A Distributed Framework for Integrated Task Allocation and Safe Coordination in Networked Multi-robot Systems

Andrea Miele[1], Martina Lippi[1], Andrea Gasparri[1]

***Abstract*—Deploying a team of autonomous robots, operating collaboratively towards a common objective within dynamic environments, has the potential to improve the system efficiency across several fields. This paper proposes a distributed comprehensive framework enabling a networked multi-robot system to serve time-varying requests arising from different locations within the environment in a distributed and safe manner, i.e., by guaranteeing no collisions with possible obstacles and preserving connectivity among the robots. To this aim, a two-layer architecture is proposed where the top layer is in charge of distributively assigning new service requests to the robots by resorting to an auction-based algorithm, while the bottom layer is in charge of safely navigating the environment to serve the assigned requests by relying on Control Barrier Functions. However, the presence of connectivity constraints might affect the number of service requests that the multi-robot system can handle simultaneously and might lead to deadlock situations where robots cannot reach the designated locations due to loss of network connectivity. Hence, a distributed strategy based on consensus algorithms to detect and solve deadlocks in a distributed fashion is proposed. The completeness of the approach is proved. Simulation results in an agricultural setting and real-world laboratory experiments are provided to validate the effectiveness of the proposed approach.**

***Note to Practitioners*—This paper was inspired by the necessity to coordinate a team of robots to perform tasks within an unstructured agricultural field, including both the decision-making and navigation strategies, with no central control unit as envisioned by the European project CANOPIES. To this aim, a distributed approach is designed where robots only rely on local data and information from neighboring robots to assign and execute tasks effectively in a coordinated manner. In addition, as working under local communication constraints may prevent parallel execution of all tasks, potentially leading to deadlock situations, a distributed strategy is developed to enable each robot to detect and solve such situations. The proposed approach can be employed in several domains where the cooperation of multiple autonomous robots might be beneficial, ranging from logistics settings to search and rescue scenarios up to agricultural environments. Laboratory experiments with three robots demonstrate the effectiveness of the approach.**

***Index Terms*—Cooperative robots, distributed control, task allocation, multi-robot systems**

I. Introduction

Coordinating multiple robots to effectively carry out diverse tasks in a real-world environment necessitates addressing several aspects. First, any time new tasks are requested to be executed, it is necessary to allocate them to the available robots according to predefined optimality criteria. Second, once the allocation is established, robots must be able to properly coordinate to reach the designated locations for performing the tasks, while ensuring safe navigation in dynamic environments. Third, in scenarios where simultaneous execution of all tasks is not feasible, a strategy must be established to prioritize them while ensuring that all tasks are ultimately executed. In addition, it is generally desirable to achieve this coordination in a distributed manner, i.e., with no central control unit, to promote system scalability, flexibility, and robustness [1]. This is particularly relevant in large-scale or remote environments, such as vineyards, orchards, or open fields, where centralized communication infrastructures may not be practical, or even unfeasible, due to factors like physical obstructions, extensive coverage requirements, or high deployment costs. However, distributed settings are generally significantly more challenging to control compared to centralized architectures. Indeed, in the latter case, a central control unit orchestrates all robots based on global knowledge of the system, while in the former case, each robot makes its own decisions by solely relying on local data, i.e., gathered from onboard sensors or neighboring robots. Hence, for effective coordination in networked distributed settings, it is also crucial to guarantee *connectivity* among robots at all times [2], ensuring that each robot can communicate with every other robot of the team.

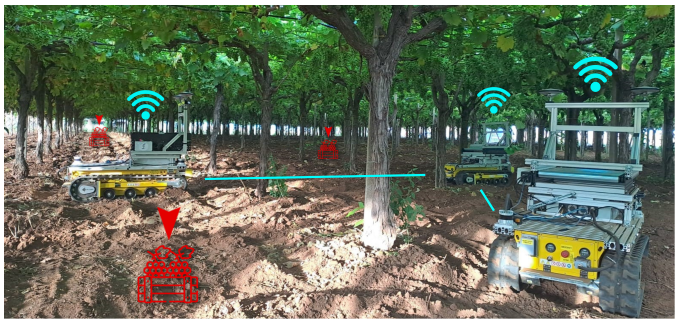


Fig. 1: Illustrative example of the operational scenario. Multiple robots communicate with each other and must perform services at scattered locations (red waypoints) across the field.

In this work, we propose a comprehensive distributed framework to achieve online task allocation and coordination of a networked multi-robot system in a safe manner, i.e., by ensuring no collision with dynamic obstacles, such as humans, in the environment and preserving network connectivity. A two-layer architecture is designed where the bottom layer addresses the low-level safe navigation by resorting to Control Barrier Functions (CBFs), while the top layer provides the distributed decision-making strategy involving the assignment of the tasks in a distributed manner and the management of possible deadlock situations due to connectivity constraints.

*This work was supported by the European Commission under the Grant agreement number 101016906 – Project CANOPIES

[1]A. Miele, M. Lippi, and A. Gasparri are with the Department of Civil, Computer Science, and Aeronautical Technologies Engineering, Roma Tre University, Italy

Auction-based and consensus-based algorithms are leveraged in this layer. Moreover, a global Finite State Machine (FSM), i.e., with full knowledge of the system, is designed and then distributed to the robot level, resulting in multiple local FSMs running in parallel. Inspired by the needs of the H2020 European Project CANOPIES[1], we consider the illustrative agricultural scenario shown in Figure 1, where robots are requested to empty crates filled by harvesters, which can be whether human or robotic, in a vineyard. Numerical simulations in a digital twin, including a comparative analysis with approaches at the state-of-the-art, and laboratory experiments are provided to validate the approach.

II. Related work

This section reviews the literature on task assignment in multi-agent systems, multi-robot coordination, and integrated solutions for online task assignment and coordination, and states the main contributions of this work.

Multi-agent task assignment. The field of multi-agent task assignment focuses on establishing the optimal assignment of tasks to available agents based on given criteria [3]. Early research in this field often relied on centralized approaches where a central unit assigns tasks to multiple agents based on global information. The Hungarian algorithm, introduced by Kuhn [4], is one of the earliest and widely studied methods for optimally matching tasks to agents while minimizing total cost through an iterative approach. It has been applied in multi-robot exploration [5] and pattern formation [6]. Extensions to distributed settings include [7] and [8]. In [7], robots autonomously execute sub-steps based on local information, with root robots responsible for starting the message exchange and synchronizing all robots. In [8], the root robots are removed, and robots cooperatively converge to an optimal assignment by exchanging solution estimates.

A further common method for task assignment has an economic foundation and is based on auctions [9]. Auction-based methods were introduced in a centralized version by D. P. Bertsekas [10] and are based on the concept of equilibrium. Briefly, the auction algorithm assigns prices to tasks and updates them through auctions to achieve optimal resource allocation or optimal element selection. This process continues until an optimal or satisfactory solution is achieved. Several works have explored distributed implementations of the auction-based method. First, shared memory was used in [11] to parallelize the problem. Then, the algorithm was extended to fully distributed settings in [12], where no shared memory or central auctioneer is required but only communication exchanges between neighboring robots are considered. A finite number of synchronous communication rounds is required for the algorithm to converge under the assumption that the number of agents (robots) is equal to or less than the number of tasks to be assigned. Additionally, a scenario with limited robot communication range is addressed in [13], while lossy communications are analyzed in [14].

Multi-robot motion coordination. A crucial aspect within collaborative multi-robot systems is how to coordinate the motion of the robots to achieve a common goal while ensuring safe navigation [15]. In this regard, one of the first approaches available in the literature is based on the concept of artificial potential fields [16]. Briefly, these represent virtual forces that can attract robots towards goals and can repel them from obstacles, other robots, or unsafe regions in general, thus facilitating safe and coordinated motion. Several coordination tasks have been solved by resorting to artificial potential fields. For instance, in [17], a flocking problem is investigated and the system stability under switching topologies is analyzed; in [18], a formation control problem is solved with a regular polygon shape, and the system asymptotic stability with no local minima is proved; in [19], the problem of limited field of views is taken into account while in [20], Linear Temporal Logic (LTL) specifications are included. Differently, the study in [21] uses spatial-temporal optimization techniques for swarm coordination in cluttered environments.

In recent years, the adoption of CBFs [22] has spread significantly in multi-robot coordination settings to ensure system safety. More in detail, CBFs are frequently utilized as constraints for optimization-based controllers to achieve minimum energy control or minimal input variation. For example, [23] uses CBFs for obstacle avoidance with limited actuation, [24] integrates CBFs into distributed model predictive control, [25] applies CBFs in adversarial situations, and [26] ensures human safety in collaborative settings using CBFs.

However, most of the above approaches assume connectivity of the communication network to achieve coordination. In this regard, several works focus on preserving connectivity by prioritizing connectivity constraints over task resolution. To mention a few, the study in [27] introduces a distributed swarm aggregation framework using attractive potential fields to maintain connectivity in both static and dynamic topologies, [28] employs a similar approach to maintain connectivity and avoid collisions, while [29] presents a decentralized control law for rendezvous and formation control in dynamic graphs by dynamically weighting edges. Connectivity based on algebraic connectivity is explored in [30], [2] using a potential-based approach and in [31], [32], [33] using CBFs. The above methods generally prioritize connectivity while supporting cooperative tasks in a best-effort manner. In real-world scenarios such as search and rescue or exploration, completing tasks within a limited timeframe is crucial, requiring both connectivity and task completion. However, this requires to possibly handle deadlocks arising from communication constraints. Most algorithms in this domain focus on exploration, guiding robots to reach scattered locations in the environment. For example, [34] avoids deadlocks by designating a prime traveler guaranteed to complete its task, while others proceed in a best-effort manner, sacrificing parallelization but preserving connectivity. Similarly, [35] uses recovery strategies like rendezvous techniques to resolve deadlocks. Other works, such as [36] and [37], use minimum spanning trees for connectivity. The former assumes fewer target points than robots, enabling parallel tasks but risking deadlocks. The latter allows intermittent disconnections to complete tasks but prevents dynamic task updates during disconnection. In conclusion, despite progress in multi-robot coordination, scalable solutions are

[1]https://canopies.inf.uniroma3.it

needed to balance connectivity, task completion, and efficiency in dynamic real-world settings.

Integrated online task assignment and coordination solutions. The task assignment and the coordination strategies are typically addressed in a decoupled manner. For instance, the framework in [34] assumes that tasks are assigned a priori to the robots. However, building frameworks that address both task assignment and multi-robot coordination can account for coupled aspects, such as potential interference between robots when reaching assigned tasks [38]. Furthermore, many task assignment approaches assume a fixed, known number of tasks, but in real-world scenarios, the task set is often unknown a priori and provided *online*. In these cases, the framework must continuously adjust task assignments and coordinated motion to ensure system safety. The work in [38] proposes a *centralized* comprehensive architecture for online task assignment and coordination, accounting for interference and preventing collisions among robots. A similar centralized approach is presented in [39], addressing a multi-agent pickup and delivery problem with a task assignment and path planning heuristic. In [40], a method for simultaneous task allocation and planning is proposed, using LTL to formalize tasks. An extension to uncertain settings with Markov Decision Processes (MDP) is discussed in [41]. For distributed architectures, a market-based solution is presented in [42] for formation control and obstacle avoidance, a factor graph optimization approach is proposed in [43] with max-sum algorithm for task allocation and conflict resolution, while a hierarchical method is presented in [44] where local planning handles deadlocks and generates velocity commands for non-holonomic robots in dense environments. Finally, [45] presents a Reinforcement Learning (RL) approach, where the high-level strategy is guided by feedback from the low-level navigation.

Although the previous distributed algorithms require communication among agents for proper operation, none address network connectivity maintenance. As a result, there is no guarantee that all agents will remain connected, potentially leading to isolation and hindering task completion. Connectivity constraints may conflict with task objectives, for example, when robots' assigned locations are far apart, risking network disconnection if all robots reach their destinations. Therefore, it is essential to develop *integrated* strategies that balance the need for network connectivity with task completion goals, ensuring a safe and efficient system. Clearly, designing these strategies in a distributed setting is significantly more challenging than in a centralized one.

Contributions. The main contributions of this article are:

1) A distributed *integrated* system is proposed to achieve *online* task allocation and coordination in networked multi-robot systems with safety guarantees, in terms of both obstacle avoidance and *connectivity maintenance*. Notably, the completeness of the approach is proven.
2) A distributed strategy is designed to detect and handle possible deadlocks arising from conflicts between connectivity maintenance constraints and robots' tasks.
3) A local finite state machine is designed for each robot to regulate its behavior and is proved to be equivalent to a global finite state machine where the behavior of the entire multi-robot team is regulated.
4) Extensive simulation, including a comparative analysis with respect to the frameworks in [34] and [35], and laboratory validation are provided using heterogeneous teams of robots in the context of precision agriculture.

Table I: Main notation of the paper.

Variable	Description
Problem	
n	Number of robots
m	Number of total requests
m_t	Number of requests currently handled
x	System state
u	Control input
λ_2	Algebraic connectivity
ε^λ	Minimum threshold for algebraic connectivity
δ^λ	Limit value $> \varepsilon^\lambda$ for algebraic connectivity
R	Maximum distance for communication
t_j^r	Arrival time of the request j
t_j^w	Waiting time of the request j
β_{ij}	Assignment benefit for agent i to perform task j
$\mathcal{X}$	Set of safe states
$\overline{\mathcal{X}}$	Set of states where connectivity is $\leq \delta^\lambda$
$\mathcal{R}$	Set of requests to serve
$\mathcal{P}$	Set of robot-request assignment pairs
$\mathcal{V}^a$	Set of active robots
$\mathcal{V}^s$	Set of robots performing the service operation
$\mathcal{V}^d$	Set of deadlocked robots
Global Finite State Machine	
A	Global state *Assignment*
E	Global state *Execution*
H	Global state *Homing*
Local Finite State Machine	
R_i	Local state *Reaching*
I_i	Local state *Idle*
S_i	Local state *Service*
D_i	Local state *Deadlock*
A_i	Local state *Assignment*
H_i	Local state *Homing*
Deadlock Management	
m_i^p	Robot i proposal for deadlock management
$\overline{m^p}$	Maximum proposal for deadlock management
Safety Filter	
h_i^c	CBF for connectivity maintenance
$h_i^{o_j}$	CBF for obstacle j avoidance

III. Preliminaries

Table I summarizes the main variables of the manuscript.

