Automatic Deployment of Distributed Teams of Robots From Temporal Logic Motion Specifications

Marius Kloetzer and Calin Belta, *Member, IEEE*

***Abstract*—We present a computational framework for automatic synthesis of decentralized communication and control strategies for a robotic team from global specifications, which are given as temporal and logic statements about visiting regions of interest in a partitioned environment. We consider a purely discrete scenario, where the robots move among the vertices of a graph. However, by employing recent results on invariance and facet reachability for dynamical systems in environments with polyhedral partitions, the framework from this paper can be directly implemented for robots with continuous dynamics. While allowing for a rich specification language and guaranteeing the correctness of the solution, our approach is conservative in the sense that we might not find a solution, even if one exists. The overall amount of required computation is large. However, most of it is performed offline before the deployment. Illustrative simulations and experimental results are included.**

***Index Terms*—Cooperative systems, distributed control, mobile-robot motion planning, temporal logic.**

I. Introduction

THE GOAL in motion planning and control is to be able to specify a motion task in a rich, high-level language and have the robot(s) automatically convert this specification into a set of low-level primitives, such as feedback controllers and communication protocols, to accomplish the task [2]–[4]. This study is motivated by two disadvantages of the current approaches to robot motion planning and control. First, in most of the existing works, the motion-planning problem is simply specified as "go from A to B while avoiding obstacles" [2]. This is not rich enough to describe a large class of tasks of interest in practical applications. The accomplishment of the mission might require the attainment of either A or B, convergence to a region (i.e., "reach A eventually, and stay there for all future times"), visiting targets sequentially (i.e., "reach A, then B, and then, C"), surveillance (i.e., "reach A, and then B, infinitely often"), or the satisfaction of more complicated temporal and logic conditions about the reachability of regions of interest (e.g., "Never go to A. Do not go to B unless C or D were visited."). Second, most current approaches to multirobot planning and control are bottom-up, in the sense that local interaction rules, which are thought to be true in natural systems [5] (e.g., schools of fish and swarms of bees) or designed from first principles [6], are shown to produce emergent behaviors at global level (largely known as "consensus algorithms" [7], [8]). However, the inverse (i.e., top-down) problem, which seems more relevant to robotics, remains largely unanswered: "Can we automatically generate provably correct local communication and control strategies from global task specifications given in rich and natural language over regions of interest in an environment?"

The starting point for this paper is the observation that "rich" and "human-like" task specifications, such as the temporal and logic statements about the reachability of regions of interest given earlier, translate naturally to formulas of temporal logics, such as linear temporal logic (LTL) and computation tree logic (CTL) [9]. Such logics and corresponding model-checking algorithms are normally used to specify and check the correctness of computer programs, which can be seen as continuously operating, reactive (concurrent) systems [10].

Inspired by this, and enabled by recent results on the construction of feedback controllers for facet reachability and invariance in polytopes [11]–[13], we consider a purely discrete problem, in which n agents can move among the vertices of a graph, which can be interpreted as the quotient of a partitioned environment. We model the robot communication constraints as a graph, which can, in general, vary in time to capture proximity or line-of-sight communication constraints. The motion of each robot is modeled as a transition system over the graph that models the environment, whose transitions capture the robot-control constraints (e.g., due to underactuation, a robot cannot move from a region to a particular adjacent region in a partitioned environment). We present a framework for automatic generation of local control strategies from a task specification, which is given as an arbitrary LTL formula over the set of vertices. The solution to this purely discrete and finite-dimensional problem is readily implementable for robots with (continuous) dynamics moving in (infinite) environments partitioned using popular schemes, such as triangulations and rectangular grids, as already demonstrated for the one-robot case in our previous work [14], [15].

The use of temporal logic for task specification and controller synthesis in mobile robotics has been advocated in [16], and

Manuscript received July 9, 2009; revised October 15, 2009. First published December 4, 2009; current version published February 9, 2010. This paper was recommended for publication by Associate Editor G. Antonelli and Editor L. Parker upon evaluation of the reviewers' comments. This work was supported in part by the Air Force Office of Scientific Research Young Investigator Research Program under Grant FA9550-09-1-020, in part by the Army Research Office (ARO) under Contract W911NF-09-1-0088, and in part by the National Science Foundation under Grant CNS-0834260. This paper was presented in part at the IEEE International Conference on Networking, Sensing and Control, Ft. Lauderdale, FL, April 23–25, 2006, and also at the IEEE International Conference on Robotics and Automation, Pasadena, CA, May 19–23, 2008.

