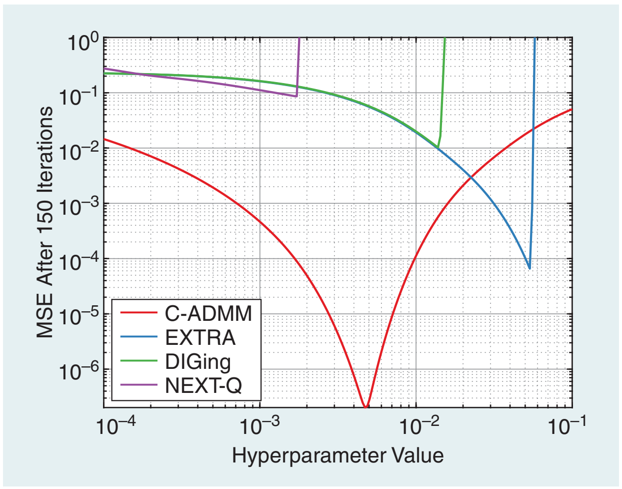

FIGURE 8. A hyperparameter sensitivity sweep for a distributed multidrone vehicle target tracking problem with $N = 20$ and $n = 64$. EXTRA, DIGing, and NEXT-Q diverge when their respective step-sizes are too large, while the C-ADMM converges over all choices of ρ. (C-ADMM values are reported with respect to $\rho/100$ in order to fit on the same axes as the other methods.)

algorithms. Figure 8 shows the sensitivity of these methods to variation in step-size and highlights that three of the methods (all except the C-ADMM) diverge for large step-sizes. In the case of EXTRA in this example, the optimal step-size is close in value to step-sizes that lead to divergence, posing a practical challenge for parameter tuning. While the C-ADMM seems to be the most effective algorithm in this problem instance, we note that other algorithms have properties that are advantageous in other instances of this problem or other problems. Furthermore, the optimal step-size depends on the problem structure. For instance, in this problem, as the number of agents increases, the optimal step-size decreases for the C-ADMM and increases for the other methods.

“BANDWIDTH LIMITATIONS HIGHLIGHT THE IMPORTANCE OF CONSIDERING LOW-DIMENSIONAL REPRESENTATIONS OF THE STATE OF THE PROBLEM AND/OR QUANTIZATION METHODS.”

As discussed in the “Dynamic or Lossy Communication” section, the convergence of distributed optimization algorithms may degrade under dynamic or lossy communication. In Figure 6, we demonstrate this effect given a geometric random graph with $N = 20$. For all four methods considered, a low probability of missing edges does not significantly degrade convergence compared to a static network. In particular, DIGing and NEXT-Q are robust to dropped edges, while EXTRA diverges for high rates of dropped edges, and the C-ADMM converges for carefully chosen values of ρ but at orders-of-magnitude increased computation time. While the C-ADMM converges in fewer iterations than the other methods in the examples of Figures 7 and 8, the dynamic graph topology in Figure 6 means that we cannot precompute matrix inverses, resulting in slower computation per iteration (the reported computation time is based on a MacBook Pro with an M1 Pro chip and 16 GB of unified memory). Of the methods considered, only DIGing handles directed dropped edges. While NEXT also addresses directed network communication, it requires a doubly stochastic matrix at each iteration. Fast distributed construction of doubly stochastic matrices is still an open question [63].

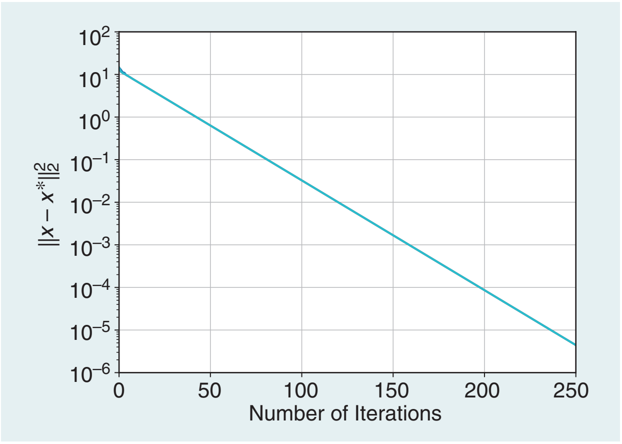

FIGURE 9. The convergence of the iterates computed by each robot, using the C-ADMM implemented on hardware, for the optimization problem with three robots in (39). The convergence errors of all the robots overlap in the figure.