A. Agent and network modeling

Let us consider a networked multi-agent system composed of n agents with first-order dynamics, i.e., for each agent i it holds $\dot{x}_i = u_i$, where $x_i \in \mathbb{R}^p$ is the state vector and $u_i \in \mathbb{R}^p$ is the control input vector. The system aggregate dynamics is

$$\dot{x} = u, \tag{1}$$

where $x = [x_1^T, ..., x_n^T]^T \in \mathbb{R}^{np}$ and $u = [u_1^T, ..., u_n^T]^T \in \mathbb{R}^{np}$ represent the collective state and control input vectors, respectively. Note that the model in (1) can be used to design high-level control policies, which can be adapted to different robot dynamics using low-level controllers, e.g.,

feedback linearization for unicycle systems [46]. We assume a limited communication range R between agents, i.e., the communication between two agents i and j is possible if and only if $\|x_i - x_j\| < R$ with R a positive constant, and bidirectional communication links between agents. The network topology can be represented as an undirected time-varying graph $\mathcal{G}(t) = (\mathcal{V}, \mathcal{E}(t))$, where $\mathcal{V} = \{1, ..., n\}$ is the set of nodes and $\mathcal{E}(t) = \{(i, j) : i, j \in \mathcal{V},\ i \neq j\}$ is the set of edges at time t. In our setting, each node is associated with an agent, while each edge $(i, j) \in \mathcal{E}(t)$ is a communication link between agent pairs i and j at time t. As the communication links are assumed bidirectional, the presence of an edge (i, j) implies the presence of an edge (j, i), and hence they will be used in an interchangeable manner. We define the diameter of the graph as the maximum distance between any pair of nodes, i.e., the longest shortest path within the graph. We define $\mathcal{N}_i(t) = \{j \in \mathcal{V} : \|x_i - x_j\| < R\}$ as the set of neighbors of the agent i. The graph $\mathcal{G}(t)$ can be described using the adjacency matrix $A(t) \in \mathbb{R}^{n\times n}$ where the generic element $a_{ij}(t) = a_{ji}(t)$ denotes the presence of an edge between agents i and j at time t, i.e., if $j \in \mathcal{N}_i(t)$, then $a_{ij}(t) > 0$; otherwise, $a_{ij}(t) = 0$. Similar to [30], we use continuously differentiable edge weights $a_{ij}(t) \in A(t)$ defined as:

$$a_{ij}(t) = \begin{cases} \phi^l \left(e^{\frac{\left(R^2 - \|x_i - x_j\|^2\right)^2}{\sigma}} - 1 \right) & \text{if } \|x_i - x_j\| < R, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

with $\sigma = \frac{R^4}{log(2)}$ a normalization constant and $\phi^l > 0$ a tuning parameter. This formulation implies that, as two robots i and j move farther apart, the weight of the respective edge decreases until it reaches zero when they are no longer connected. Let $D(t) \in \mathbb{R}^{n\times n}$ be the degree matrix. This is a diagonal matrix $D(t) = \text{diag}\{d_1(t), ..., d_n(t)\}$ where each element $d_i(t)$ is given by the sum of the elements in the respective row (or, equivalently, column) i of $A(t)$, i.e., $d_i(t) = \sum_{j=1}^{n} a_{ij}(t)$. Based on the above matrices, it is possible to define the Laplacian matrix $L(t)$ associated with the graph $\mathcal{G}(t)$ as $L(t) = D(t) - A(t)$ [47]. The graph is defined as connected if there exists a path connecting any pairs of nodes. The algebraic connectivity of the time-varying undirected graph $\mathcal{G}(t)$ at time t is described by the second smallest eigenvalue $\lambda_2(x(t))$ of the Laplacian matrix. It is noteworthy that $\lambda_2(x(t)) > 0$ if and only if the graph is connected. Therefore, $\lambda_2(x(t))$ can be used as a metric to measure the connectivity of the network topology. We make the following assumption.

Assumption 1. *Each robot is aware of the number of robots in the team and the network starts in a connected configuration, i.e.,* $\lambda_2(x(0)) > 0$.

This assumption can be easily fulfilled by communicating the network size during an initialization phase, or by resorting to distributed algorithms for network size estimation, e.g. [48].

B. Control Barrier Functions

Control Barrier Functions (CBFs) represent a useful mathematical tool to ensure the fulfillment of safety constraints within a system [22]. Consider an affine control system

$$\dot{x} = f(x) + g(x)u, \tag{3}$$

where $x \in \mathbb{R}^r$ and $u \in \mathcal{U} \subset \mathbb{R}^q$ represent the system state and control input, respectively. Consider a safety constraint expressed as a super level set of the function $h(x) : \mathcal{H} \subset \mathbb{R}^r \to \mathbb{R}$, i.e., the set $\mathcal{C} = \{x \in \mathbb{R}^r : h(x) \geq 0\}$ defines the set of states in which the system is safe. The goal of the function $h(x)$ is to maintain the system (3) within this safe set $\mathcal{C}$, guaranteeing its forward invariance [22]. To ensure this, $h(x)$ must be a CBF, which means that an extended class-$\mathcal{K}$ function $\alpha : \mathbb{R} \to \mathbb{R}$ must exist such that

$$\sup_{u\in\mathcal{U}} \{L_f h(x) + L_g h(x) u\} \geq -\alpha(h(x)), \quad \forall x \in \mathcal{H}, \tag{4}$$

where L_f and L_g are the Lie derivatives of $h(x)$. We define $\mathcal{U}_{\text{cbf}}(x) = \{u \in \mathcal{U} : L_f h(x) + L_g h(x) u \geq -\alpha(h(x))\}$ as the set of control inputs ensuring that the set $\mathcal{C}$ remains forward invariant. Then, any control input $u(x) : \mathcal{H} \to \mathcal{U}$ can be applied provided that $u(x) \in \mathcal{U}_{\text{cbf}}(x)$. Since our model in (1) has single integrator dynamics, we consider $f(x) = O_{np} \in \mathbb{R}^{np\times np}$ and $g(x) = I_{np} \in \mathbb{R}^{np\times np}$ in (3), where O_{np} and I_{np} represent the zero and the identity matrix with dimension $np \times np$, respectively.

C. Auction-based distributed assignment

Given m tasks to realize and a multi-agent system with n agents, we define the benefit $\beta_{ij}\ \forall i \in \{1, ..., n\}, j \in \{1, .., m\}$ as the utility value associated with assigning agent i to task j. Let $\Omega : \{1, ..., n\} \to \{1, ..., m\}$ be an injective map such that $\Omega(i) = j$ if and only if task j is assigned to agent i [12]. The goal of the auction-based distributed assignment problem is to define the map Ω in a distributed manner guaranteeing that distinct agents are assigned to distinct tasks and maximizing the overall assignment benefit $\sum_{i=1}^{n} \beta_{i\,\Omega(i)}$.

The basic idea of auction-based algorithms is to utilize an iterative approach for comparing several bids and determining the most suitable one [10]. The final sales are assigned to the highest bidders. The algorithm ends once all agents are "happy", meaning that the set of prices is at equilibrium. Mathematically, this can be expressed as

$$\beta_{ij} - c_j(t) = \max_{k\in\{1,...,m\}} \{\beta_{ik} - c_k(t)\}, \quad \forall i \in \{1, ..., n\}, \tag{5}$$

where $c_j(t)$ is the cost (or price) of task j at time t and $\beta_{ij} - c_j(t)$ represents the net value of task j for agent i. However, an algorithm attempting to attain the above equilibrium may result in infinite loops [10]. To address this issue, (5) is modified as

$$\beta_{ij} - c_j(t) \geq \max_{k\in\{1,...,m\}} \{\beta_{ik} - c_k(t)\} - \varepsilon^a, \quad \forall i \in \{1, ..., n\}, \tag{6}$$

with ε^a a positive scalar providing a perturbation mechanism. The formulation in (6) ensures that each bid for a task must raise the price by a minimum amount, preventing bidding cycles with no increase in price. In a centralized setting, there is a central auctioneer, and the memory is shared by all agents, i.e., each agent has global knowledge of the system. In a distributed setting, computational and memory resources are

distributed among the agents that have limited communication capabilities. In this case, each agent locally stores the task price estimates, which may possibly be obsolete, and updates them using protocols of agreement with neighbors. The algorithm presented in [12] addresses the distributed problem for $n \leq m$ by relying on a price exchange protocol and a tie-breaking mechanism to compensate for the lack of global information. Briefly, at start each agent holds an assignment that provides the best net value based on a comparison of bids and prices. In each iteration, every agent follows a two-step process to update task prices and identify the best bidders using local protocols. First, each agent i receives prices and highest bidders from the neighbors in $\mathcal{N}_i$ and updates its own values accordingly. Next, if agent i is no longer the highest bidder on its assigned task, it places a new bid on the task with the best net value; otherwise, no action is taken. This two-step process ensures that all robots eventually possess updated request prices and corresponding highest bidders.

IV. Problem setting and formulation

A. Setting and notation

Let us consider a networked multi-robot system consisting of n robots with communication topology modeled as an undirected time-varying graph $\mathcal{G}(t)$. These robots are deployed within an environment where obstacles can be present and service requests can be made at any time. Each service request j is associated with a specific location $p_j \in \mathbb{R}^p$, which must be reached for servicing, with a finite processing time Δt_j^p, which represents the time required to actually carry out the service at the respective location, and with a request activation time t_j^r, that is the time instant when the request is made. We denote with t_j^w the waiting time of the request j at the current time t, i.e., $t_j^w = t - t_j^r$. A service request is said to be currently *handled* if a robot is either in the process of approaching the designated location or is executing the service operation at that location. The following assumption is made.

Assumption 2. *A single robot is needed to handle each service operation, and each robot can handle at most one service request at a time. Moreover, each service location is accessible, meaning that there are no permanent obstacles that prevent it from being reached.*

As mentioned in the Introduction, an example of a service request could be a box exchange request in a precision agriculture setting. More in detail, once a harvester has filled a box with harvested fruits, this has to be replaced with an empty one, and a service request is made. In order to fulfill this request, a robot must travel to the harvester location and execute the box exchange operation, dedicating the necessary time to complete the task. We denote with $\mathcal{R}(t)$ the set of all service requests that have been raised but are not yet fulfilled up to time t and with $m(t)$ the cardinality of the set. Note that this set can vary over time due to the arrival of new service requests and the accomplishment of existing ones. Without loss of generality, we consider that the index of each request is an integer in the range $[1, m(t)]$ and denotes its order of arrival, i.e., by considering two service requests j and l, if $j > l$ then $t_j^r \geq t_l^r$. We make the following assumption regarding the service requests.

Assumption 3. *Each robot is aware of the set of requests to serve as well as the respective locations and processing times.*

This also implies that robots are aware of new or completed service requests. The above assumption can be easily met by employing, for example, cloud-based architectures, e.g., [49], [50]. It is worth noticing that such a global communication of requests is only required sporadically, while we rely on local and continuous interactions only for coordinating the robots. Additionally, we make the following assumption.

Assumption 4. *The service operations are atomic, meaning that they are indivisible and uninterruptible.*

The above assumption is reasonable since routines such as exchanging boxes are not interruptible. However, to maximize the flexibility of the proposed framework, we enable the other phases to be interrupted, i.e., when a robot is approaching the assigned service location, a new assignment could happen, and a different service request might be assigned to the robot for optimality reasons.

Our main objective is to fulfill the time-varying service requests in a distributed manner by determining the assignment of the services to the available agents and subsequently defining the control strategy to reach the assigned service locations in a *safe* manner, i.e., avoiding obstacles and guaranteeing the connectivity of the network. Clearly, due to the limited agents and the safety constraints, it might occur that not all service requests in $\mathcal{R}(t)$ can be handled simultaneously. Hence, our objective also involves determining the maximum number of service requests that the multi-robot system can handle at time t, denoted by $m_t(t)$ with $m_t(t) \leq m(t)$ and $m_t(t) \leq n$. We say that a service request is *paused* if this is not handled at time t. To formalize the above, we introduce the following variables. From now on, we will omit time dependence for simplicity of notation, unless necessary. We define $\mathcal{P}$ as the set of robot-request assignment pairs that we aim to define given n robots, and m_t currently handled requests

$$\mathcal{P} = \{(i,j) \in \mathcal{R} \times \mathcal{V} : \quad j = \Omega(i)\}, \tag{7}$$

with $|\mathcal{P}| = \min\{n, m_t\}$. We indicate with $\mathcal{V}^a$ the set of robots that are currently handling a request, i.e., with an assigned request,

$$\mathcal{V}^a = \{i \in \mathcal{V} : \quad (i,j) \in \mathcal{P}\},$$

while its complement is denoted by $\overline{\mathcal{V}^a}$, i.e., $\overline{\mathcal{V}^a} = \mathcal{V} \setminus \mathcal{V}^a$. If a robot belongs to the set $\mathcal{V}^a$, it is said to be *active* or *assigned*. We introduce the subset of active robots that are performing a service operation denoted by $\mathcal{V}^s \subseteq \mathcal{V}^a$, i.e., $\mathcal{V}^s$ is composed of the robots that have reached the assigned position

$$\mathcal{V}^s = \{i \in \mathcal{V}^a : \quad \|x_i - p_j\| \leq \varepsilon^s, \quad (i,j) \in \mathcal{P}\},$$

with ε^s a small positive threshold. The set of active robots that are not serving is denoted by $\overline{\mathcal{V}^s}$, i.e., $\overline{\mathcal{V}^s} = \mathcal{V}^a \setminus \mathcal{V}^s$. The set of active robots that have completed the assigned task is denoted by $\mathcal{V}^c$, with $\mathcal{V}^c \subseteq \mathcal{V}^s$.

As far as the safety constraints are concerned, we consider that connectivity is preserved if $\lambda_2 > \varepsilon^\lambda$, with ε^λ a positive constant. We define the set of all possible states for which the system is safe, i.e., $\lambda_2 > \varepsilon^\lambda$ and there are no collisions, as $\mathcal{X} \subseteq \mathbb{R}^{np}$ and use the notation $\mathcal{X}_i \subseteq \mathbb{R}^p$ to indicate the set of possible safe states for robot i. Additionally, we introduce the set of states $\overline{\mathcal{X}} \subset \mathcal{X}$ for which the algebraic connectivity is equal or below a limit value $\delta^\lambda > \varepsilon^\lambda$, i.e., such that $\lambda_2 \leq \delta^\lambda$. We denote by $\overline{\mathcal{X}}_i \subset \mathcal{X}_i$ the set of states of robot i for which the system connectivity is equal or less than the threshold δ^λ and the only permissible control input for robot i is the zero vector. When the robot i is active and has zero velocity for safety reasons, i.e., $x_i \in \overline{\mathcal{X}}_i$, a local deadlock for the robot is said to be occurring, as detailed in Section V-C. We collect the robots with local deadlock condition in the set $\mathcal{V}^d$, i.e.,

$$\mathcal{V}^d = \{i \in \overline{\mathcal{V}^s} : \quad x_i \in \overline{\mathcal{X}}_i\}.$$

Finally, we can define the global deadlock condition.

Definition 1 (Global Deadlock)**.** *A global deadlock occurs when no active robot can move toward the assigned service location due to connectivity constraints.*

This situation occurs because the assigned requests are too far from each other and cannot be simultaneously reached due to connection limits. Based on the local deadlock condition, we can observe that if all robots with assigned requests that are not serving are in a *local deadlock*, the team is in a *global deadlock*, i.e., the deadlock condition is $i \in \mathcal{V}^d$, $\forall i \in \overline{\mathcal{V}^s}$. In the following, we refer to global deadlock simply as *deadlock*.

B. Problem statement

We can now formalize the problem addressed in this work.

Problem 1. *Given a networked multi-robot system composed of n robots, with network topology modeled as an undirected time-varying graph $\mathcal{G}(t)$, and a set $\mathcal{R}(t)$ of service requests, for which Assumptions 1-4 hold true, the objectives are to*

1) *Define the subset of $m_t(t)$ requests that can be simultaneously handled by the multi-robot system and establish their assignment in a distributed manner by maximizing the overall benefit $\sum_{(i,j)\in\mathcal{P}(t)} \beta_{ij}(t), \quad \forall t \geq 0$.*
2) *Guarantee that each request in $\mathcal{R}(t)$ is served by a robot in finite-time.*
3) *Guarantee the safety conditions of the system, i.e., each robot can navigate while avoiding obstacles and always keeping $\mathcal{G}(t)$ connected, i.e., $x \in \mathcal{X}(t)$, $\forall t \geq 0$.*

It is worth noticing that, as previously mentioned, connectivity constraints may impede the robots from reaching the designated locations, causing deadlocks in the system, i.e., situations where none of the robots can move and fulfill the assigned tasks. To overcome this issue, we propose a fully distributed framework to detect deadlocks and solve them by gradually reducing the number of requests to handle simultaneously, namely m_t, while ensuring that all service requests are accomplished in a finite time. An overview of the proposed framework is provided in the next section.