M. Kloetzer is with the Technical University of Iasi, 700050 Iasi, Romania (e-mail : kmarius@ac.tuiasi.ro).

C. Belta is with the Department of Mechanical Engineering and the Division of Systems Engineering, Boston University, Brookline, MA 02446 USA (e-mail: cbelta@bu.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2009.2035776

1552-3098/$26.00 © 2009 IEEE
Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

more recent works include [15] and [17]–[20]. As opposed to [15] and [20], here, we consider teams, rather than just single agents. In the robotics literature, the closest related works are [18] and [19], which treat the problem of deployment of robotic teams from temporal-logic specifications. Quottrup *et al.* [18] consider a team of robots interacting among themselves and with a dynamic environment. The system in which all control actions are allowed is model-checked with respect to a timed CTL specification using UPPAAL. If the system is correct, a witness trajectory, which can be optimal in either the number of discrete steps or in the total time, is generated. The policy to produce robot actions is obtained in [19] by applying a game-theoretic framework [21]. While this allows accommodation of environmental events, the specification language is limited to formulas in the GR(1) fragment of LTL. As opposed to both these works, in this paper, the specifications are global (i.e., over environmental features), with no subtask-robot preassignment. However, in the present version, our framework does not capture environmental events.

Arguably, the closest related works are recent results in concurrency theory (for a review, see [22]), where global specifications given as languages over a set of actions are checked for implementability by a set of agents jointly owning the actions and synchronizing on them. However, in these works, the expressivity of the specifications is restricted to a subset of regular languages. In addition, in the robotics problem that we consider, the specification is given over features in the environment, and we could not find a general method to map these features to robot actions. Our approach also relates to (and draws inspiration from) control of discrete-event systems from specifications given in terms of languages [23].

The rest of this paper is organized as follows. Some preliminaries that are necessary throughout the paper are given in Section II. The problem is stated in Section III. The solution is presented in Sections IV and V. The complexity and conservativeness of the approach described in this paper is given in Section VI. The need for communication among the robots is discussed in Section VII, together with an algorithm to find distributed implementations that do not require moving robots to communicate for all times. The connection between the purely discrete framework, which is central to this paper, and robots with continuous dynamics and a simple experimental result are presented in Section VIII. Final remarks and discussions on the applicability of the method are presented in Section IX.

II. Preliminaries

Definition 1: A transition system is a tuple $T = (Q, Q_0, \rightarrow, \Pi, \models)$, where Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, $\rightarrow \subseteq Q \times Q$ is a transition relation, Π is a finite set of atomic propositions (or observations), and $\models \subseteq Q \times \Pi$ is a satisfaction relation.

For an arbitrary state $q \in Q$, let $\Pi_q = \{\pi \in \Pi \,|\, q \models \pi\}$, $\Pi_q \in 2^{\Pi}$ denote the set of all atomic propositions satisfied at q. A *trajectory* or *run* of T starting from q is an infinite sequence $r = r(1)r(2)r(3)\ldots$, with the property that $r(1) = q$, $r(i) \in Q$, and $(r(i), r(i+1)) \in \rightarrow \forall\, i \geq 1$. A trajectory $r = r(1)r(2)r(3)\ldots$ defines a *word* over the set 2^{Π}, $w = w(1)w(2)w(3)\ldots$, where $w(i) = \Pi_{r(i)}$. The set of all words that can be generated by T is called the (ω-) language of T.

In this paper, we consider motion specifications, which are given as formulas of a fragment of LTL [9] called LTL_{-X}, which we will simply denote by LTL throughout the paper. A formal definition for the syntax and semantics of LTL formulas is beyond the scope of this paper. Informally, LTL formulas are recursively defined over a set of atomic propositions Π by using the standard boolean operators and a set of temporal operators. The boolean operators are $\neg$ (for negations), $\vee$ (for disjunction), and $\wedge$ (for conjunction), and some temporal operators that we use include $\mathcal{U}$ (standing for "until"), $\Box$ (for "always"), and $\Diamond$ (for "eventually"). LTL formulas are interpreted over infinite words over the set 2^{Π}, like those generated by transition system T. Let us assume that ϕ_1 and ϕ_2 are two LTL formulas over Π, and w is a word produced by T. Formula $\phi_1 \mathcal{U} \phi_2$ intuitively means that (over word w) ϕ_2 will eventually become true and that ϕ_1 is true until this happens. Formula $\Diamond\phi$ means that ϕ becomes eventually true, whereas $\Box\phi$ indicates that ϕ is true at all positions of w. More expressiveness can be achieved by combining the operators that have been described earlier.