HARDWARE IMPLEMENTATION

In this section, we discuss our implementation of the C-ADMM algorithm on hardware. Each robot is equipped with local computational resources and communication hardware necessary for peer-to-peer communication with neighboring robots. In the following discussion, we provide details of the hardware platform, the underlying communication network among robots, and the optimization problem considered in this section.

We consider the linear least-squares optimization problem

$$\min_{p} \sum_{i=1}^{N} (G_i p - z_i)^\top M_i (G_i p - z_i) \tag{39}$$

with the optimization variables $p \in \mathbb{R}^{32}$, $G_i \in \mathbb{R}^{m_i \times 32}$, $M_i \in \mathbb{R}^{m_i \times m_i}$, $z_i \in \mathbb{R}^{m_i}$, and $N = 3$ robots, where m_i depends on the number of measurements available to robot i. In this experiment, we have $m_1 = 3{,}268$, $m_2 = 5{,}422$, and $m_3 = 3{,}528$. We implement the C-ADMM to solve the problem, with a state size consisting of 32 floating-point variables.

The core communication infrastructure that we use consists of Digi XBee DigiMesh 2.4 radio-frequency mesh networking modules, which allow for peer-to-peer communication among robots. Local computation for each robot is performed using Raspberry Pi 4B single-board computers. The lower-level mesh network is managed by the DigiMesh software, and we interact with it through XBee Python Library.

We utilize the neighbor discovery application programming interface provided by Digi International to enable each robot to identify neighboring robots. This approach results in a fully connected communication network, considering the XBee radios have an indoor range of up to 90 m and an outdoor range of up to 1,500 m. The XBee modules used in our experiments have a maximum payload size of 92 B. However, the local variable of each robot in our experiment consists of 32 floating-point variables, which exceeds the maximum payload size that can be transmitted by the XBee radios at each broadcast round, presenting a communication challenge. To overcome this challenge, we break up the local variables into a series of packets of size 92 B and perform multiple broadcast rounds. The resulting implementation requires approximately 5.5 s per round of communication in the C-ADMM (i.e., for all the robots to exchange their decision variable information). In contrast, the Raspberry Pi computation for each iteration of the C-ADMM is approximately 15 μs, so the communication time is approximately five orders of magnitude slower than the computation time in our implementation. This slow communication speed is due to the severe bandwidth limitations of the XBee radios. We expect that an optimized implementation over a state-of-the-art 5-Gb/s WiFi or 5G network would reduce this communication time to about 0.2 μs per round.

As the C-ADMM is robust to wide range of penalty parameters (as in Figure 8), we set the penalty parameter in the C-ADMM to a value of 5 and do not perform a comprehensive search for the penalty parameter. In our experiments, this value of the penalty parameter provides suitable performance. In Figure 9, we provide the convergence error between the iterates of each robot and the global solution, which is obtained by aggregating the local data of all the robots and then computing the solution centrally. The convergence errors of all the robots' iterates overlap in the figure, with the error decreasing below 10^{-5} within 250 iterations, showing convergence of the local iterates of each robot to the optimal solution. Again, due the severe bandwidth limitations of the XBee radios, these 250 iterations correspond to approximately 23 min of wall clock time, of which approximately 99.97% was due to communication overhead. With a well-engineered 5-Gb/s Wi-Fi or 5G implementation, we expect this wall clock time for executing the 250 iterations of the C-ADMM in Figure 9 to take approximately 0.005 s.

This small-scale experiment reveals several of the important considerations in implementing distributed optimization algorithms using physical communication hardware. First, while synchrony is crucial for certain methods, including the C-ADMM, we can satisfy this requirement even on relatively simple equipment by using a barrier strategy. Second, bandwidth limitations highlight the importance of considering low-dimensional representations of the state of the problem and/or quantization methods. For instance, communicating the optimization variable requires fewer broadcast rounds than communicating the measurements in the example problem that we considered. Finally, tuning is an important consideration, and the C-ADMM provides a suitable solution due to its robustness to the choice of the ρ parameter.