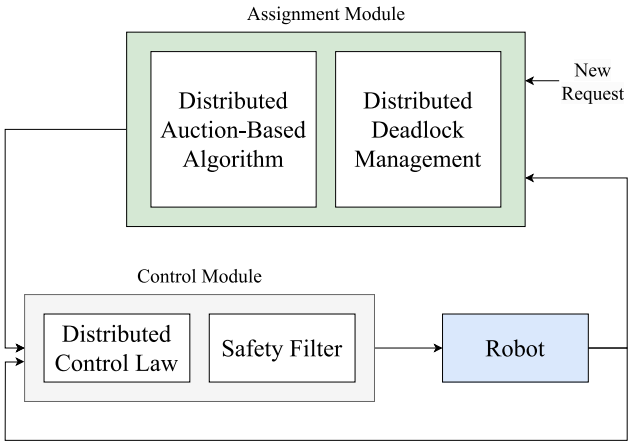


Fig. 2: Overview of the distributed framework architecture.

C. Solution overview

To solve Problem 1, a distributed architecture, as shown in Figure 2, is designed. It consists of two main modules for each robot. The first module, called *Assignment Module*, defines the high-level decision-making strategy, by defining the request assignment $\mathcal{P}$ as well as the number of currently handled requests. The second module, called *Control Module*, provides the low-level control strategy by enabling the robot to reach the assigned position while maintaining safety constraints. We emphasize that connectivity maintenance is an essential constraint for our approach, as it is required for dynamic distributed decision-making. More in detail, the *Assignment Module* is composed of the following two components:

- *Distributed Auction-Based Algorithm*, which determines the optimal request assignment in a distributed manner, taking into account the waiting times of the service requests as well as their closeness to the robots.
- *Distributed Deadlock Management*, which detects deadlock situations in a distributed manner and manages them by temporarily pausing specific service requests that will be subsequently reactivated.

The *Control Module* consists of the following components:

- *Distributed Control Law*, which enables each robot to reach the assigned positions, while actively contributing to connectivity maintenance.
- *Safety Filter*, which filters the output of the control law to ensure that robots meet safety constraints using CBFs. Specifically, connectivity maintenance and collision avoidance are taken into account.

Finally, a local Finite State Machine (FSM) is designed to regulate the behavior of each robot.

V. Finite State Machine

An incremental approach is taken to define the distributed behavior of the multi-robot system. First, we identify the fundamental logic of the system and define a simple Global Finite State Machine (GFSM) to illustrate the team-level behavior assuming complete state knowledge of all robots. Then, we derive a Local Finite State Machine (LFSM) for

each robot that facilitates a distributed architecture. Finally, an equivalence between the GFSM and the LFSMs is established.

A. Overview of Operational Logic

Before proceeding with modeling the FSMs, an overview of the operational logic is provided. In summary, the robot team's behavior is as follows: at each time step, the robots may receive service requests. Upon receiving a request, they solve an assignment problem to determine which robot should serve which request. At this point, the active robots, i.e., in the set $\mathcal{V}^a$, move to fulfill the requests while adhering to safety constraints. Robots without assigned requests, i.e., in the set $\overline{\mathcal{V}^a}$, are solely responsible for ensuring compliance with safety constraints. In the event of a deadlock, the robots attempt to resolve it before resuming movement. In particular, the number of simultaneously handled requests is decreased, enabling a previously assigned robot to serve as a bridge, preserving the network connectivity, and facilitating other robots to reach their designated positions. The team continues following this process until no deadlock condition occurs anymore.

After completing a service, the assignment problem is reevaluated and the number of requests to be handled m_t can increase, ensuring that $m_t \leq \min\{m, n\}$.

B. Global Finite State Machine

In the GFSM, we model the behavior of the robot team with global knowledge. This consists of three states.

- ***Assignment*** (A). In this state, the team solves the assignment problem detailed in Section VI and establishes the assignment set $\mathcal{P}$.
- ***Execution*** (E). This state represents the phase where robots are globally engaged in executing the respective tasks, such as moving to designated locations in a safe manner (i.e., for robots $i \in \mathcal{V}^a \wedge i \notin \mathcal{V}^s$), serving (i.e., for robots in the set $\mathcal{V}^s$), or acting as a bridge for maintaining connectivity (i.e., for robots in the set $\overline{\mathcal{V}^a}$). In case robots are serving, they will perform the service for the required processing time.
- ***Homing*** (H). This denotes the condition with no active requests ($m = |\mathcal{R}| = 0$).

In each state of the GFSM, it is necessary to ensure the system safety and functionality, i.e., that $x \in \mathcal{X}$, as detailed in Section VII. Each state takes a value in $\{0, 1\}$, where 1 indicates that it is active, 0 otherwise, and only one state is active at each time step, i.e., $\mathrm{A} + \mathrm{E} + \mathrm{H} = 1$. Let τ^- and τ^+ denote the instant immediately before and after a transition, respectively. Before defining the conditions for transitioning between states, we introduce the following auxiliary conditions, which, when triggered, can activate the automatic execution of predefined actions as detailed below. Each condition takes a value in $\{0, 1\}$, where 1 indicates that it is satisfied, 0 otherwise.

- *No requests:* indicating that there are no active requests, i.e., $m(\tau^-) = 0$.
- *Initial requests:* meaning that initial service requests are provided, i.e., $m(\tau^-) \neq 0$. The number of requests handled is set to $m_t(\tau^+) := \min\{m(\tau^-), n\}$, reflecting the constraint that the number of requests to handle cannot exceed the available robots or the total requests.
- *Assignment solved:* denoting that the assignment problem has been solved with $|\mathcal{P}(\tau^-)| = \min\{n, m_t(\tau^-)\}$.
- *Reassignment:* denoting that the assignment problem needs to be solved. This can occur in the following cases.
  - *New request:* a new service request $j \notin \mathcal{R}(\tau^-)$ is available, increasing the number of requests to handle to $m_t(\tau^+) = m_t(\tau^-) + 1$, provided there is capacity for it. Specifically, this happens if all requests available at τ^- are being handled, and there are robots not yet assigned to any request, i.e., $m_t(\tau^-) = m(\tau^-)$ and $m(\tau^-) < n$.
  - *Served request:* a service j is completed, i.e., $\exists\, i \in \mathcal{V}^c(\tau^-)$, $(i, j) \in \mathcal{P}$. This removes the request from $\mathcal{R}(\tau^+)$, decreasing $m(\tau^+) = m(\tau^-) - 1$. The number of requests to be handled is updated as $m_t(\tau^+) := \min\{m_t(\tau^-) + 1, m(\tau^+), n\}$.
  - *Global deadlock:* indicating that all the active robots, that are not serving, are in local deadlock, i.e., $i \in \mathcal{V}^d(\tau^-)$, $\forall i \in \overline{\mathcal{V}^s}(\tau^-)$, as per Definition 1. This triggers the number of requests to handle m_t to be decreased by one as a recovery strategy, i.e., $m_t(\tau^+) = m_t(\tau^-) - 1$.

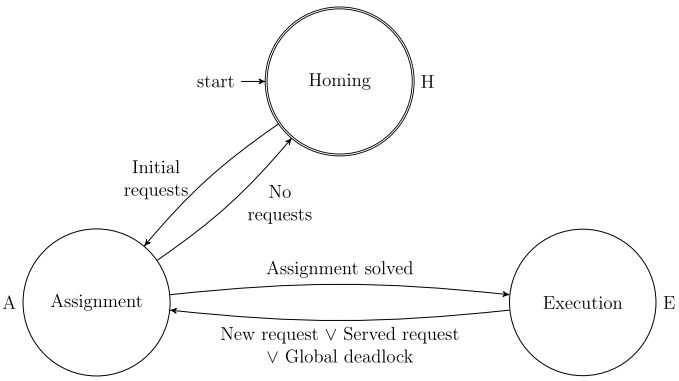


Fig. 3: GFSM describing the behavior of the system with global knowledge.

The robot team's functioning follows the FSM depicted in Figure 3. At the beginning, the system is in state H. Once at least a service request is made, i.e., *Initial requests* is triggered, the system transitions to state A, where the robots are assigned to serve these requests. After solving the assignment problem, i.e., $\mathcal{P}$ is obtained and *Assignment solved* is triggered, the system transitions to the execution state E, where it attempts to fulfill the requests. The system remains in E until one of the conditions *Served request*, *New request*, or *Global deadlock* is met. As previously described, with the occurrence of these conditions, the number of service requests to handle may vary accordingly. In all cases, the system returns to the assignment state A. The system transitions through the states A and E until the last service request is completed and there are no additional requests, i.e., *No requests* is triggered. In this case, the system returns to state H and waits for new requests.

By considering the global system behavior explained above,

the following conditions of state activation are obtained:

$$\begin{aligned} \mathrm{A}(\tau^+) &= \textit{Initial requests} \vee (\textit{New request} \vee \\ &\quad \vee \ \textit{Global deadlock} \vee \textit{Served request}) \\ &= \textit{Initial requests} \vee \textit{Reassignment}, \quad (8) \\ \mathrm{E}(\tau^+) &= \textit{Assignment solved}, \quad (9) \\ \mathrm{H}(\tau^+) &= \textit{No requests}. \quad (10) \end{aligned}$$

C. Local Finite State Machine

In the case of local FSM (LFSM), we consider that each robot i behaves according to its own FSM, denoted by LFSM_i. The execution of the n LFSMs in parallel yields the same behavior as the GFSM, as presented in Section V-D. Each LFSM_i consists of 6 states described below.

- ***Reaching*** (R_i). The robot has an assigned request ($i \in \mathcal{V}^a$) and is moving towards the assigned service location ($i \notin \mathcal{V}^s$).
- ***Idle*** (I_i). The robot does not have any assigned request ($i \in \overline{\mathcal{V}^a}$).
- ***Serving*** (S_i). The robot is currently performing the service operation ($i \in \mathcal{V}^s \wedge \ i \notin \mathcal{V}^c$).
- ***Deadlock*** (D_i). The robot is in local deadlock ($i \in \mathcal{V}^d$), i.e., it has an assigned request ($i \in \mathcal{V}^a$) but cannot reach the respective location ($i \notin \mathcal{V}^s$) due to connectivity constraints ($x_i \in \overline{\mathcal{X}}_i$).
- ***Assignment*** (A_i). The robot participates in solving the distributed assignment problem. In this phase, no motion of the robot is foreseen.
- ***Homing*** (H_i). There are no active requests ($m = 0$).

Each state takes value in $\{0, 1\}$ with only one active state per LFSM_i, i.e., $\mathrm{R}_i + \mathrm{I}_i + \mathrm{G}_i + \mathrm{D}_i + \mathrm{A}_i + \mathrm{H}_i = 1$. As per Section V-B, a list of conditions is introduced to identify the LFSM transitions with τ^- and τ^+ the time instant immediately before and after a transition, respectively. The conditions *Initial requests*, *Reassignment* (*New request*, *Served request* and *Global deadlock*) and *No requests* are as in Section V-B.

- *Assignment solved:* denoting that the distributed assignment problem is solved, according to the procedure in Section VI. This can occur in the following cases.
  - *Request not assigned:* the robot has no assigned request
  $$\big(|\mathcal{P}(\tau^-)| = min\{n, m_t(\tau^-)\}\big) \ \wedge \ \big(i \in \overline{\mathcal{V}^a}(\tau^-)\big).$$
  - *Request assigned:* the robot has an assigned request and was not previously serving a request
  $$\big(|\mathcal{P}(\tau^-)| = min\{n, m_t(\tau^-)\}\big) \ \wedge \ \big(i \in \overline{\mathcal{V}^s}(\tau^-)\big).$$
  - *Request reassigned:* the robot was previously serving and is reassigned to the same request
  $$\big(|\mathcal{P}(\tau^-)| = min\{n, m_t(\tau^-)\}\big) \ \wedge \ \big(i \in \mathcal{V}^s(\tau^-)\big).$$
- *Position reached*: denoting that the robot has reached the desired position for the service operation $i \in \mathcal{V}^s(\tau^-)$.
- *Local deadlock*: indicating that the robot is in local deadlock and can no longer move due to connectivity constraints, i.e., $i \in \mathcal{V}^d(\tau^-)$.

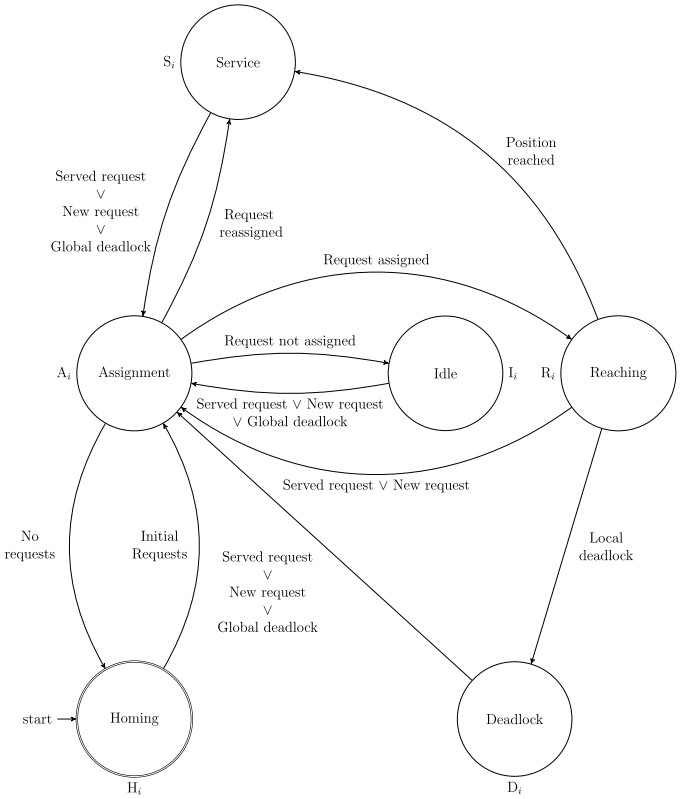


Fig. 4: LFSM_i associated with the robot i.

All the conditions considered for the LFSM can be locally detected, except for the *Global deadlock* and *Local deadlock* conditions, for which we resort to distributed estimators as detailed later in Sections VI-C and VII. The LFSM_i functioning is illustrated in Figure 4. The robot starts in the homing state H_i. Upon triggering the *Initial requests* condition, it transitions to the assignment state A_i for the assignment process. After solving the assignment problem, the robot behavior depends on whether it has an assigned request:

- If no request is assigned (*Request not assigned*), the robot enters the idle state I_i and acts as a communication bridge. It returns to A_i when any *Reassignment* condition is met.
- If a request is assigned (*Request assigned*), the robot enters the reaching state R_i to move to the service location. Then it transitions as follows:
  - To serving state S_i if the service location is reached (*Position reached*).
  - To local deadlock state D_i, if *Local deadlock* is triggered. The robot stays in D_i until a *Reassignment* condition triggers a return to A_i.
  - Back to A_i if a new request (*New request*) or a completed service (*Served request*) occur.
- If the robot is already serving a request, it is reassigned to this (*Request reassigned*) and enters the serving state S_i until a *Reassignment* condition is met.

Once all requests in $\mathcal{R}$ are served, and *No requests* is triggered, the robot returns to the homing state H_i.