The expressivity of LTL makes it suitable to specify motion tasks, such as reachability (i.e., "reach π_1 eventually," which is written as $\Diamond\pi_1$), reachability and obstacle avoidance (i.e., "reach π_1 eventually, while always avoiding π_2," which is written as $\Diamond\pi_1 \wedge \Box\neg\pi_2$), convergence tasks ("reach π_1 eventually, and stay there for all future times," which is written as — $\Diamond\Box\pi_1$), etc. Moreover, if more robots are available, the attainment of disjoint regions at the same time might be of interest, as in "reach π_1 and π_2 eventually" [which is written as $\Diamond(\pi_1 \wedge \pi_2)$].

Remark 1: Classical LTL allows for an additional temporal operator, which is called "next." We do not allow for the "next" operator because, as shown in our previous work [15], it is meaningless when abstracting a continuous system to a finite discrete one, as considered in this paper. On the other hand, LTL without the "next" operator cannot distinguish between words with different numbers of finitely many consecutive repetitions of a symbol. For example, $\pi_1\pi_2\pi_2\pi_3\ldots$ satisfies exactly the same formulas as $\pi_1\pi_2\pi_3\ldots$. As a consequence, for a transition system T, as given in Definition 1, we will only consider words without finitely many consecutive repetitions of any symbol.

The method of checking whether the language of a transition system T satisfies an LTL formula ϕ over its set of propositions Π is called model checking. Available off-the-shelf packages for LTL model checking include NuSMV [24] and SPIN [25]. In short, given a transition system T and an LTL formula ϕ over its set of propositions, a model checker will return the initial states for which the language satisfies the formula. If the language generated from a state does not satisfy the formula, the model checker returns a certificate in the form of a counterexample, i.e., a run violating the formula. Among the several runs of T that satisfies a formula ϕ, there are some with a particular structure of a *prefix* followed by infinitely many repetitions of a *suffix*. A prefix is a finite trajectory from the initial state to a final state (excluding this latter state), while a suffix starts and ends at the aforementioned final state. If there exists a run of T

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

that satisfies ϕ starting from an initial state, then there always exists a run with the earlier particular prefix–suffix structure from that state [26]. Such a run is of particular interest to us, as will become clear in Section V.

In this paper, the problem of finding runs of T that satisfy a formula ϕ is of special interest. To this goal, one can use an off-the-shelf model checker. Indeed, by feeding T and $\neg\phi$ into a model checker, if the formula is not satisfied at a state, the model checker will return that initial state together with a counterexample, which is a run that satisfies ϕ. However, the user does not have any control over the structure of the produced counterexamples. They can be very long, or not even satisfy the particular structure, which is defined in Remark 1. An attractive alternative is to use our previously developed tool for model checking and control of transition systems [15], which is a tool that we will use in the approaches described in this paper. In short, given a formula ϕ and a transition system T, we would first derive a Büchi automaton B_ϕ and then construct a product automaton $A = T \times B_\phi$, whose language projected onto T satisfies ϕ. Then, in this transition system, we can search for runs with any imposed structure. In particular, we can look for runs in the prefix–suffix form, for runs which are "minimal" with respect to the overall length of prefix and suffix (or overall cost if weights are attached to the transitions of T), for runs that do not have finitely many consecutive repetitions of a symbol, etc. The obtained run can then be easily projected back to obtain a satisfying run of T.

III. Problem Formulation

Let

$$G = (P, \rightarrow_G) \tag{1}$$

be a graph where $P = \{p_1, \ldots, p_k\}$ is the set of vertices, and $\rightarrow_G \subset P \times P$ is a symmetric relation modeling the set of edges. As an exemplification of a mobile-agent framework, G can be the quotient graph of a partitioned environment, where P is a set of labels for the regions in the partition, and $\rightarrow_G$ is the corresponding adjacency relation. Let us assume that we have a team of n, where $n < k$, robots that can move instantaneously between adjacent vertices of G. We assume that the agents have identical communication capabilities, which are induced by the environment. Explicitly, the set of communication constraints is defined as

$$C = (P, \rightarrow_C) \tag{2}$$

where $\rightarrow_C \subset P \times P$ is a symmetric and reflexive relation. In other words, if two robots are at p_i and p_j, respectively, then they can directly communicate if and only if $(p_i, p_j) \in \rightarrow_C$. The assumption that $\rightarrow_C$ is reflexive means that any robot can communicate with itself at all times. While this does not make sense from an application point of view (i.e., it is obviously true), it is assumed for technical reasons to become clear later. While it is natural to assume that either $\rightarrow_G \cup_{j=1}^{k}(p_j, p_j) = \rightarrow_C$ (i.e., two robots can directly communicate if and only if they are in adjacent vertices), or $\rightarrow_G \cup_{j=1}^{k}(p_j, p_j) \subseteq \rightarrow_C$ (i.e., two robots can directly communicate when in adjacent vertices), neither of these assumptions are necessary. We assume that the communication is instantaneous, and each robot can act as a communication relay. In other words, any two robots in a connected component of C can instantaneously communicate.