CONCLUSION

In this tutorial, we have demonstrated that a number of canonical problems in multi-robot systems can be formulated and solved through the framework of distributed optimization. We have identified three broad classes of distributed optimization algorithms: DFO methods, distributed sequential convex programming methods, and the ADMM. Further, we have described the optimization techniques employed by the algorithms within each category, providing a representative algorithm for each category. In addition, we have demonstrated the application of distributed optimization in simulation on a distributed multidrone vehicle tracking problem and on hardware, showing the practical effectiveness of distributed optimization algorithms. However, important challenges remain in developing distributed algorithms for constrained nonconvex robotics problems and algorithms tailored to the limited computation and communication resources of robot platforms, which we discuss in greater detail in the second article in this series [7].

ACKNOWLEDGMENT

This project was funded in part by National Science Foundation (NSF) National Robotics Initiative Awards 1830402 and 1925030. Trevor Halsted was supported by a National Defense Science and Engineering Graduate Fellowship, and Javier Yu was supported on an NSF Graduate Research Fellowship. The authors would like to thank Siddharth Tanwar for implementing the C-ADMM multi-drone target tracking algorithm on XBee networking hardware.

AUTHORS

Ola Shorinwa, Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 USA. E-mail: shorinwa@stanford.edu.

Trevor Halsted, Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 USA. E-mail: halsted@stanford.edu.

Javier Yu, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA. E-mail: javieryu@stanford.edu.

Mac Schwager, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA. E-mail: schwager@stanford.edu.

REFERENCES

[1] R. T. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM J. Control Optim.*, vol. 14, no. 5, pp. 877–898, 1976, doi: 10.1137/0314056.

[2] J. N. Tsitsiklis, "Problems in decentralized decision making and computation," Cambridge Lab for Information and Decision Systems, Massachusetts Inst. of Technol., Cambridge, USA, Tech. Rep., 1984. [Online]. Available: https://www.mit.edu/~jnt/Papers/PhD-84-jnt.pdf

[3] O. Shorinwa, J. Yu, T. Halsted, A. Koufos, and M. Schwager, "Distributed multi-target tracking for autonomous vehicle fleets," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2020, pp. 3495–3501, doi: 10.1109/ICRA40945.2020.9197241.

[4] H.-T. Wai, Z. Yang, Z. Wang, and M. Hong, "Multi-agent reinforcement learning via double averaging primal-dual optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9649–9660.

[5] J. Bento, N. Derbinsky, J. Alonso-Mora, and J. S. Yedidia, "A message-passing algorithm for multi-agent trajectory planning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 521–529.

[6] L.-L. Ong, T. Bailey, H. Durrant-Whyte, and B. Upcroft, "Decentralised particle filtering for multiple target tracking in wireless sensor networks," in *Proc. 11th Int. Conf. Inf. Fusion*, 2008, pp. 1–8.

[7] O. Shorinwa, T. Halsted, J. Yu, and M. Schwager, "Distributed optimization methods for multi-robot systems: Part II—A survey," *IEEE Trans. Robot.*, vol. 39, no. 3, pp. 1781–1800, Jun. 2023. [Online]. Available: https://msl.stanford.edu/papers/shorinwa_distributed_2023-1.pdf

[8] A. Nedic and A. Ozdaglar, "On the rate of convergence of distributed subgradient methods for multi-agent optimization," in *Proc. IEEE Conf. Decis. Control*, 2007, pp. 4711–4716, doi: 10.1109/CDC.2007.4434693.

[9] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009, doi: 10.1109/TAC.2008.2009515.

[10] A. Nedic, A. Olshevsky, A. Ozdaglar, and J. N. Tsitsiklis, "On distributed averaging algorithms and quantization effects," *IEEE Trans. Autom. control*, vol. 54, no. 11, pp. 2506–2517, Nov. 2009, doi: 10.1109/TAC.2009.2031203.