By considering the behavior of the LFSM$_i$ explained above, the following conditions of state activation are obtained:

$$\begin{aligned}\mathrm{A}_i(\tau^+) &= \textit{Initial requests} \vee (\textit{New request} \\ &\quad \vee \textit{Global deadlock} \vee \textit{Served request}) \\ &= \textit{Initial requests} \vee \textit{Reassignment}, \end{aligned} \tag{11}$$

$$\mathrm{I}_i(\tau^+) = \textit{Request not assigned}, \tag{12}$$

$$\mathrm{R}_i(\tau^+) = \textit{Request assigned}, \tag{13}$$

$$\mathrm{S}_i(\tau^+) = \textit{Position reached} \vee \textit{Request reassigned}, \tag{14}$$

$$\mathrm{D}_i(\tau^+) = \textit{Local deadlock}, \tag{15}$$

$$\mathrm{H}_i(\tau^+) = \textit{No requests}. \tag{16}$$

D. Relationship Between GFSM and LFSMs

In the following, we first analyze the correspondence between the states of the GFSM and the ones of the n LFSMs. Then, we analyze the correspondence among the transitions.

- **Assignment equivalence.** The system is in the global state A when all robots are collaborating to the assignment process, i.e.,

$$\mathrm{A} = 1 \iff \mathrm{A}_i = 1, \forall i \in \mathcal{V}. \tag{17}$$

- **Execution equivalence.** The global state E describes the behavior of the robot team to accomplish the $m_t(t) > 0$ requests until re-assignment. Note that this state admits robots in local deadlock, as long as there exists at least one active robot reaching the assigned location. More specifically, the following equivalence holds

$$\begin{aligned}\mathrm{E} = 1 \iff &\big((\mathrm{R}_i = 1) \vee (\mathrm{I}_i = 1) \\ &\vee (\mathrm{D}_i = 1 \wedge \exists\, r : \mathrm{R}_r = 1), \forall i \notin \mathcal{V}^s\big) \\ &\wedge (\mathrm{S}_j = 1, \forall j \in \mathcal{V}^s). \end{aligned} \tag{18}$$

- **Homing equivalence.** If there are no requests, all robots are in H$_i$, so the team is in H, i.e.,

$$\mathrm{H} = 1 \iff \mathrm{H}_i = 1, \forall i \in \mathcal{V}. \tag{19}$$

Property 1. *The GFSM behaves equivalently to n LFSMs running in parallel.*

Proof. Based on the equivalence relations in (17)-(19), we show that, for each state of the GFSM, the same conditions trigger the transition to the global state and the equivalent local states, i.e., the GFSM and the n LFSMs are synchronized.

Initial and final state. The team initial state is H, which aligns with each individual robot initial state H$_\mathrm{i}$. Regarding the global final state H, we can observe that once the final request has been processed, triggering *Served request*, the global FSM enters the *Assignment* state, as well as all local FSMs enter the states $\mathrm{A}_i, \forall i$. Since there are no requests left, *No requests* is triggered (which is a global condition, as stated in Assumption 3), and both the global and local systems return to the respective *Homing* states, by virtue of (10) and (16), leading to (19).

Assignment. To reach the global *Assignment* state A, the system can either *i)* be in the *Homing* state H and the *Initial requests* condition is triggered or *ii)* be in the *Execution* state E and a *Reassignment* condition is triggered. In *i)*, in light of (19), this implies that all LFSMs are in the local *Homing* state and, since *Initial requests* is a global condition, they will all transition in the local *Assignment* state, fulfilling (17). Regarding *ii)*, if the GFSM is in the *Execution* state E, then the LFSMs can be either in *Idle*, *Reaching*, *Deadlock*, or *Service* states as reported in (18). For each robot i in I_i, D_i and S_i any *Reassignment* condition leads the robot to the state A_i, while for each robot i in R_i, this can transition to A_i if there are new requests or if a request is served. In this state, no *Global deadlock* condition can be triggered at a global level since this first requires the active robots (that are not serving) to reach a local deadlock state. Hence, the relation (17) is fulfilled under the same conditions for the global and local FSMs.

Execution. To enter the global *Execution* state E, it is necessary that the system is in the *Assignment* state A and the assignment problem is solved, triggering the condition *Assignment Solved*, as reported in (9). By virtue of (17), when the GFSM is in A, all LFSMs are in the local *Assignment* state as well. Hence, when the condition *Assignment Solved* is triggered, each robot i transitions to R_i, I_i or S_i depending on the assignment outcome, thus fulfilling (18). □

VI. Distributed assignment

In this section, we describe the components of the *Assignment Module* introduced in Section IV-C.

A. Assignment Problem Formulation

Let us first introduce the assignment problem formulation. Given n robots, m total service requests, and $m_t \leq \min\{m, n\}$ requests to handle, the assignment problem aims to determine the assignment set $\mathcal{P}$ in (7). In doing so, any robots already engaged in serving a specific request must remain assigned to that task. Let $\mathcal{P}_{prev}$ represent the previous assignment, with $\mathcal{P}_{prev} = \emptyset$ when the optimization problem is solved for the first time. We introduce the binary assignment variable $X_{ij} \in \{0, 1\}$, where $i \in \mathcal{V}$ and $j \in \mathcal{R}$, which is equal to 1 if robot i is assigned to request j, and 0 otherwise. Moreover, we define the benefit β_{ij} for the robot i to carry out the service j by taking into account the distance between the robot and the service location as well as the waiting time of the request and a maximum waiting time $T^w > 0$. Specifically, the benefit is defined as follows

$$\beta_{ij} = \beta_{ij}^d + \beta_j^t + \beta_j^p, \tag{20}$$

where $\beta_{ij}^d \in [0, 1]$ is a decreasing coefficient with respect to the distance between the robot i and the location of the request j, $\beta_j^t \in [0, 1]$ is a coefficient proportional to the waiting time of the request j, and $\beta_j^p \geq 0$ is a priority value activated when the maximum waiting time $T^w > 0$ is exceeded. In detail, the distance term β_{ij}^d is modeled as a sigmoid function, i.e.,

$$\beta_{ij}^d = \frac{1}{1 + e^{a(d_{ij} - b)}}, \tag{21}$$

with d_{ij} the distance between robot i and request j, and $a > 0$, $b \geq 0$ positive parameters. The waiting time term β_j^t is defined as the normalized waiting time of the request j

$$\beta_j^t = t_j^w / \|t^w\|, \tag{22}$$

where $t^w = [t_1^w \ ... \ t_m^w]^T \in \mathbb{R}^m$ is the vector collecting all the request waiting times. It should be noted that each robot is aware of t^w, as per Assumption 3. Finally, the priority term β_j^p is defined as follows

$$\beta_j^p = \begin{cases} m - j + 1 & \text{if } t_j^w > T^w, \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

The rationale behind β_j^p is that, if the request has been waiting longer than T^w, its assignment must be prioritized, taking into account the order of arrival. This guarantees the completeness of the assignment module as discussed in Section VI-D.

The following optimization problem is formulated:

$$\max_{X_{ij}, \forall i,j} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} \beta_{ij} X_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} X_{ij} \leq 1, \quad \forall \, i \in \mathcal{V} \tag{24a}$$

$$\sum_{i=1}^{n} X_{ij} \leq 1, \quad \forall \, j \in \mathcal{R} \tag{24b}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} X_{ij} = m_t, \tag{24c}$$

$$X_{ij} = 1, \quad \forall \, i,j : i \in \mathcal{V}^s \wedge (i,j) \in \mathcal{P}_{prev} \tag{24d}$$

$$X_{ij} \in \{0,1\}, \quad \forall \, i \in \mathcal{V}, \, j \in \mathcal{R}. \tag{24e}$$

Regarding the objective function, this aims to maximize the overall benefits related to robot-request matching. Regarding the constraints, the inequalities (24a)-(24b) ensure that each request is handled by at most one robot and that each robot handles at most one request, respectively. Constraint (24c) determines the number of assignments based on the value of m_t, with $m_t \leq \min\{m, n\}$. The equality in (24d) ensures that the serving robots are reassigned to their request. Constraint (24e) specifies that the variables X_{ij} are binary.

B. Assignment Problem Solution

As discussed in Section III-C, the work in [12] presented an algorithm to solve an assignment problem optimizing the overall benefit with n robots and m requests in a distributed fashion, under the condition that $n \leq m$. In this work, we modify the method of [12] to solve the assignment problem in (24) where we *i)* ensure an assignment also when $n > m$, *ii)* guarantee that serving robots are reassigned to their requests and *iii)* assign n robots to m_t requests, with $m_t \leq m$.

Based on the algorithm in [12], our procedure works as follows: first, we introduce variations in the algorithm in [12] to find an assignment for n robots and m requests, fulfilling points *i)* and *ii)*. Then, for point *iii)*, we use a distributed algorithm to pause service requests that cannot be currently handled. To address point *i)*, we simply introduce $n - m$ dummy requests, which allow us to recover the condition that the number of robots equals the total number of requests (including dummy requests). These dummy requests can be created by all robots since each of them is aware of the values of n under Assumption 1 and m under Assumption 3. Clearly, when a robot is assigned to a dummy request, no service is actually performed and the robot is in the *Idle* state.

To handle point *ii)* and reassign robots to the same requests that they are serving, these propose an infinitely high bid during the bidding phase to the respective requests. In this manner, it is not feasible for the others to provide a higher bid, ensuring correct reassignment.

Algorithm 1 Pausing requests selection

Require: β_i^*, q
1: $r_i(0) := \{(i, \beta_i^*), (\infty, \infty), ..., (\infty, \infty)\}$
2: **for** $k = 1, ..., n$ **do**
3: $\quad r^c := r_i(k-1)$
4: $\quad$ **for** $j \in \mathcal{N}_i$ **do**
5: $\quad\quad$ Gather $r_j(k-1)$
6: $\quad\quad r^c := r^c \cup r_j(k-1)$
7: $\quad$ **end for**
8: $\quad r_i(k) \leftarrow$ Get Min Ranking Values $\left(r^c, q\right)$
9: **end for**
10: **if** $(i, \beta^*) \in r_i$ **then**
11: $\quad$ Set Idle State i
12: **end if**

To address point *iii)* and handle the assignment in the case $m_t < \min\{m, n\}$, we propose an algorithm to pause the requests with the lowest benefits in a distributed manner. Let β_i^* be the benefit of the agent i executing the assigned task, i.e., $\beta_i^* = \beta_{ij}$, if $(i,j) \in \mathcal{P}$ and j is not a dummy task, and $\beta_i^* = \infty$ otherwise, and let q denote the number of requests to be paused such that

$$q = \begin{cases} n - m_t, & \text{if} \quad n \leq m, \\ m - m_t, & \text{if} \quad n > m. \end{cases}$$

We introduce the concept of *ranking* r, which is the set collecting the q tuples in the form (j, β_j^*), with $j \in \mathcal{V}$, having the lowest benefit values β_j^* in the network. For equal β_j^* values among robots, ascending order on the identifier values j is considered in r. Each robot i holds a local ranking variable r_i and the objective is to achieve convergence to r. To this aim, Algorithm 1 is executed by each robot i after the assignment protocol described above. This generalizes a static minimum consensus problem [51]. More specifically, at first, the ranking r_i is initialized with the local information (i, β_i^*) (line 1). Then, at each time step k, the set r^c collecting the ranking tuples from the neighbors and the robot itself is built (lines 3-7) and the q tuples with lowest benefit values are computed and stored as current ranking set r_i (line 8). This procedure is repeated n times to account for the worst-case scenario of communication graph diameter equal to n, as discussed in [51]. Finally, if the robot identifier falls in r_i, it goes into *Idle* state I_i and pauses the respective service request (lines 10-12).

C. Deadlock Management

The objective of the deadlock management strategy is to enable each robot to detect global deadlock situations in a distributed manner. Briefly, the proposed strategy is based on having each robot propose the number of requests to handle

based on the knowledge of the local state and, then, properly combine them using a dynamic maximum consensus protocol.

Let $m_i^p(k)$ represent the number of requests that the robot i proposes to handle defined as

$$m_i^p(k) = \begin{cases} m_t & \text{if } \mathrm{R}_i = 1, \\ m_t - 1 & \text{if } \mathrm{D}_i = 1, \\ -1 & \text{if } \mathrm{I}_i = 1 \text{ or } \mathrm{S}_i = 1, \end{cases} \tag{25}$$

where the value -1 denotes that the robot is in a state irrelevant to vary the number of requests to handle, as explained in the following. As long as at least one robot proposes m_t or all propose -1, the system is in the global *Execution* state, as described in equivalence (18), and no global deadlock is triggered. However, if no robot proposes m_t and at least one proposes $m_t - 1$, i.e.,

$$\overline{m}^p(k) = m_t - 1, \quad \text{with } \overline{m}^p(k) = \max_{i \in \mathcal{V}} \{m_i^p(k)\}, \tag{26}$$

it means that all active robots, that are not in service, are in a local deadlock. Hence, this condition denotes a global deadlock according to Definition 1. Based on these considerations, we resort to the Exact Dynamic Max-Consensus (EDMC) algorithm presented in [48] to estimate $\overline{m}^p$ and recognize if the condition (26) is met. Let $\widehat{m}_i(k) = [\widehat{m}_i^0(k) \ ... \ \widehat{m}_i^n(k)]^T \in \mathbb{R}^{n+1}$ be the state vector of the robot i for the EDMC. According to [48], the first element $\widehat{m}_i^0$ is set equal to the value proposed by the robot itself, i.e.,

$$\widehat{m}_i^0(k) = m_i^p(k), \tag{27}$$

while the remaining state variables are updated through a cascading process considering the neighboring robots as follows

$$\widehat{m}_i^l(k) = \max_{j \in \mathcal{N}_i \cup \{i\}} \{\widehat{m}_j^{l-1}(k-1)\}, \quad l = 1, ..., n. \tag{28}$$

The EDMC protocol is not relevant when the robots are performing the distributed assignment, i.e., $A_i = 1, \forall i \in \mathcal{V}$, and every time that the assignment phase is completed, the EDMC state variables are reset. Furthermore, note that, although in general the EDMC algorithm of [48] only provides *bounded* estimation error, in our setup we can prove that this error converges to zero as discussed in the following.

Theorem 1. *Consider a multi-robot system with each agent running the EDMC update law in* (27)-(28)*. Consider that Assumption 1 holds. Then, all agents reach a consensus on* $\overline{m}^p(k)$ *in maximum* n *steps.*

Proof. The proof is provided in Appendix A. □

D. Completeness of the Management System

In this section, we show the completeness of the assignment module, ensuring point *ii)* of Problem IV-B.

Property 2. *All requests are served in finite time.*

Proof. Let us first consider that the number of requests to serve does not increase and analyze the possible scenarios for each service request. When the requests are made (*Initial requests*) the following possible scenarios can happen for each request: *i)* it is immediately assigned, the robot reaches the location and performs the service without deadlocks, *ii)* it is immediately assigned, but the robot cannot reach the location due to safety constraints, entering a local *Deadlock* state, and the request is paused, or *iii)* it is not assigned and is queued.