We model the motion capabilities of each robot i, for $i = 1, \ldots, n$, on the graph G using a transition system T_i, which is defined as follows:

$$T_i = (Q_i, q_{0_i}, \rightarrow_i, \Pi_i, \models_i),\ i = 1, \ldots, n \tag{3}$$

where

1) $Q_i = P$ is the set of states;
2) $q_{0_i} \in Q_i$ is the initial location of agent i (a singleton);
3) $\rightarrow_i \subset P \times P$ is a reflexive transition relation that satisfies $\rightarrow_i \subseteq \rightarrow_G \cup_{j=1}^{k}(p_j, p_j)$, and $\rightarrow_i \subseteq \rightarrow_C$;
4) $\Pi_i = P$;
5) $\models_i$ is the trivial satisfaction relation $(q, \pi) \in \models_i$ if and only if $q = \pi$.

In other words, the motion of robot i among the vertices of G is restricted by the transition relation $\rightarrow_i$. We assume that each robot can stay at a vertex and can only move between adjacent vertices. However, it is not necessary that a robot can move between any adjacent vertices. In addition, in order to be able to avoid collision with other robots, we assume that a robot can only transit from a current vertex to a vertex with which communication is possible.

Note that the only differences between the transition systems T_i are given by their initial states and possibly by their transition relations (for agents with different movement capabilities). Also, for this particular definition of a transition system, trajectories are equivalent to words. From Remark 1, it is enough to consider words without finitely many consecutive repetitions of a symbol. A *motion* of robot i on the graph G is such a (infinite) word produced by T_i. The occurrence of a vertex in the motion of robot i means that the vertex is visited and then left. Infinitely many consecutive repetitions of a vertex means that the robot stays at that vertex for all future times. A *control strategy* for robot i is an algorithm that, for each state $q \in Q_i$, determines which allowed transition the robot should take and what the robot should communicate with the other robots in its communication range.

Remark 2: The finite and purely discrete framework introduced earlier represents a formal abstraction of a realistic multirobot scenario. Indeed, as already mentioned, G can be the quotient of a partitioned environment (which is obtained using, for example, triangulations or rectangular grids). The communication constraints formulated earlier can restrict the robots to communicate only when they are in adjacent regions or are "close" to each other. Alternatively, G can be a visibility graph, and the interrobot communication constraints might refer to line of sight. The "motion-capacity" relation $\rightarrow_i$ captures our capability to design controllers guaranteeing that robot i can be either kept in a region or driven to a neighbor region, regardless of the initial position of the robot in the current region. Note that here we assume that the first type of controller can always be found, i.e., $\rightarrow_i$ is reflexive—the need for this assumption will become clear later in the paper. Computationally efficient algorithms for the construction of such controllers exist for triangular and

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

rectangular partitions and robot dynamics ranging from fully actuated point-like robots to unicycles of nonnegligible size [14] (for more details and for an example, see Section VIII).

Definition 2: The *behavior* of the team of robots is an infinite sequence of sets $\{p_1^1, \ldots, p_n^1\}, \{p_1^2, \ldots, p_n^2\}, \ldots$, where $p_i^j \in P$, and $p_i^j \neq p_k^j$, for $i \neq k$, and $j = 1, 2, \ldots$. Each entry $\{p_1^j, \ldots, p_n^j\}$, for $j = 1, 2, \ldots$ denotes the set of vertices of G occupied by the n robots. The first entry corresponds to the initial positions of the robots $\{p_1^1, \ldots, p_n^1\} = \{q_{0_1}, \ldots, q_{0_n}\}$. A set $\{p_1^j, \ldots, p_n^j\}$, for $j \geq 2$, is added to the sequence whenever at least one robot changes its position. An infinite number of identical entries $\{p_1^j, \ldots, p_n^j\}$ is added to the sequence if the set of vertices $\{p_1^j, \ldots, p_n^j\}$ is reached and never left.

From Section II, the semantics of LTL formulas over P can be defined over team behaviors. We are now ready to formulate the main problem that we consider in this paper.