[11] I. Lobel and A. Ozdaglar, "Distributed subgradient methods for convex optimization over random networks," *IEEE Trans. Autom. Control*, vol. 56, no. 6, pp. 1291–1306, Jun. 2011, doi: 10.1109/TAC.2010.2091295.

[12] A. I.-A. Chen, "Fast distributed first-order methods," Ph.D. dissertation, Massachusetts Inst. of Technol., Cambridge, USA, 2012.

[13] A. Nedić and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 601–615, Mar. 2015, doi: 10.1109/TAC.2014.2364096.

[14] A. Nedic, A. Olshevsky, and W. Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs," *SIAM J. Optim.*, vol. 27, no. 4, pp. 2597–2633, 2017, doi: 10.1137/16M1084316.

[15] M. Zhu and S. Martínez, “Discrete-time dynamic average consensus,” *Automatica*, vol. 46, no. 2, pp. 322–329, 2010, doi: 10.1016/j.automatica.2009.10.021.

[16] P. Di Lorenzo and G. Scutari, “NEXT: In-network nonconvex optimization,” *IEEE Trans. Signal Inf. Process. Netw.*, vol. 2, no. 2, pp. 120–136, Jun. 2016, doi: 10.1109/TSIPN.2016.2524588.

[17] J. E. Dennis Jr. and J. J. Moré, “Quasi-newton methods, motivation and theory,” *SIAM Rev.*, vol. 19, no. 1, pp. 46–89, 1977, doi: 10.1137/1019005.

[18] R. H. Byrd, H. F. Khalfan, and R. B. Schnabel, “Analysis of a symmetric rank-one trust region method,” *SIAM J. Optim.*, vol. 6, no. 4, pp. 1025–1039, 1996, doi: 10.1137/S1052623493252985.

[19] R. Tapia, “On secant updates for use in general constrained optimization,” *Math. Comput.*, vol. 51, no. 183, pp. 181–202, 1988, doi: 10.1090/S0025-5718-1988-0942149-3.

[20] A. Mokhtari, Q. Ling, and A. Ribeiro, “Network newton,” in *Proc. Conf. Rec. – Asilomar Conf. Signals, Syst. Comput.*, Apr. 2015, pp. 1621–1625, doi: 10.1109/ACSSC.2014.7094740.

[21] M. Eisen, A. Mokhtari, A. Ribeiro, and A. We, “Decentralized quasi-newton methods,” *IEEE Trans. Signal Process.*, vol. 65, no. 10, pp. 2613–2628, May 2017, doi: 10.1109/TSP.2017.2666776.

[22] H. Liu, J. Zhang, A. M.-C. So, and Q. Ling, “A communication-efficient decentralized newton’s method with provably faster convergence,” *IEEE Trans. Signal Inf. Process. Netw.*, vol. 9, pp. 427–441, Jul. 2023, doi: 10.1109/TSIPN.2023.3290397.

[23] G. Mateos, J. A. Bazerque, and G. B. Giannakis, “Distributed sparse linear regression,” *IEEE Trans. Signal Process.*, vol. 58, no. 10, pp. 5262–5276, Oct. 2010, doi: 10.1109/TSP.2010.2055862.

[24] W. Shi, Q. Ling, G. Wu, and W. Yin, “EXTRA: An exact first-order algorithm for decentralized consensus optimization,” *SIAM J. Optim.*, vol. 25, no. 2, pp. 944–966, 2015, doi: 10.1137/14096668X.

[25] Y. Tian, Y. Sun, B. Du, and G. Scutari, “ASY-SONATA: Achieving geometric convergence for distributed asynchronous optimization,” 2018, *arXiv:1803.10359*.

[26] O. Shorinwa, T. Halsted, and M. Schwager, “Scalable distributed optimization with separable variables in multi-agent networks,” in *Proc. Amer. Control Conf. (ACC)*, 2020, pp. 3619–3626, doi: 10.23919/ACC45564.2020.9147590.

[27] T. Cieslewski and D. Scaramuzza, “Efficient decentralized visual place recognition using a distributed inverted index,” *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 640–647, Apr. 2017, doi: 10.1109/LRA.2017.2650153.