We need to ensure that, even if a request is paused or queued, this will be assigned, and then served, in finite time. In this regard, let us analyze the worst-case scenario, where, due to safety constraints, multiple *Global deadlocks* occur and m_t reaches the minimum value equal to $m_t = 1$, with $m > 1$ and $n > 1$. This situation implies that all robots i are in the *Idle* state I_i except for one robot l that is either in the *Reaching*, i.e., $\mathrm{R}_l = 1$, or in the *Serving* state, i.e., $\mathrm{S}_l = 1$. In this case, all robots in *Idle* state will possibly move, if needed, to preserve the safety constraints and will ensure that the active robot can reach the assigned location (as demonstrated in Theorem 2) and accomplish the service request. Once the service is completed, the condition *Served request* is triggered, and the system executes the assignment procedure with $m_t = 2$. At this point, two requests are assigned and at least one will be served. By iterating this reasoning, all requests will be served in finite time. Note that the cases $n = 1$ or $m = 1$ are trivially solved, as they lead to the previously discussed case $m_t = 1$.

At this point, let us study the case where new requests arise during execution (*New request*). The number of requests increases if there is available capacity, i.e., if all previous requests are being handled ($m_t(\tau^-) = m(\tau^-)$) and some robots are idle ($m(\tau^-) < n$). This ensures that if m_t was decreased due to global deadlocks, it is not increased afterward. As done previously, let us analyze the worst-case scenario, where $m_t = 1$ due to multiple global deadlocks. In this case, if a new request occurs, the assignment procedure is executed with an unchanged number of requests to handle m_t. This might generally result in a different request being assigned if it has a higher benefit compared to the current one. However, the priority term in (23) ensures that, if there are requests waiting for more than T^w, these will have higher benefit values than those with shorter waiting times and will be sorted by their order of arrival. To prove this, let us consider two requests j and l with $j < l$ and $t_j^w > T^w$. In view of (20) and (23), the following inequalities hold for each robot i $\beta_{ij} \geq \beta_j^t + 1 + m - j$ and $\beta_{il} \leq 1 + \beta_l^t + 1 + m - l$. Thus, to prove that $\beta_{ij} > \beta_{il}$, we can show that $\beta_j^t + 1 + m - j > 1 + \beta_l^t + 1 + m - l$, which is fulfilled considering that $\beta_j^t > \beta_l^t$ and $j < l$. The above implies that, after a maximum waiting time T^w, the requests will be prioritized with respect to their order and will be served in finite time. □

VII. Distributed control

As stated in Section IV-C, the *Control Module* comprises two components: the distributed control law and the safety filter. The first ensures that each robot performs its task, i.e., reaching the assigned position for active robots or providing proactivity to maintain the connection for idle robots. The safety filter is cascaded to the controller and designed to minimize alterations to the control input in order to maintain safety constraints, i.e., connectivity and obstacle avoidance. Figure 5 shows the control module architecture.

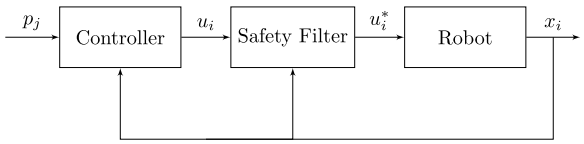


Fig. 5: Block diagram for robot i control module.

A. Distributed controller

By resorting to the control law proposed in [30], two contributions are considered for each robot i: one for maintaining the connection, u_i^c, and the other for achieving a secondary goal, such as reaching the assigned location, u_i^p, i.e.,

$$u_i = u_i^p + u_i^c, \tag{29}$$

where $\|u_i^p\| \leq U^p$ and $\|u_i^c\| \leq U^c$, with $U^p + U^c = U \in \mathbb{R}$ representing physical limitations on the robot actuators. Given the collective dynamics in (1), we obtain $\dot{x} = u^p + u^c$, where $u^p = [{u_1^p}^T \ ... \ {u_n^p}^T]^T$ and $u^c = \left[{u_1^c}^T \ ... \ {u_n^c}^T\right]^T$ are the stacked vectors of the terms for reaching the desired positions and for connectivity maintenance, respectively. Below, we provide a description and analysis of the terms u_i^p and u_i^c. Clearly, these terms are only relevant when the team is in the execution global state, while no motion is foreseen if the robots are in the homing or assignment states, i.e., if $H_i = 1$ or $A_i = 1$ it holds $u_i^p = u_i^c = 0$, for all $i \in \mathcal{V}$.

Reaching desired positions. We distinguish the expression of u_i^p depending on the robot state. If i is an active robot, i.e., R_i, S_i or D_i is 1, with j being the assigned request, a proportional controller is employed to reach the position p_j, i.e.,

$$u_i^p = -\kappa^p \frac{x_i - p_j}{\|x_i - p_j\|}, \tag{30}$$

where $\kappa^p \in (0, U^p]$ is a positive gain. Instead, if the robot is in the idle state ($I_i = 1$), no request is assigned. Thus, no location has to be reached and u_i^p is set to zero.

Connectivity maintenance. Let $\nabla_{x_i}\lambda_2(x)$ be the gradient of $\lambda_2(x)$ with respect to x_i which, in view of [52], is equal to

$$\nabla_{x_i}\lambda_2(x) = \sum_{j \in \mathcal{N}_i(t)} \nabla_{x_i} a_{ij}(t)(\nu_{2_i} - \nu_{2_j})^2, \tag{31}$$

where ν_{2_i} and ν_{2_j} are the i-th and j-th component of the eigenvector ν_2 associated to λ_2, respectively. The general term of the robot i for the connectivity maintenance is

$$u_i^c = \begin{cases} \kappa^c \, e^{\frac{\varepsilon^\lambda - \lambda_2(x)}{\kappa^e}} \frac{\nabla_{x_i}\lambda_2(x)}{\|\nabla_{x_i}\lambda_2\|}, & \text{if } \|\nabla_{x_i}\lambda_2(x)\| \neq 0 \wedge \mathrm{S}_i = 0, \\ 0, & \text{otherwise,} \end{cases} \tag{32}$$

where $\varepsilon^\lambda > 0$ is the desired lower-bound for the algebraic connectivity $\lambda_2(x)$, while $\kappa^e > 0$ and $\kappa^c \in (0, U^c]$ are parameters to tune, with $\kappa^c > \kappa^p$. Let us define the threshold δ^λ introduced in Section IV-A as

$$\delta^\lambda = \varepsilon^\lambda + \kappa^e \log\left(\frac{\kappa^c}{\kappa^p}\right). \tag{33}$$

Briefly, when the system is in the global execution state, the control law (29) operates as follows for each active robot i: when λ_2 is far from ε^λ, u_i^p dominates over u_i^c, enabling the robot to reach its assigned position. As λ_2 approaches ε^λ and falls below δ^λ ([30, Theorem 5]), u_i^c grows to dominate u_i^p, ensuring connectivity by maintaining $\lambda_2 > \varepsilon^\lambda$. Regarding the idle robots, since the respective term u_i^p is zero, the connectivity component u_i^c always dominates over u_i^p. This results in the robot actively moving to preserve connectivity and increase λ_2. Note that this active motion would also occur when all other active robots are serving, even though there is no need to increase connectivity in this situation. Thus, to prevent unnecessary motion, the distributed consensus described in Section VI-C is used to detect this case, and, if the robot is in *Idle* state but it holds $\hat{m}_i^n = -1$, the connectivity input u_i^c is set to zero to prevent motion. Finally, note that the connectivity input u_i^c is zero when the robot is in the serving state S_i to ensure completion of the atomic service operation.

Since we realize a distributed architecture, λ_2 and ν_2 must be estimated locally. As in [30], methods like [52] enable distributed estimation with convergence of the estimation error to zero. Hence, let $\hat{\lambda}_{2_i}$ and $\hat{\nu}_2^i$ represent the locally estimated λ_2 and ν_2 by robot i, respectively, the control law remains unchanged, but $\hat{\lambda}_{2_i}$ and $\hat{\nu}_2^i$ replace λ_2 and ν_2 in (31)-(32).

In [30, Theorem 6], conditions for reaching assigned positions with connectivity constraints are given but may not necessarily be fulfilled in any operational setting. Hence, we formally prove that the control law in (29) ensures position reachability even in worst-case scenario for the system completeness (see Property 2), thus corroborating this property of the proposed framework.

Theorem 2. *Consider a multi-robot system with each agent behaving according to the LFSM described in Section V-C and with control input* (29)*. Consider that Assumptions 1 and 2 hold and assume that only one agent r is in the reaching state R_r, while the others are in the idle state, i.e., $\mathrm{I}_k = 1$, $\forall k \in \mathcal{V}$ with $k \neq r$. Then, the agent r is guaranteed to reach its desired position j, with $(r, j) \in \mathcal{P}$.*

Proof. The proof is provided in Appendix B. □

B. Safety Filter

The objective of the safety filter is to minimize the modifications to u_i obtained from (29), while fulfilling safety requirements of collision avoidance and connectivity maintenance. Let $u_i^* \in \mathbb{R}^p$ be the filtered control input for the robot i, satisfying the safety constraints. To compute it, as introduced in III-B, we resort to CBFs and formulate the following Quadratic Programming (QP) problem:

$$\begin{aligned} \min_{u_i^*} \quad & \frac{1}{2}\|u_i^* - u_i\|^2 \\ \text{s.t.} \quad & \nabla_{x_i} h_i^{o_j}(x) u_i^* \geq -\alpha^o(h_i^{o_j}(x)), \quad \forall \text{ obstacle } j, \quad (34a) \\ & \nabla_{x_i} h_i^c(x) u_i^* \geq -\alpha^c(h_i^c(x)), \quad (34b) \end{aligned}$$

where $h_i^{o_j}(\cdot)$ represents the CBF to avoid the obstacle j and $\alpha^o(\cdot)$ is the respective extended class-$\mathcal{K}$ function, while $h_i^c(\cdot)$ represents the CBF for connectivity maintenance, with $\alpha^c(\cdot)$ is the respective extended class-$\mathcal{K}$ function. These functions are detailed in the following.

Obstacle avoidance. We approximate each obstacle j, representing, for instance, a human operator, as a circle/sphere with center o_j and radius $\rho_j^o > 0$, while for the robots we consider that they can be approximated either with circular/spherical or rectangular/parallelepiped shape.

In the case of circular (or spherical) shape for the robot i, with center x_i and radius ρ_i^r, the CBF function proposed in [22] can be leveraged, i.e., $h_i^{o_j} = \|x_i - o_j\|^2 - (\rho_i^r + \rho_j^o)^2 \geq 0$. This implies that the distance between the centers of robot i and obstacle j must be at least equal to the sum of the respective radii. We define the class-$\mathcal{K}$ function $\alpha^o(h_i^{o_j}(x)) = \phi^o h_i^{o_j}(x)$, with ϕ^o a positive value. Thus, the constraint (34a) for avoiding the obstacle j is

$$2(x_i - o_j)^T u_i^* \geq -\phi^o \left(\|x_i - o_j\|^2 - (\rho_i^r + \rho_j^o)^2\right). \quad (35)$$

In the case of rectangular shape of the robot i, we can identify it with four segments describing its sides, denoted as $L_{i,1}$, $L_{i,2}$, $L_{i,3}$, and $L_{i,4}$. The points of any segment $L_{i,k}$ can be expressed as a convex combination of the segment endpoints, i.e., $l_{i,k}(\theta_{i,k}) = l_{i,k}^A(1 - \theta_{i,k}) + l_{i,k}^B \theta_{i,k}$, with $l_{i,k}^A$ and $l_{i,k}^B$ the endpoints and $\theta_{i,k} \in [0, 1]$ the combination weight. Let $\theta_{i,k}^*$ denote the weight associated with the point on the segment $L_{i,k}$ that is the closest to the obstacle center o_j, i.e., $\theta_{i,k}^* = \arg\min_{\theta_{i,k} \in [0,1]} \{\|L_{i,k}(\theta_{i,k}) - o_j)\|\}$, which can be efficiently computed according to the algorithm in [53]. Then the distance d_{ij}^k between the segment $L_{i,k}$ of the rectangular robot i and the center of the obstacle j can be calculated as

$$d_{ij}^k = \|l_{i,k}(\theta_{i,k}^*) - o_j\|, \quad \forall k \in \{1, ..., 4\}, \quad (36)$$

with gradient $\nabla_{x_i} d_{ij}^k = \left(l_{i,k}(\theta_{i,k}^*) - o_j\right)/d_{ij}^k$. To ensure that no collision occurs between the rectangular robot and the object, we require the distance between each segment and the obstacle center to be greater than the obstacle radius, i.e., we choose $h_i^{o_j} = [d_{ij}^1 - \rho_j^o,\ d_{ij}^2 - \rho_j^o,\ d_{ij}^3 - \rho_j^o,\ d_{ij}^4 - \rho_j^o]^T$ and formulate the constraint (34a) as

$$\begin{bmatrix} \nabla_{x_i} d_{ij}^1{}^T \\ \nabla_{x_i} d_{ij}^2{}^T \\ \nabla_{x_i} d_{ij}^3{}^T \\ \nabla_{x_i} d_{ij}^4{}^T \end{bmatrix} u_i^* \geq -\phi^o \begin{bmatrix} d_{ij}^1 - \rho_j^o \\ d_{ij}^2 - \rho_j^o \\ d_{ij}^3 - \rho_j^o \\ d_{ij}^4 - \rho_j^o \end{bmatrix}. \quad (37)$$

Finally, the case of parallelepiped representation of the robot i easily follows by extending the previous reasoning to all of its 12 segments. It is worth noticing that, by following a reasoning similar to the above, it is also possible to consider rectangular (or parallelepiped) shapes for the obstacles.

Connectivity. The presence of the hard constraint on obstacle avoidance might lead the robots to disconnect despite the presence of the term u_i^c in the control law (see the simulation results in Section VIII for more details). Hence, we additionally include a connectivity constraint in the safety filter. More specifically, let $\mathcal{H}^c$ be the set of connected states, i.e., $\mathcal{H}^c = \{x \in \mathbb{R}^{np} : \quad \lambda_2(x) > 0\}$, and let us first analyze a centralized scenario. The CBF $h^c : \mathcal{H}^c \to \mathbb{R}$ proposed in [31] can be taken into account:

$$h^c(x) = \lambda_2(x) - \varepsilon^\lambda, \quad (38)$$

resulting in the safe set $\mathcal{C}^c = \{x \in \mathbb{R}^{np} : \quad \lambda_2(x) \geq \varepsilon^\lambda\} \subset \mathcal{H}^c$. By defining the class-$\mathcal{K}$ function $\alpha^c(h^c(x)) = \phi^c h^c(x)$, with ϕ^c a positive value, the following constraint is considered

$$\nabla_x \lambda_2^T u^* \geq -\phi^c(\lambda_2 - \varepsilon^\lambda), \quad (39)$$

with $u^* = [u_1^{*T}\ ...\ u_n^{*T}]^T \in \mathbb{R}^{np}$, which is proven in [31] to guarantee convergence and belonging to the safe set $\mathcal{C}^c$ if the system starts connected. In the case of distributed architecture, the previous statements are applicable. In particular, given the estimated λ_2, i.e., $\hat{\lambda}_{2_i}$, the problem can be solved in a distributed manner by having each robot i solve the component i of constraint (39), as stated in [32]. Thus, given $h_i^c = \phi^c(\hat{\lambda}_{2_i} - \varepsilon^\lambda)$, constraint (34b) can be expressed as

$$\nabla_{x_i} \hat{\lambda}_{2_i}^T u_i^* \geq -\phi^c(\hat{\lambda}_{2_i} - \varepsilon^\lambda). \quad (40)$$

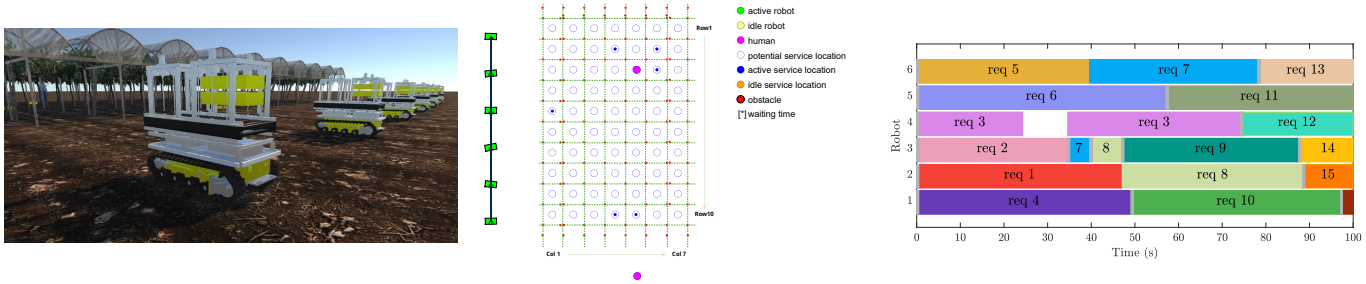
Note that in view of Assumption 2, the safety filter does not affect the completeness property discussed in Section VI-D and Theorem 2. Furthermore, note that a smaller positive threshold might be used in (38) instead of ε^λ to provide more flexibility to the robots for avoiding obstacles, allowing them to reach a lower connectivity value than the one considered in the control law. We do not further explore this direction for the sake of clarity of the paper presentation.