Problem 1: Given a team of n agents with motion capabilities (3) and communication constraints (2) on a graph (1), their initial nonoverlapping positions, and a task specified as an LTL formula ϕ over P, find individual control strategies such that the behavior of the team satisfies the specification. In addition, no two robots are allowed to overlap at a vertex or to swap vertices at any time.

Remark 3: Since, as required in Problem 1, no two robots can swap vertices at a time, there will be a change in the set of vertices occupied by the robots (i.e., a new entry added to the behavior from Definition 2), whenever at least one robot moves to a different vertex.

The solution to Problem 1 will be one generic algorithm, which will be run by each robot. From an implementation point of view, this means that the robots will be first programmed and then deployed in the environment and simultaneously powered on. Once the agents are programmed and started, they will autonomously evolve and interact with each other according to the imposed communication constraints. There will be no supervising controller for the team during the execution of the task.

A. Case Study

For illustration, throughout this paper, we consider the example, as shown in Fig. 1. A planar environment is partitioned into a set of $k = 16$ rectangles, which are labeled from the set $P = \{p_1, \ldots, p_{16}\}$. Two vertices p_i and p_j are adjacent, i.e., $(p_i, p_j) \in \rightarrow_G$ if and only if the corresponding rectangles share a facet. We consider a team of $n = 3$ identical robots, with initial positions in regions p_1, p_3, and p_{13}, respectively. Two robots can communicate when they are in rectangles that share an edge (rectangles diagonally placed do not communicate). In other words, the communication relation is given by $\rightarrow_C = \rightarrow_G \cup_{j=1}^{16} (p_j, p_j)$.

The motion of each robot is modeled as a transition system with 16 states, where each state corresponds to a rectangle from the partition. The initial states are $q_{0_1} = p_1$, $q_{0_2} = p_3$, and $q_{0_3} = p_{13}$. We assume that the robots have identical motion capabilities and, from every vertex, can move to all adjacent vertices. In other words, $\rightarrow_1 = \rightarrow_2 = \rightarrow_3 = \rightarrow_G \cup_{j=1}^{16} (p_j, p_j)$. The only differences among T_1, T_2, and T_3 are their initial states.

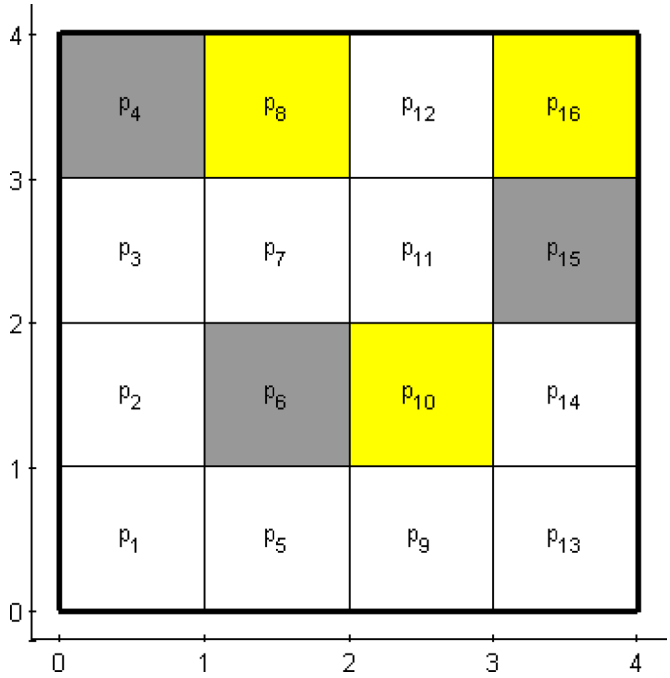

Fig. 1. Partitioned environment and the region labels for the case study considered throughout the paper.

We want to accomplish the following task: *"Do not visit any yellow region (p_8, p_{10}, and p_{16}) until all these three regions are simultaneously visited, and always avoid the gray regions (p_4, p_6, and p_{15})."* Note that the nature of the task clearly implies a cooperation in the form of synchronization between robots (all yellow regions must be simultaneously visited). The previous specification immediately translates to the following LTL formula:

$$\phi = \neg(p_8 \vee p_{10} \vee p_{16}) \mathcal{U} (p_8 \wedge p_{10} \wedge p_{16}) \wedge \Box \neg (p_4 \vee p_6 \vee p_{15}).$$

IV. Finding a Distributable Solution

To provide a solution to Problem 1, we proceed as follows. First, we construct a transition system T_g, which captures all possible motions of the group of robots while guaranteeing that no two robots ever overlap at a vertex or transit at the same time between adjacent vertices (swap vertices). In addition, T_g requires that moving robots communicate and move simultaneously. The group transition system T_g is then pruned to produce a set of transition systems T_r, which have the property that all their runs are implementable by the robots in a distributed manner. Runs of T_r that satisfy the LTL specification are then projected to individual runs of T_i, which are then implemented in each robot.