[28] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part I,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, Jun. 2006, doi: 10.1109/MRA.2006.1638022.

[29] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): Part II,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 108–117, Sep. 2006, doi: 10.1109/MRA.2006.1678144.

[30] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based SLAM,” *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, Winter 2010, doi: 10.1109/MITS.2010.939925.

[31] A. Ahmad, G. D. Tipaldi, P. Lima, and W. Burgard, “Cooperative robot localization and target tracking based on least squares minimization,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5696–5701, doi: 10.1109/ICRA.2013.6631396.

[32] V.-L. Dang, B.-S. Le, T.-T. Bui, H.-T. Huynh, and C.-K. Pham, “A decentralized localization scheme for swarm robotics based on coordinate geometry and distributed gradient descent,” in *Proc. MATEC Web Conf.*, 2016, vol. 54, p. 02002, doi: 10.1051/matecconf/20165402002.

[33] N. A. Alwan and A. S. Mahmood, “Distributed gradient descent localization in wireless sensor networks,” *Arabian J. Sci. Eng.*, vol. 40, no. 3, pp. 893–899, 2015, doi: 10.1007/s13369-014-1552-2.

[34] M. Todescato, A. Carron, R. Carli, and L. Schenato, “Distributed localization from relative noisy measurements: A robust gradient based approach,” in *Proc. Eur. Control Conf. (ECC)*, 2015, pp. 1914–1919, doi: 10.1109/ECC.2015.7330818.

[35] R. Tron and R. Vidal, “Distributed 3-D localization of camera sensor networks from 2-D image measurements,” *IEEE Trans. Autom. Control*, vol. 59, no. 12, pp. 3325–3340, Dec. 2014, doi: 10.1109/TAC.2014.2351912.

[36] A. Sarlette and R. Sepulchre, “Consensus optimization on manifolds,” *SIAM J. Control Optim.*, vol. 48, no. 1, pp. 56–76, 2009, doi: 10.1137/060673400.

[37] K.-K. Oh and H.-S. Ahn, “Distributed formation control based on orientation alignment and position estimation,” *Int. J. Control, Autom. Syst.*, vol. 16, no. 3, pp. 1112–1119, 2018, doi: 10.1007/s12555-017-0280-2.

[38] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Res. Logistics Quart.*, vol. 2, nos. 1–2, pp. 83–97, 1955, doi: 10.1002/nav.3800020109.

[39] R. N. Haksar, O. Shorinwa, P. Washington, and M. Schwager, “Consensus-based ADMM for task assignment in multi-robot teams,” in *Proc. Int. Symp. Robot. Res.*, Cham, Switzerland: Springer-Verlag, 2019, pp. 35–51, doi: 10.1007/978-3-030-95459-8_3.

[40] L. Liu and D. A. Shell, “Optimal market-based multi-robot task allocation via strategic pricing,” *Robot., Sci. Syst.*, vol. 9, no. 1, 2013, pp. 33–40.

[41] S. Giordani, M. Lujak, and F. Martinelli, “A distributed algorithm for the multi-robot task allocation problem,” in *Proc. Int. Conf. Ind., Eng. Appl. Appl. Intell. Syst.*, Springer-Verlag, 2010, pp. 721–730, doi: 10.1007/978-3-642-13022-9_72.

[42] O. Shorinwa and M. Schwager, “Distributed contact-implicit trajectory optimization for collaborative manipulation,” in *Proc. Int. Symp. Multi-Robot Multi-Agent Syst. (MRS)*, 2021, pp. 56–65, doi: 10.1109/MRS50823.2021.9620665.

[43] L. Ferranti, R. R. Negenborn, T. Keviczky, and J. Alonso-Mora, “Coordination of multiple vessels via distributed nonlinear model predictive control,” in *Proc. Eur. Control Conf. (ECC)*, 2018, pp. 2523–2528, doi: 10.23919/ECC.2018.8550178.

[44] O. Shorinwa and M. Schwager, “Scalable collaborative manipulation with distributed trajectory planning,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2020, vol. 1, pp. 9108–9115, doi: 10.1109/IROS45743.2020.9340957.