Remark 1. *To assess the time complexity of the proposed method, we can analyze the one of the two composing modules. For the Assignment module, its time complexity is dominated by the polynomial one of the distributed auction algorithm, which achieves $O(k_o n^3)$ in the worst-case scenario, with k_o a positive constant. Regarding the Control module, its time complexity is dominated by the one of the safety filter, which, being a QP problem with linear constraints and convex cost function, can be efficiently solved using polynomial-time algorithms. For instance, the interior point method for convex QP has complexity $O(p^3)$ [54], where p is 2 or 3 in our case. However, we would like to point out that the distributed assignment algorithm is only* sporadically *executed, triggered by specific events rather than running continuously as the control module. In particular, it is executed when the conditions on initial, new, or served requests are triggered. In contrast, during a global deadlock, the pausing algorithm (with complexity $O(n)$) alone can suffice, as each robot generally stays closer to its assigned request, thus reducing the computational burden.*

VIII. Simulation results

The proposed framework has been evaluated using MATLAB, which communicates via Robot Operating System (ROS) middleware with a Unity-based simulator created as part of the CANOPIES project[2]. The simulation represents a digital twin of the real-world table grape vineyard shown in Figure 1. The simulated environment, where a team of robots operates, is shown in Figure 6a. More in detail, a team of six Alitrak DCT-450P robots are deployed in a vineyard composed of 132 poles supporting a pergola structure. Additionally, two humans traverse the vineyard while the robots perform their tasks. Figure 6b provides a schematic overview of the

[2] https://pale.blue/solutions/robotics-autonomy/



(a) (b) (c)

Fig. 6: Representation of the environment in Unity (a) and respective schematic overview (b). Robot-request assignments over time in (c). Each request is associated with a unique color and gray borders indicate the respective request activation times.

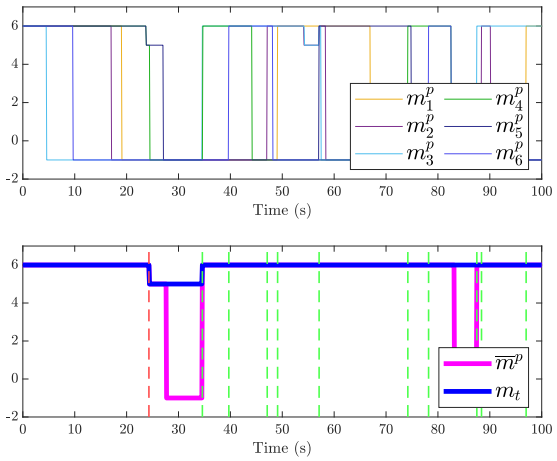


Fig. 7: Simulation results. Proposals for individual robots (top) and consensus values (bottom).

scenario showing the initial positions of the robots, humans, and requests. The field is modeled as a grid where, in each cell, a request can occur (blue circles). Note that, from a navigation standpoint, the considered scenario is structurally similar to other agricultural settings, like orchards, and the proposed approach can be easily used in row-arranged vineyards. Service requests, which correspond to box-exchange tasks, are randomly generated in empty grid cells, each with a processing time of 30 seconds. The number of requests $m(t)$ is fixed at 6 throughout the simulation. As soon as a robot completes a service, a new request is randomly generated at one of the available positions. The total simulation time is set to 100 seconds, during which 16 service requests are generated, with their locations listed in Table II. The parameters are empirically selected as $a = 0.2$ and $b = 25$ for (21), and for the controller and safety filter as: $R = 15$, $\kappa^p = 1.8$, $\kappa^c = 3.6$, $\kappa^e = 0.5$, $\varepsilon^\lambda = 0.1$, $\phi^l = 25$, $\phi^c = 1$ and $\phi^o = 5$.

A. Simulation Results

We analyze the simulation results in three time intervals as described below, and present two figures. Figure 6c shows the assignments made over time, with each color denoting a request and gray borders indicating the respective request

Table II: Simulation setup. Location of each request j in the field expressed as *(row, column)*.

j	Cell	j	Cell	j	Cell	j	Cell
1	(2,4)	5	(10,4)	9	(4,2)	13	(4,5)
2	(5,1)	6	(10,5)	10	(3,6)	14	(9,6)
3	(3,6)	7	(5,4)	11	(7,2)	15	(1,2)
4	(2,6)	8	(1,3)	12	(1,6)	16	(2,4)

activation times. Figure 7 shows the proposals made by the robots on the number of requests to handle (in the top part), as well as the maximum proposal value ($\overline{m}^p$) in magenta, and the number of requests that the robots attempt to serve simultaneously (m_t) in blue (in the bottom part). Vertical red dashed lines indicate global deadlocks, while vertical green dashed lines indicate the completion of a service. We can observe that the trend of $\overline{m}^p$ coincides with m_t unless all robots are serving or idle, and propose -1. The accompanying video shows the evolution of the overall simulation.

From 0 to 30 seconds. Initially, $m(0) = 6$ requests are randomly generated and assigned following the distributed assignment procedure. Robots 3, 6, and 2 reach their service locations at approximately $t = 5s$, $t = 10s$, and $t = 16s$, respectively, entering the local *Service* state and proposing $m_i^p = -1$, as shown in Figure 7. Due to connectivity constraints, the remaining robots sequentially enter local *Deadlock* states, proposing a decrease in m_t. At $t = 24s$, *Global deadlock* ($\overline{m}^p = 5$) is detected, triggering robot 4 to become *Idle*, and pausing its assigned request (request 3). Subsequently, robots 1 and 5 reach their service locations at $t = 26s$ and $t = 27s$, respectively, leading to all robots proposing $m_i^p = -1$. Hence, robot 4 stops as it is no longer required to improve connectivity (Figure 8).

From 30 to 60 seconds. When robot 3 completes its service at $t = 34.5$ s, request 7 is generated, and robot 3 is initially assigned to it. However, at $t = 40$ s, robot 6 is assigned request 7, while robot 3 takes request 8. This reassignment is due to the higher benefit for robot 3 in servicing request 8 ($\beta^d_{3,8} = 0.93$) compared to request 7 ($\beta^d_{3,7} = 0.98$), while robot 6 has a higher benefit for request 7 ($\beta^d_{6,7} = 0.9$) than request 8 ($\beta^d_{6,8} = 0.48$). A similar reassignment happens at $t = 47$ s, when request 8 is transferred from robot 3 to robot 2, and robot 3 takes request 9. The robots operate without connection problems until the end of this time interval.

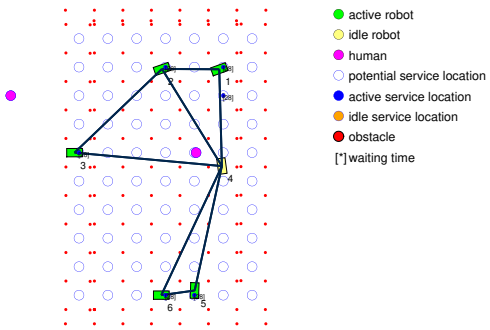


Fig. 8: Depiction of the idle state of robot 4.

From 60 to 100 seconds. At the beginning of this time interval, all robots are in the local reaching state and move towards the assigned locations. Starting from $t = 74s$, each robot successively proposes $m_i^p = -1$ when the service location is reached. At time $t = 85s$ all robots reach a consensus on $\overline{m}^p = -1$ (as shown in Figure 7), but in this case, no idle robots are present that have to stop. When a robot finishes serving (such as at $t = 88s$), a new request is generated, and the team returns to the assignment state. The robots navigate the vineyard without connection issues until the end of the simulation.

Note that, throughout the simulation, the robots demonstrate the ability to serve all requests while preserving connectivity and avoiding collisions both with static obstacles, i.e., poles of the vineyard, and with dynamic ones, i.e., with humans.

Connectivity. Figure 9a shows the estimated algebraic connectivity, $\hat{\lambda}_2$, for all agents (since their estimates coincide, no subscript is used). The connectivity remains above the threshold ε^λ (the red horizontal line), indicating that the network remains connected. The magenta line represents $\delta^\lambda = \varepsilon^\lambda + \kappa^e \log\left(\frac{\kappa^c}{\kappa^p}\right)$, used for local *Deadlock* detection: if the robot velocity is low and $\hat{\lambda}_2 \leq \delta^\lambda$, the robot proposes decreasing m_t. Figure 9b shows the algebraic connectivity (top) and robot velocities (bottom) during the first 50 seconds. At about $t = 24$ s, robots 1, 4, and 5 slow down significantly while others remain stationary. At this point, the algebraic connectivity drops close to δ^λ, indicating that not all robots can reach their service locations. As a result, robots 1, 4, and 5 enter local Deadlock states. At $t \in [28, 35]$ s, no robot motion is observed, as they are all in Service or Idle states.

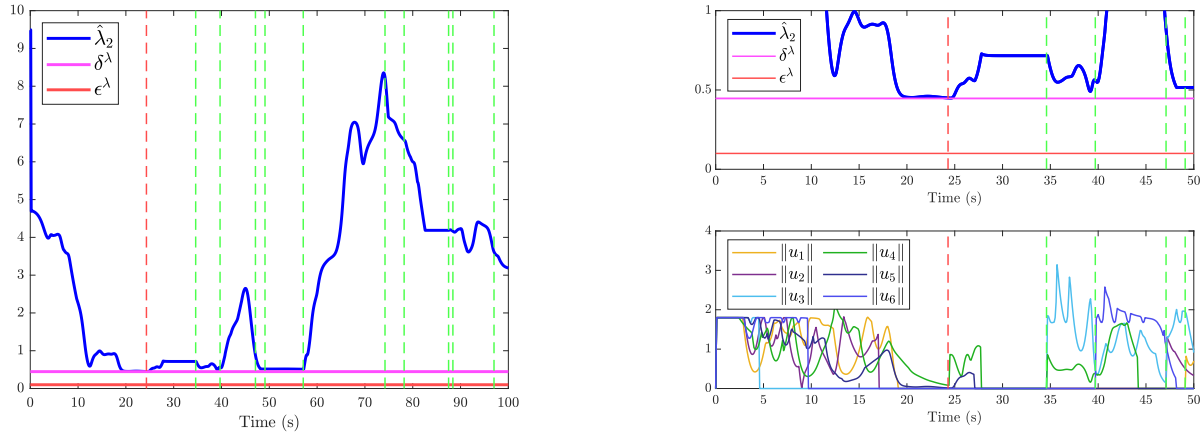
Benefit. Figure 10 shows the total benefit $\sum_{(i,j)\in\mathcal{P}(t)} \beta_{ij}(t)$ at each time t. Discontinuities are related to new assignments, whether from a new request or a pausing one. The benefit decreases considerably in correspondence of the red dotted line (denoting a global deadlock), which is expected since the number of requests to handle m_t is decreased by 1. Furthermore, the figure shows that the overall benefit is always non-decreasing until new assignments are made. More specifically, the benefit remains constant in the interval $[23, 35]s$ since the requests have equal waiting time and all active robots are serving, while it increases in the other intervals, showing that the robots successfully maximize the overall benefits.

B. Comparison with state-of-the-art approaches

We evaluate the scalability and performance of the proposed framework compared to two state-of-the-art baselines: the frameworks of Nestmeyer et al. [34] and Rooker et al. [35]. Briefly, as mentioned in the introduction, the work in [34] prioritizes the motion of a *prime traveler* robot, which is guaranteed to reach the target destination, while [35] resolves deadlocks by redirecting all robots to a meeting point, with one robot acting as a stationary beacon. Since both baselines address multi-target exploration applications and lack an assignment module, we integrate our assignment module into their methods for a fair comparison. We carry out simulations with robot teams of varying sizes (10, 20, 30, 40, and 50) within a 2-hectare area. For each team size n, we conduct 15 tests, each lasting 100 seconds, with randomized obstacle positions, and request locations. A constant number of requests $m(t) = n$ is maintained throughout each test. Figure 11 shows the average number of completed requests (top plot) and the average number of detected deadlocks (bottom plot) for each number of robots using the different frameworks (ours in blue, Nestmeyer et al. in red, and Rooker et al. in yellow). The variance is represented by the thin black lines. The results demonstrate that the proposed framework consistently serves more requests than both baselines, achieving higher efficiency and scalability. In detail, compared to [35], it leads to an average improvement of 45% for smaller teams ($n = 10$) and over 10% for larger teams ($n = 50$). This is because [35] disrupts parallel execution as robots abandon ongoing tasks and expend time traveling to and from the meeting point. Similarly, the proposed framework outperforms [34] by 10% for smaller teams and 5% for larger ones. Indeed, although the approach in [34] is effective in avoiding deadlocks, it limits parallelism by slowing or halting secondary robots to support the *prime traveler*. Note that the diminishing returns with large teams are only motivated by the fact that the environment has a fixed size in all tests. Hence, as the number of robots increases, deadlocks become less frequent because the robots are able to collectively cover almost the entire environment while preserving connectivity. With fewer deadlocks, the differences between the approaches diminish, as the primary task for the robots becomes to navigate the field and serve the requests.

C. Connectivity Analysis

This section aims to experimentally demonstrate the importance of including a connectivity component both in the controller and in the safety filter. Three solutions are compared: *S1)* a centralized architecture where the low-level controller only ensures reaching the assigned positions while the safety filter ensures connectivity maintenance; *S2)* a distributed version of the previous solution, where the local low-level controller only ensures reaching the assigned positions while the safety filter ensures connectivity maintenance; and *S3)* the proposed distributed architecture described in Section VII, with both the low-level controller and the safety filter including a connectivity component. We consider a case study involving $n = 3$ robots and $m = 3$ requests. Due to connectivity constraints, the robots cannot simultaneously reach all requests



(a) (b)

Fig. 9: Simulation results. Algebraic connectivity (on the left), with zoom during the first 50 seconds in the top right and the respective robot velocities in the bottom right.

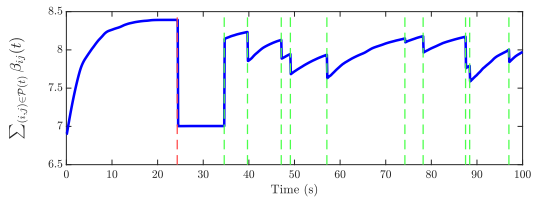


Fig. 10: Evolution of the total benefit over time. Discontinuities occur in correspondence of new assignment phases.