Let Q_g be the set of all ordered n-tuples with elements from P with no repeated entries (see Definition 3). Let

$$\mathcal{M} : Q_g \times Q_g \rightarrow 2^{\{1, \ldots, n\}} \tag{4}$$

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

be a function such that $\mathscr{M}(q,q')$ gives the set of indices of agents that occupy different regions in configurations q and q' (e.g., if we have three agents, $\mathscr{M}((p_2,p_6,p_7),(p_2,p_7,p_5)) = \{2,3\}$). Obviously, function $\mathscr{M}$ can be easily computed for all its possible inputs.

Furthermore, we define

$$\mathscr{C} : Q_g \times \{1,\ldots,n\} \rightarrow 2^{\{1,\ldots,n\}} \tag{5}$$

where $\mathscr{C}(q,i)$ is the set of all agents communicating with robot i when the configuration of the team is q. As already mentioned, $\mathscr{C}(q,i)$ captures more than direct communication between robots (i.e., the robots can act as communication relays). The set $\mathscr{C}(q,i)$ can be computed for any $q \in Q_g$, $i \in \{1,\ldots,n\}$ by using the communication graph C (see Definition 2) as follows: First, we construct the adjacency matrix A_q of the subgraph of C induced by the vertices contained in q. Second, we find the largest connected component of this graph that includes robot i. This can be easily done by raising A_q at the $(n-1)$th power and finding the indices of nonzero elements on the ith row (this is true because every node has a self-loop, and since n is small, the involved computation is not time consuming).

Definition 3: The transition system $T_g = (Q_g, Q_{g0}, \rightarrow_g, \Pi_g, \models_g)$ capturing the behavior of the group of n agents is defined as follows.

1) $Q_g \subset Q_1 \times \ldots \times Q_n$, where $(q_1,\ldots,q_n) \in Q_g$ if and only if $q_i \neq q_j$ for $i \neq j$.
2) $Q_{g0} = (q_{0_1},\ldots,q_{0_n})$.
3) $\rightarrow_g \subset Q_g \times Q_g$ is defined by $(q,q') \in \rightarrow_g$, with $q = (q_1,\ldots,q_n)$, and $q' = (q'_1,\ldots,q'_n)$, if and only if
   a) $(q_i,q'_i) \in \rightarrow_i$, $i = 1,\ldots,n$;
   b) $\forall i,j = 1,\ldots,n$, with $i \neq j$, if $q'_i = q_j$, then $q'_j \neq q_i$;
   c) $\forall i \in \mathscr{M}(q,q')$, $\mathscr{M}(q,q') \subseteq \mathscr{C}(q,i)$.
4) $\Pi_g = P$.
5) $\models_g \subset Q_g \times \Pi_g$ is defined by $((q_1,\ldots,q_n),\pi) \in \models_g$ if $\pi \in \{q_1,\ldots,q_n\}$.

In other words, the states of the transition system T_g capture all possible ways in which the k vertices of G can be occupied by the n robots. The configurations in which two agents overlap (occupy the same vertex) are excluded, thus guaranteeing the avoidance of interrobot collisions. The possible motions of the team are modeled by the transition relation $\rightarrow_g$. A transition of T_g occurs when all agents synchronously take allowed transitions (see requirement a) of Definition 3), and we exclude the case when two agents swap positions, since this could cause collision [see requirement b) of Definition 3]. Moreover, requirement c) of Definition 3 shows that all moving agents should communicate. This last requirement is implied by the fact that only communicating agents can synchronize when changing their currently occupied locations. Note that the moving agents should communicate only while the team is in the configuration to be left (which is denoted by $q = (q_1,\ldots,q_n)$ in Definition 3), and they are not restricted to communicate when the next configuration [which is denoted by $q' = (q'_1,\ldots,q'_n)$] is reached. This is because the synchronization, when leaving configuration q, guarantees that the corresponding locations from q' are synchronously hit. In requirement c), $i \in \mathscr{M}(q,q')$ can be arbitrarily chosen, since $\forall i_1,i_2 \in \mathscr{M}(q,q')$, $\mathscr{C}(q,i_1) = \mathscr{C}(q,i_2)$, and we call the set $\mathscr{C}(q,i)$ as the *active communicating set*. Finally, each team configuration is equipped with n predicates enumerating the locations occupied by the agents (satisfied propositions), without explicitly specifying the exact position of each agent.