[45] J. Yu, J. A. Vincent, and M. Schwager, “DiNNO: Distributed neural network optimization for multi-robot collaborative learning,” *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 1896–1903, Apr. 2022, doi: 10.1109/LRA.2022.3142402.

[46] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, “Fully decentralized multi-agent reinforcement learning with networked agents,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 5872–5881.

[47] Y. Zhang and M. M. Zavlanos, “Distributed off-policy actor-critic reinforcement learning with policy consensus,” in *Proc. IEEE 58th Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 4674–4679, doi: 10.1109/CDC40024.2019.9029969.

[48] A. OroojlooyJadid and D. Hajinezhad, “A review of cooperative multi-agent deep reinforcement learning,” 2019, *arXiv:1908.03963*.

[49] A. Mokhtari, W. Shi, Q. Ling, and A. Ribeiro, “A decentralized second-order method with exact linear convergence rate for consensus optimization,” *IEEE Trans. Signal Inf. Process. Netw.*, vol. 2, no. 4, pp. 507–522, Dec. 2016, doi: 10.1109/TSIPN.2016.2613678.

[50] M. Eisen, A. Mokhtari, and A. Ribeiro, “A primal-dual quasi-newton method for exact consensus optimization,” *IEEE Trans. Signal Process.*, vol. 67, no. 23, pp. 5983–5997, Dec. 2019, doi: 10.1109/TSP.2019.2951216.

[51] F. Mansoori and E. Wei, “A fast distributed asynchronous newton-based optimization algorithm,” *IEEE Trans. Autom. Control*, vol. 65, no. 7, pp. 2769–2784, Jul. 2020, doi: 10.1109/TAC.2019.2933607.

[52] L. Xiao and S. Boyd, “Fast linear iterations for distributed averaging,” *Syst. Control Lett.*, vol. 53, no. 1, pp. 65–78, Sep. 2004, doi: 10.1016/j.sysconle.2004.02.022.

[53] S. Jafarizadeh and A. Jamalipour, “Weight optimization for distributed average consensus algorithm in symmetric, CCS & KCS star networks,” 2010, *arXiv:1001.4278*.

[54] V. Khatana and M. V. Salapaka, “D-DistADMM: A O(1/k) distributed ADMM for distributed optimization in directed graph topologies,” in *Proc. 59th IEEE Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 2992–2997, doi: 10.1109/CDC42340.2020.9304086.

[55] V. Khatana and M. V. Salapaka, “DC-DistADMM: ADMM algorithm for constrained distributed optimization over directed graphs,” 2020, *arXiv:2003.13742*.

[56] K. Rokade and R. K. Kalaimani, “Distributed ADMM with linear updates over directed networks,” 2020, *arXiv:2010.10421*.

[57] G. Qu and N. Li, “Harnessing smoothness to accelerate distributed optimization,” *IEEE Trans. Control Netw. Syst.*, vol. 5, no. 3, pp. 1245–1260, Sep. 2018, doi: 10.1109/TCNS.2017.2698261.

[58] W. H. Press et al., *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge, U.K.: Cambridge Univ. Press, 1989.

[59] Y. Sun and G. Scutari, “Distributed nonconvex optimization for sparse representation,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 4044–4048, doi: 10.1109/ICASSP.2017.7952916.

[60] N. S. Arenstorf and H. F. Jordan, “Comparing barrier algorithms,” *Parallel Comput.*, vol. 12, no. 2, pp. 157–170, Nov. 1989, doi: 10.1016/0167-8191(89)90050-1.

[61] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 3043–3052.

[62] R. Olfati-Saber, “Distributed Kalman filtering for sensor networks,” in *Proc. 46th IEEE Conf. Decis. Control*, Piscataway, NJ, USA: IEEE Press, 2007, pp. 5492–5498, doi: 10.1109/CDC.2007.4434303.

[63] C. Xi, Q. Wu, and U. A. Khan, “On the distributed optimization over directed networks,” *Neurocomputing*, vol. 267, pp. 508–515, Dec. 2017, doi: 10.1016/j.neucom.2017.06.038.