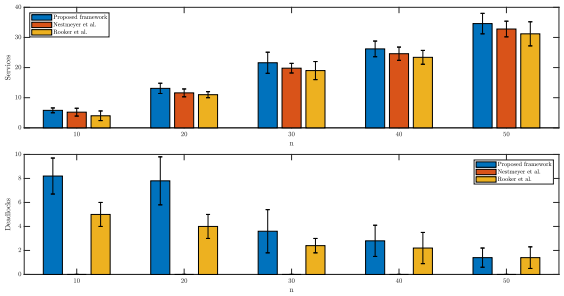


Fig. 11: Performance comparison. Average number of completed services per number of robots (top) and average number of deadlocks detected per number of robots (bottom).

but only one can be served at a time. We analyze the behavior until the first service request location is reached. The accompanying video shows the three solutions. Figure 12 shows the path of each robot (top) and algebraic connectivity evolution (bottom) with all the solutions. In all cases, the optimal initial assignment is $\mathcal{P} = \{(1,2),(2,1),(3,3)\}$.

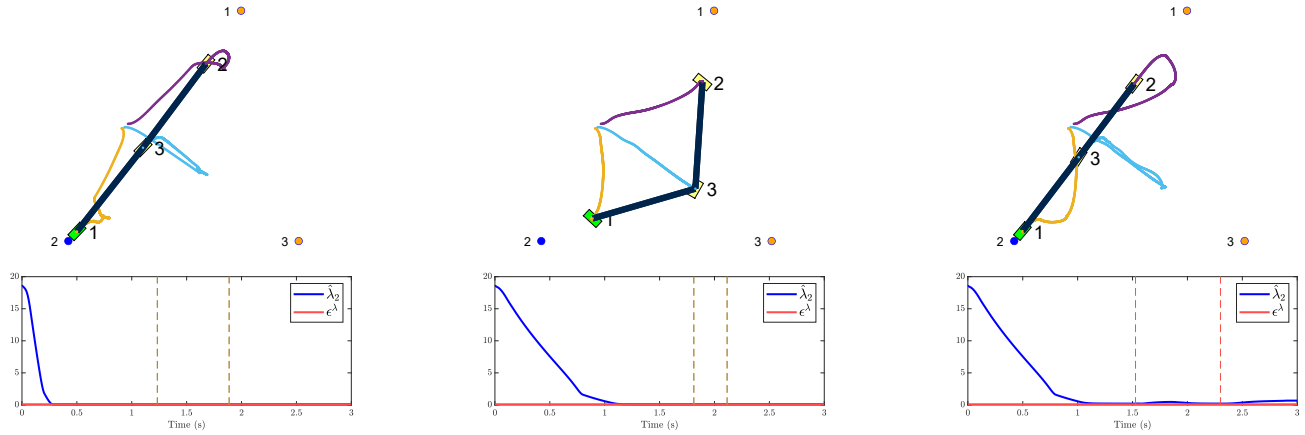
Figure 12a reports the outcome of S1. At the start, the robots attempt to reach the assigned positions, but stop once the connectivity limit is reached, causing robot 3 to become *Idle*. Robots 1 and 2 then proceed, while robot 3 is dragged to maintain connectivity. Once the limit is reached again, robot 2 becomes *Idle*. At this point, robot 1 reaches its assigned position by dragging the idle robots 2 and 3. With S1, the robots operate with minimal connectivity and move only when necessary after reaching the connection limit, thus no proactive behavior is exhibited by the idle robots.

Figure 12b shows the outcome of S2. As before, robot 3 becomes *Idle*. However, unlike the case with S1, robot 3 remains stationary to maintain connectivity. Subsequently, robot 2 also becomes *Idle*, leaving only robot 1 active and resulting in all robots remaining stationary. Therefore, none of the robots succeed in reaching the assigned position. Hence, with S2, we can observe that *i)* the algebraic connectivity remains at the limits of the safe set and *ii)* after at least one robot becomes *Idle*, all robots in team remain stationary, preventing the accomplishment of the service requests. This is motivated by the local safety filter's lack of global awareness, optimizing each robot's velocity independently. It acts as a minimum-energy controller, assigning zero velocity to *Idle* robots, causing active robots to halt to maintain connectivity.

Finally, Figure 12c shows the behavior with the proposed solution. In this case, robot 3 becomes *Idle* but actively tends to increase connectivity. Next, the constraints make robot 2 *Idle*, and both actively allow robot 1 to reach its target. Hence, in this case, the robots no longer work at the limits of the safe set, i.e., the *Idle* robots are no longer dragged by the active ones, but proactively operate to increase the algebraic connectivity, while the filter guarantees a minimum λ_2 value.

D. Realistic operational factors

To validate the applicability of the framework in real-world settings, we conduct additional simulations incorporating the following realistic operational factors: *i) Localization Errors (LE)*, simulated as Gaussian noise in position estimation with zero mean and variance 0.05 m. *ii) Packet Losses (PL)*, modeled as random message drops with rates uniformly distributed between 0.1% and 5%. *iii) Communication Delays (CD)*, represented as delays uniformly distributed between 0.04 s and 0.3 s. We evaluate the framework under ideal conditions,



(a) S1 (centralized solution with no connectivity action in the controller).

(b) S2 (distributed solution with no connectivity component in the controller).

(c) S3 (proposed distributed solution with connectivity component in the controller).

Fig. 12: Results of the connectivity analysis. The robots' paths (top), and algebraic connectivity (bottom) are shown.

Table III: Results with different realistic operational factors (LE, PL, CD) included and in the ideal case.

	Ideal	LE	PL	CD	Combo
Serv.	10.2 ± 0.8	10.2 ± 0.9	10.1 ± 0.7	9.8 ± 0.6	9.8 ± 1.2
Dead.	1.2 ± 1.2	1.1 ± 1.1	1.3 ± 1.3	1.2 ± 1.2	1.4 ± 1.4

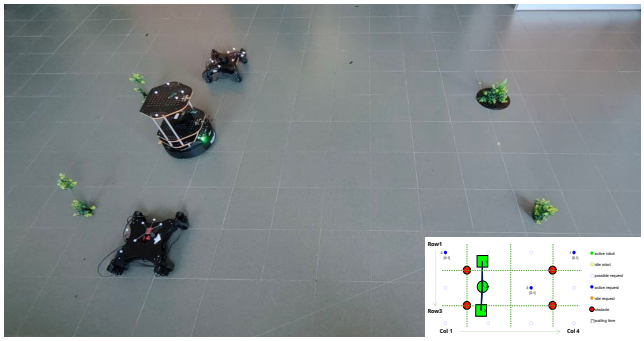


Fig. 13: Representation of the laboratory setup.

when including individual factors (LE, PL, CD), and with combined factors using the simulation setup in Section VIII-A with $n = 6$ robots and 15 tests. Each test has random target locations and duration equal to 100 s. Table III summarizes the results: the first row shows the average number of services completed, while the second row reports the number of deadlocks detected. Results confirm the framework's robustness, completing approximately 10 services across all scenarios. LE has minimal impact compared to PL and CD, while CD has the greatest effect due to delays in request processing.

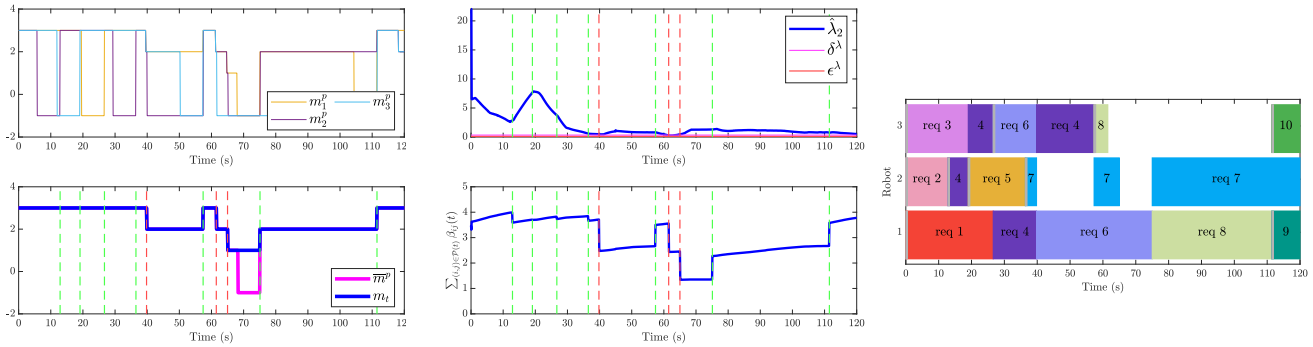
Table IV: Experimental setup. Location of each request j expressed as *(row, column)*.

j	**Cell**	j	**Cell**	j	**Cell**	j	**Cell**	j	**Cell**
1	(2,3)	3	(3,3)	5	(2,2)	7	(1,1)	9	(1,4)
2	(3,2)	4	(1,4)	6	(2,4)	8	(3,3)	10	(1,3)

IX. Experimental validation

After validating the proposed framework through simulations including realistic operational factors, we further test it in a real-world laboratory environment. The team consists of three robots: one TurtleBot2 and two TurtleBot Conveyors. The TurtleBot2 is a differential drive robot with a circular base, equipped with an Intel NUC with i7 processor and 16GB of RAM. The TurtleBot Conveyors are omnidirectional robots with a square base, each equipped with a Raspberry Pi 4 with 8GB of RAM. Each robot has an integrated WiFi module for communication and uses ROS middleware for communication and control. Localization is obtained through an OptiTrack motion capture system, providing precise positioning data transmitted to the robots via a router. The system latency ranges from 30 to 70 milliseconds. Note that the localization information is shared through ROS topics, making the framework agnostic to the specific localization module. As a result, the Optitrack system can be easily replaced with alternative solutions, such as GPS modules for outdoor environments. A small agricultural setting, with size $7\,\text{m} \times 4\,\text{m}$, is recreated where the field comprises four plants that must be avoided, and twelve potential request locations arranged around them as illustrated in Figure 13. Similar to the simulation case study, the number of requests is set equal to 3 at all times and the same set of parameters as in the simulation is employed except for κ^p and κ^c which are scaled by a factor of 10, and R which is set equal to 2. These choices are motivated by the need to limit the robots speed in the real setup and to facilitate operating conditions at the connectivity limit within the confined laboratory space. The accompanying video reports the execution of the experiment.

Figure 14 shows the results of a two-minute experiment. Figure 14a depicts robot proposals (top) and the consensus value (bottom). Figure 14b represents algebraic connectivity (top) and overall benefit (bottom). Red dashed lines mark global deadlock events, while green dashed lines indicate service completions. Figures 14c and Table IV detail the robot-request assignments over time and request locations,



(a) (b) (c)

Fig. 14: Experimental results. Proposals $m_i^p, \forall i$ (top left), consensus value for global deadlock (bottom center), algebraic connectivity (top center), overall benefit evolution (bottom right), robot-request assignments over time (right). Each request is associated with a unique color while gray borders indicate the respective request activation times.

respectively. The experiment starts with each robot assigned to a request. By $t = 40$s, four requests are fulfilled without violating connectivity constraints. At this point, a global deadlock is detected, reducing the number of active requests, and robot 2 is set to *Idle* (following Algorithm 1). This allows robot 3 to complete its task at $t = 57$s and initiate a new assignment with a higher m_t. In the next 10 seconds, two additional global deadlocks occur at $t = 62$s and $t = 65$s. After the first deadlock, robot 3 becomes *Idle*, but this does not resolve the issue, so a second deadlock occurs, where both robots 2 and 3 are set to *Idle*. Robot 1 is reassigned to request 6 due to its longer waiting time and proximity to the team. At $t = 75$s, robot 1 completes its service, and m_t increases. Robots 1 and 2 are reassigned to requests 8 and 7, respectively, while request 9 is not assigned due to its lower priority. By $t = 112$s, request 8 is fulfilled, and m_t is increased again. From this point on, the robots coordinate effectively, and no further deadlocks occur. The experiment shows that, despite unmodeled real-world factors such as localization errors and communication delays, the robots can safely complete tasks and perform similarly to the simulation, successfully addressing Problem 1.

X. Conclusions

In this paper, we proposed a distributed integrated framework for multi-robot task allocation and safe coordination in dynamic environments. More in detail, a two-layer architecture was designed where the top layer handles the high-level decision-making strategy and defines the assignment of the service requests to the available robots in a distributed manner, while the bottom layer provides the low-level distributed control law for safe coordination, i.e., ensuring connectivity maintenance and no collision with obstacles such as human operators. The behavior of each robot is defined according to a local FSM which was proven to be equivalent to a global FSM, where the knowledge of the state of all robots is assumed. The completeness of the proposed method was demonstrated. Finally, numerical simulations and experiments with a real team of mobile robots were provided to validate the approach.

Appendix

A. Proof of Theorem 1

In view of the reasoning in [48], for each robot i we have

$$\widehat{m}_i^n(k) = \overline{m}^p(k-n), \quad \forall k \geq k_0 + n, \tag{41}$$

where n represents an upper bound of the graph diameter and k_0 is the start time. At this point, we can characterize the convergence properties of the EDMC algorithm in our particular setting. In particular, we can observe that, when the robots are running the EDMC protocol, the only proposal change by a robot i occurs when it leaves the reaching state R_i and enters either the serving state or the local deadlock state, where it proposes -1 and $m_t - 1$, respectively. If this change influences the maximum value $\overline{m}^p(k)$, this is detected in maximum n steps as per (41). Hence, the EDMC protocol enables the successful detection of global deadlocks, i.e., the fulfillment of condition (26), with a maximum delay of n steps. Note that this delay does not compromise the team behavior but only causes the robots to remain stationary until the global deadlock is detected by all of them.

B. Proof of Theorem 2

Let us recall that δ^λ is defined as $\delta^\lambda = \varepsilon^\lambda + \kappa^e \log\left(\frac{\kappa^c}{\kappa^p}\right)$. As per the assumption, we focus on the case when only one agent r is in the *Reaching* state R_r and the others are in the *Idle* state. From [30, Theorem 6], it follows that the robot r is guaranteed to reach the assigned position if

$$\lambda_2(x) > \delta^\lambda. \tag{42}$$

This implies that as long as (42) is fulfilled, the robot r moves towards the target. Let $\mathcal{H}^\lambda = \{x \in \mathbb{R}^{np} \,|\, \lambda_2(x) > \delta^\lambda\}$ denote the set of states for which (42) is met. To prove the theorem, we aim to demonstrate that for this operational setting *i)* if the team starts in a situation with $\lambda_2(x) \leq \delta^\lambda$, then the connectivity will increase and will fulfill (42), i.e., the system state x will enter the set $\mathcal{H}^\lambda$, and *ii)* once the condition (42) is met, it will remain satisfied at all times, i.e., $\mathcal{H}^\lambda$ represents an invariant set. This ensures that the robot r will reach the

assigned position as per [30, Theorem 6]. Clearly, if at the beginning the condition (42) is met, the second point applies and the desired position is reached by r.