Remark 4: From Remark 1, it is enough to consider only words of T_g with no finitely many consecutive repetitions of an entry. Under this assumption, any word produced by T_g, from Definition 3, is a valid behavior that satisfies Definition 2 .

Remark 5: Some comments are in order regarding the "synchronization" in Definition 3. Since moving agents are required to communicate, then they can execute transitions synchronously. However, noncommunicating and communicating, but nonmoving, robots take self-transitions synchronously with the moving robots. For the latter category, this is not a problem, since they can communicate (and, therefore, synchronize their self-transitions) with the moving robots. Expecting noncommunicating robots to synchronously take transitions with the others is, of course, incorrect, since noncommunicating robots cannot synchronize. However, we use this definition of T_g for mathematical convenience. As it will become clear in Section V, self-transitions for noncommunicating robots will, in fact, mean "doing nothing." Robots will need to synchronize only when moving from a region to another.

Once we have T_g, and a specification, which is given as an LTL formula ϕ over P, we can try to find a solution to Problem 1 by using off-the-shelf model checking (see Section II). If a counterexample for $\neg\phi$ in the prefix–suffix form is produced, then we can collapse finitely possible many consecutive repetitions of a symbol in both prefix and suffix to produce a run r. Alternatively, as mentioned in Section II, we can use our model-checking tool to produce a run r that satisfies an optimality criterion, in addition to the two conditions mentioned earlier. Once a run r is determined through one of the two methods presented previously, it can be projected to the individual robots. The transition relation from T_g guarantees that, for every transition from r, the moving agents can communicate and, therefore, synchronize.

However, there is an additional restriction that the runs r should satisfy: In order for a moving group of robots to know when to take their next transitions, they need to know when the previously moving group completed their transition. For a better understanding, consider the following simple example: Assume that we have only two agents, and somehow, we obtained a run of T_g, $r = q,q',q'',\ldots$ [with $q = (q_1,q_2)$, $q' = (q'_1,q_2)$, and $q'' = (q'_1,q'_2)$], thus producing a word that satisfies ϕ. Note that the agents do not have to communicate in order to take each individual transition from run r, since only one of them moves at a time. However, once agent 1 completed its movement (i.e., transition) from q_1 to q'_1, agent 2 should be informed about this fact, in order to move from q_2 to q'_2. This is possible if and only if agents 1 and 2 were able to communicate while the team is in configuration (q'_1,q_2) (which is a fact that is not captured by the communication restrictions for individual transitions of T_g). We will refer to this requirement as *transitivity of communication*, thereby stating that a run of T_g can be decentralized if any

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

two successive active communicating sets have nonempty intersection. Formally, a run $r = r(1)r(2)r(3)\ldots$ of T_g satisfies the transitivity of communication property if, for any position $i = 1, 2, \ldots$, we have

$$\mathscr{C}(r(i), j) \cap \mathscr{C}(r(i+1), k) \neq \emptyset \quad \forall j \in \mathscr{M}(r(i), r(i+1)) \quad \forall k \in \mathscr{M}(r(i+1), r(i+2)). \tag{6}$$

One could try to use off-the-shelf model checking, as suggested earlier, and check whether the runs r, which are obtained as counterexamples, satisfy (6). However, this procedure might be time consuming, or it might not produce a result at all, since we do not have any control over the structure of the produced counterexamples. In addition, even if a satisfying run is found, it might be too "long," i.e., it might involve unnecessary robot transitions. Alternatively, if our model-checking approach is used, then one would have to find runs that satisfy (6) in the product automaton A (see Section II). However, it is not clear how to modify a standard model checker or our own model-checking approach such that requirement (6) is satisfied by the returned run. Therefore, in this paper, we propose a different approach.

We will cut transitions in T_g such that the obtained reduced transition system, which is called T_r, has only runs that satisfy (6). Then, by using our model-checking approach, which is described earlier, we will find runs that satisfy the specification ϕ. To obtain T_r from T_g, we proceed as follows. First, we find all the disjoint sets of communicating agents in the initial state Q_{g0}. For each of these sets, a different T_r can be created as follows: Consider one of these sets to be the active communicating set, which is denoted by $\mathscr{C}_0$, and start with $T_r = T_g$. From all outgoing transitions of Q_{g0} in T_r, keep only the self-loop and those for which the set of moving agents is included in $\mathscr{C}_0$. Then, for all the new states that can be visited by taking these transitions, keep only the self-loops and the outgoing transitions for which the intersection between their active communicating set and $\mathscr{C}_0$ is nonempty. The process is repeated for each newly visited state, by updating $\mathscr{C}_0$ to be the active communicating set for the currently considered outgoing transition. The algorithm finishes when no new (not yet visited) states are reached. In the end, the obtained T_r is guaranteed by construction to have only runs that satisfy property (6).