Part 1. Let us analyze the following Lyapunov candidate

$$V(x) = e^{\frac{(\varepsilon^\lambda - \lambda_2)}{\kappa^e}}.$$

Due to the discontinuity in (32), a non-smooth analysis [55], [56] should be carried out to analyze $V(x)$. This formalism is here omitted for the sake of space and for the simplicity of notation; however, the same results would follow. Hence, the following derivative is considered

$$\dot{V} = \sum_{i\in\mathcal{V}} \left(-\frac{1}{\kappa^e} e^{\frac{(\varepsilon^\lambda - \lambda_2)}{\kappa^e}} \nabla_{x_i}\lambda_2 \right)^T u_i = -\frac{1}{\kappa^e} e^{\frac{(\varepsilon^\lambda - \lambda_2)}{\kappa^e}} \sum_{i\in\mathcal{V}} g_i, \tag{43}$$

with $g_i = (\nabla_{x_i}\lambda_2)^T u_i$. By extending the result in [31, Lemma 1], it follows that, even if $\|\nabla_{x_r}\lambda_2(x)\| \neq 0$, there will exist at least one other agent $z \in \mathcal{V}$, with $z \neq r$, such that $\|\nabla_{x_z}\lambda_2(x)\| \neq 0$ (since two "extreme nodes" as defined in [31, Lemma 1] always exist). Furthermore, this agent is in *Idle* state by assumption. Hence, it holds

$$g_z = (\nabla_{x_z}\lambda_2)^T \kappa^c e^{\frac{\varepsilon^\lambda - \lambda_2}{\kappa^e}} \frac{\nabla_{x_z}\lambda_2}{\|\nabla_{x_z}\lambda_2\|} = \kappa^c e^{\frac{\varepsilon^\lambda - \lambda_2}{\kappa^e}} \|\nabla_{x_z}\lambda_2\| > 0 \tag{44}$$

Similarly, for the other agents in *Idle* state $k \neq r$ and $k \neq z$ it holds $g_k > 0$ if $\|\nabla_{x_k}\lambda_2(x)\| \neq 0$, and $g_k = 0$ otherwise. Hence, $g_k \geq 0$ for all $k \in \mathcal{V}$ with $k \neq r$ and $k \neq z$. Regarding the agent r in *Reaching* state R_r, if $\|\nabla_{x_r}\lambda_2(x)\| = 0$, it holds $g_r = 0$; otherwise, we have

$$\begin{aligned} g_r &= (\nabla_{x_r}\lambda_2)^T \left(-\kappa^p \frac{x_r - p_j}{\|x_r - p_j\|} + \kappa^c e^{\frac{\varepsilon^\lambda - \lambda_2}{\kappa^e}} \frac{\nabla_{x_r}\lambda_2}{\|\nabla_{x_r}\lambda_2\|} \right) \\ &\geq \|\nabla_{x_r}\lambda_2\| \left(-\kappa^p + \kappa^c e^{\frac{\varepsilon^\lambda - \lambda_2}{\kappa^e}} \right) := \overline{g}_r, \end{aligned} \tag{45}$$

which leads to $g_r \geq \overline{g}_r \geq 0$ as long as $\lambda_2 \leq \delta^\lambda$. In view of (43) and the inequalities for g_r and g_z, we obtain that as long as $\lambda_2 \leq \delta^\lambda$, it holds $\dot{V} < 0$, implying that the algebraic connectivity will fulfill condition (42).

Part 2. At this point, our objective is to prove that $\mathcal{H}^\lambda$ is an invariant set. Let us consider that the system state belongs to $\mathcal{H}^\lambda$. Clearly, if the system connectivity increases, the system state will continue to belong to $\mathcal{H}^\lambda$. Therefore, we analyze the scenario where the connectivity decreases. We can observe that when $\lambda_2 \to {\delta^\lambda}^+$ it holds

$$(-\kappa^p + \kappa^c e^{\frac{\varepsilon^\lambda - \lambda_2}{\kappa^e}}) \to 0, \tag{46}$$

while the term g_z in (44) is always positive, i.e., it contributes to increasing the connectivity. Hence, by the continuity of the state variables and by considering the expression of $\overline{g}_r$ in (45) and the bound in (46), we have that, if $\lambda_2(x)$ is decreasing, there exists a value x such that $||g_z|| > ||\overline{g}_r||$, resulting in $\dot{V} < 0$. Hence, when $\dot{V} < 0$, an increase of the connectivity occurs, resulting in the system not leaving the set $\mathcal{H}^\lambda$ and demonstrating its invariance.

REFERENCES

[1] J. Cortés and M. Egerstedt, "Coordinated control of multi-robot systems: A survey," *SICE J. Control Meas. Syst. Integr.*, vol. 10, no. 6, pp. 495–503, 2017.

[2] L. Sabattini, C. Secchi, N. Chopra, and A. Gasparri, "Distributed control of multirobot systems with global connectivity maintenance," *IEEE Trans. Robot.*, vol. 29, no. 5, pp. 1326–1332, 2013.

[3] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative robots and sensor networks*, pp. 31–51, 2015.

[4] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[5] K. M. Wurm, C. Stachniss, and W. Burgard, "Coordinated multi-robot exploration using a segmentation of the environment," in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pp. 1160–1165, 2008.

[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart, and P. Beardsley, "Multi-robot system for artistic pattern formation," in *IEEE Int. Conf. Robot. Autom.*, pp. 4512–4517, 2011.

[7] S. Giordani, M. Lujak, and F. Martinelli, "A distributed algorithm for the multi-robot task allocation problem," in *Trends in Applied Intelligent Systems*, pp. 721–730, 2010.

[8] S. Chopra, G. Notarstefano, M. Rice, and M. Egerstedt, "A distributed version of the hungarian method for multirobot assignment," *IEEE Trans. Robot.*, vol. 33, no. 4, pp. 932–947, 2017.

[9] D. Bertsekas, "New auction algorithms for the assignment problem and extensions," *Results in Control and Optimization*, vol. 14, p. 100383, 2024.

[10] D. P. Bertsekas, "Auction algorithms for network flow problems: A tutorial introduction," *Computational Optimization and Applications*, vol. 1, no. 1, pp. 7–66, 1992.

[11] D. P. Bertsekas and D. A. Castañon, "Parallel synchronous and asynchronous implementations of the auction algorithm," *Parallel Computing*, vol. 17, no. 6, pp. 707–732, 1991.

[12] M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas, "A distributed auction algorithm for the assignment problem," in *IEEE Conf. Decision Control*, pp. 1212–1217, 2008.

[13] D.-H. Lee, S. A. Zaheer, and J.-H. Kim, "A resource-oriented, decentralized auction algorithm for multirobot task allocation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 4, pp. 1469–1481, 2015.

[14] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Auton. Robots*, vol. 44, pp. 547–584, 2020.

[15] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *Int. J. Adv. Robot. Syst.*, vol. 10, no. 12, p. 399, 2013.

[16] C. W. Warren, "Multiple robot path coordination using artificial potential fields," in *IEEE Int. Conf. Robot. Autom.*, pp. 500–505, 1990.

[17] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Flocking in fixed and switching networks," *IEEE Trans. Autom. Control*, vol. 52, no. 5, pp. 863–868, 2007.

[18] L. Sabattini, C. Secchi, and C. Fantuzzi, "Arbitrarily shaped formations of mobile robots: artificial potential fields and coordinate transformation," *Auton. Robots*, vol. 30, pp. 385–397, 2011.

[19] M. Santilli, P. Mukherjee, R. K. Williams, and A. Gasparri, "Multirobot field of view control with adaptive decentralization," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2131–2150, 2022.

[20] P. Yu and D. V. Dimarogonas, "Distributed motion coordination for multirobot systems under ltl specifications," *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 1047–1062, 2021.

[21] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, *et al.*, "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm5954, 2022.

[22] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Europ. Control Conf.*, pp. 3420–3431, 2019.

[23] Y. Chen, A. Singletary, and A. D. Ames, "Guaranteed obstacle avoidance for multi-robot operations with limited actuation: A control barrier function approach," *IEEE Control Syst. Lett.*, vol. 5, no. 1, pp. 127–132, 2020.

[24] C. Jiang and Y. Guo, "Incorporating control barrier functions in distributed model predictive control for multirobot coordinated control," *IEEE Trans. Control Netw. Syst.*, vol. 11, no. 1, pp. 547–557, 2024.

[25] M. Cavorsi, L. Sabattini, and S. Gil, "Multirobot adversarial resilience using control barrier functions," *IEEE Trans. Robot.*, vol. 40, pp. 797–815, 2024.

[26] M. Lippi and A. Marino, "A control barrier function approach to human-multi-robot safe interaction," in *Mediterranean Conf. Control Autom.*, pp. 604–609, 2021.

[27] D. V. Dimarogonas and K. J. Kyriakopoulos, "Connectedness preserving distributed swarm aggregation for multiple kinematic robots," *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 1213–1223, 2008.

[28] M. M. Zavlanos and G. J. Pappas, "Potential fields for maintaining connectivity of mobile networks," *IEEE Trans. Robot.*, vol. 23, no. 4, pp. 812–816, 2007.

[29] M. Ji and M. Egerstedt, "Distributed coordination control of multiagent systems while preserving connectedness," *IEEE Trans. Robot.*, vol. 23, no. 4, pp. 693–703, 2007.

[30] A. Gasparri, L. Sabattini, and G. Ulivi, "Bounded control law for global connectivity maintenance in cooperative multirobot systems," *IEEE Trans. Robot.*, vol. 33, no. 3, pp. 700–717, 2017.

[31] B. Capelli and L. Sabattini, "Connectivity maintenance: Global and optimized approach through control barrier functions," in *IEEE Int. Conf. Robot. Autom.*, pp. 5590–5596, 2020.

[32] B. Capelli, H. Fouad, G. Beltrame, and L. Sabattini, "Decentralized connectivity maintenance with time delays using control barrier functions," in *IEEE Int. Conf. Robot. Autom.*, pp. 1586–1592, 2021.

[33] A. Miele, M. Lippi, and A. Gasparri, "A framework based on control barrier functions for time-varying connectivity maintenance," in *IEEE Int. Conf. Autom. Scie. Eng.*, pp. 84–89, 2024.

[34] T. Nestmeyer, P. Robuffo Giordano, H. H. Bülthoff, and A. Franchi, "Decentralized simultaneous multi-target exploration using a connected network of multiple robots," *Auton. Robots*, vol. 41, pp. 989–1011, 2017.

[35] M. N. Rooker and A. Birk, "Multi-robot exploration under the constraints of wireless networking," *Control Eng. Practice*, vol. 15, no. 4, pp. 435–445, 2007.

[36] N. Majcherczyk, A. Jayabalan, G. Beltrame, and C. Pinciroli, "Decentralized connectivity-preserving deployment of large-scale robot swarms," in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pp. 4295–4302, 2018.

[37] J. Cao, W. W. L. Leong, R. S. H. Teo, and S. Huang, "A general framework for multi-uav communication connectivity maintenance through scalable task allocation," in *Int. Conf. Unmanned Aircraft Syst.*, pp. 549–556, 2023.

[38] P. Forte, A. Mannucci, H. Andreasson, and F. Pecora, "Online task assignment and coordination in multi-robot fleets," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4584–4591, 2021.

[39] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5816–5823, 2021.

[40] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *Int. J. Robot. Res.*, vol. 37, no. 7, pp. 818–838, 2018.

[41] F. Faruq, D. Parker, B. Laccrda, and N. Hawes, "Simultaneous task allocation and planning under uncertainty," in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pp. 3559–3564, 2018.

[42] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Distributed multi-robot task assignment and formation control," in *IEEE Int. Conf. Robot. Autom.*, pp. 128–133, 2008.

[43] Y. Chen, U. Rosolia, and A. D. Ames, "Decentralized task and path planning for multi-robot systems," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4337–4344, 2021.

[44] G. Xu, X. Kang, H. Yang, Y. Wu, W. Liu, J. Cao, and Y. Liu, "Distributed multi-vehicle task assignment and motion planning in dense environments," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–13, 2023.

[45] A. Agrawal, S. Hariharan, A. S. Bedi, and D. Manocha, "Dc-mrta: Decentralized multi-robot task allocation and navigation in complex environments," in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pp. 11711–11718, 2022.

[46] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010.

[47] B. Mohar, Y. Alavi, G. Chartrand, and O. Oellermann, "The laplacian spectrum of graphs," *Graph theory, combinatorics, and applications*, vol. 2, no. 871-898, p. 12, 1991.

[48] D. Deplano, M. Franceschelli, and A. Giua, "Dynamic min and max consensus and size estimation of anonymous multiagent networks," *IEEE Trans. Autom. Control*, vol. 68, no. 1, pp. 202–213, 2023.

[49] N. Mishra, A. Singh, S. Kumari, K. Govindan, and S. I. Ali, "Cloud-based multi-agent architecture for effective planning and scheduling of distributed manufacturing," *Int. J. Prod. Res.*, vol. 54, no. 23, pp. 7115–7128, 2016.

[50] Y. Kotb, I. Al Ridhawi, M. Aloqaily, T. Baker, Y. Jararweh, and H. Tawfik, "Cloud-based multi-agent cooperation for iot devices using workflow-nets," *J. Grid Comput.*, vol. 17, no. 4, pp. 625–650, 2019.

[51] S. Giannini, A. Petitti, D. Di Paola, and A. Rizzo, "Asynchronous max-consensus protocol with time delays: Convergence results and applications," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 63, no. 2, pp. 256–264, 2016.

[52] P. Yang, R. Freeman, G. Gordon, K. Lynch, S. Srinivasa, and R. Sukthankar, "Decentralized estimation and control of graph connectivity for mobile sensor networks," *Automatica*, vol. 46, no. 2, pp. 390–396, 2010.

[53] V. J. Lumelsky, "On fast computation of distance between line segments," *Inf. Process. Lett.*, vol. 21, no. 2, pp. 55–61, 1985.

[54] Y. Ye and E. Tse, "An extension of karmarkar's projective algorithm for convex quadratic programming," *Mathematical Programming*, vol. 44, no. 1, pp. 157–179, 1989.

[55] D. Shevitz and B. Paden, "Lyapunov stability theory of nonsmooth systems," *IEEE Trans. Autom. Control*, vol. 39, no. 9, pp. 1910–1914, 1994.

[56] B. Paden and S. Sastry, "A calculus for computing Filippov's differential inclusion with application to the variable structure control of robot manipulators," *IEEE Trans. Circuits Syst.*, vol. 34, no. 1, pp. 73–82, 1987.

Andrea Miele received the M.sc in computer science and automation engineering (cum laude) from the Roma Tre University, Rome, Italy, in 2023, and is currently pursuing a Ph.D. degree in computer science and automation at Roma Tre University, Rome, Italy, under the supervision of prof. Andrea Gasparri. His main research interests include distributed systems, distributed optimization, and precision agriculture.

Martina Lippi received the M.Sc. (cum laude) and Ph.D. degrees in Information Engineering from the University of Salerno, Fisciano, Italy, in 2017 and 2020, respectively. In 2019, she was a Visiting Scholar with the KTH Royal Institute of Technology, Sweden. From November 2020 to June 2022, she was a Postdoctoral Researcher with Roma Tre University, Italy, where she has been an Assistant Professor since June 2022. Her research interests include human–robot interaction, multimanipulator systems, and distributed control.

Andrea Gasparri (M'09, SM'19) received the Laurea (cum laude) degree in computer science and the Ph.D. degree in computer science and automation from Roma Tre University, Rome, Italy, in 2004 and 2008, respectively. He is currently a Professor at Roma Tre University. His research interests include robotics, sensor networks, networked multiagent systems, and precision agriculture. He was the recipient of the Italian Grant FIRB Futuro in Ricerca 2008 for the project Networked Collaborative Team of Autonomous Robots. He was the Coordinator of the project "PANTHEON" supported by the European Community within the H2020 framework (grant agreement number 774571) and is currently the Coordinator of the project "CANOPIES" within the H2020 framework (grant agreement number 101016906).