Once T_r is constructed, we search for a run r that satisfies formula ϕ by using our model-checking [15] procedure, which is described in Section II. If we find one, then this is also guaranteed to be a run of T_g. Indeed, since the set of transitions of T_r is a subset of the set of transitions of T_g, the language of T_r is included in the language of T_g. To choose among several possible satisfying runs, we select one that corresponds to a minimum overall number of robot movements (total number of transitions in prefix and suffix of all T_i, without counting self-loops). If a run cannot be found, we construct another T_r by considering another initial $\mathscr{C}_0$, and we search for a run in the new T_r. If we do not obtain a run for at least one T_r, we conclude that we cannot find a solution to Problem 1 by using this approach. Assuming a run is obtained, we have to implement it in a decentralized manner, as described in Section V.

Remark 6: We could have used a less conservative approach to obtain a run of T_g that satisfies the transitivity of communication property. In short, this approach would resemble counterexample-guided model checking and would consist of the iterative application of the following steps: 1) Obtain a run r of T_g that satisfies formula ϕ; 2) find sequences of transitions from r that do not satisfy the transitivity of communication property; and 3) modify ϕ such that it includes the avoidance of those sequences of states, and reiterate from 1). These steps should be iterated until a run that satisfies the desired property is obtained (or no run exists). The algorithm is guaranteed to terminate (because we keep adding restrictions to formula ϕ, and thus, we restrict the set of possible runs to be returned). However, such an approach would be computationally very expensive, because the augmented LTL formula might become very long.

A. Case Study Revisited

For our example shown in Fig. 1, the obtained T_g has 3360 states, and its initial state is $Q_{g0} = (p_1, p_3, p_{13})$. There are three communicating (singleton) sets in Q_{g0}, namely, $\{1\}$, $\{2\}$, and $\{3\}$ [e.g., T_g can transit from Q_{g0} to (p_1, p_4, p_{13}) but not to (p_2, p_4, p_{13})]. T_g has a total number of 55 344 transitions linking its 3360 states, and its construction took about 45 s on a medium performance laptop. The first reduced transition system T_r is constructed by considering $\mathscr{C}_0 = \{3\}$. The obtained reduced transition system T_r has only 28 583 transitions, and it was created in 64 s. A run that satisfies the formula was obtained (in 23 s), and thus, we do not have to create any other T_r. The obtained run is depicted in Fig. 2. It has a prefix of length 7, namely, the sequence of states (p_1, p_3, p_{13}) $(p_1, p_3, p_9)(p_1, p_3, p_5)$ (p_2, p_3, p_1) (p_3, p_7, p_2) $(p_7, p_{11}, p_3)(p_{11}, p_{12}, p_7)$, and a suffix consisting of infinitely many repetitions of configuration (p_{10}, p_{16}, p_8). By design, this run of T_g satisfies both the desired specification and the transitivity of communication property. Initially, agent 3 moves toward agent 1, then they start to communicate and both move toward robot 2, and eventually, all three agents reach configuration (p_{11}, p_{12}, p_7), from where they can simultaneously enter the yellow regions (p_8, p_{10}, p_{16}). Fig. 2 presents the successive configurations that are visited along the obtained run. The decentralized solution, showing what actions are performed by each robot such that the global behavior of the team is the one from Fig. 2, is presented in the next section. We mention that, in this case, a solution would be obtained no matter what set $\mathscr{C}_0$ was initially chosen to obtain T_r. For example, if we choose $\mathscr{C}_0 = \{1\}$, the obtained run has a prefix of length 9 and the same suffix, as mentioned earlier. The prefix shows that agent 1 moves toward agent 3, then they move together toward robot 2, and eventually, the team reaches configuration (p_{11}, p_{12}, p_7). In this case, there are more robot movements than in the case depicted by Fig. 2, because the obtained T_r is different. ■

V. Distributed Implementation

In this section, we show how any run r with a prefix–suffix structure of T_g obtained, as described in Section IV, can be implemented on the robots in a distributed manner, i.e., while

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

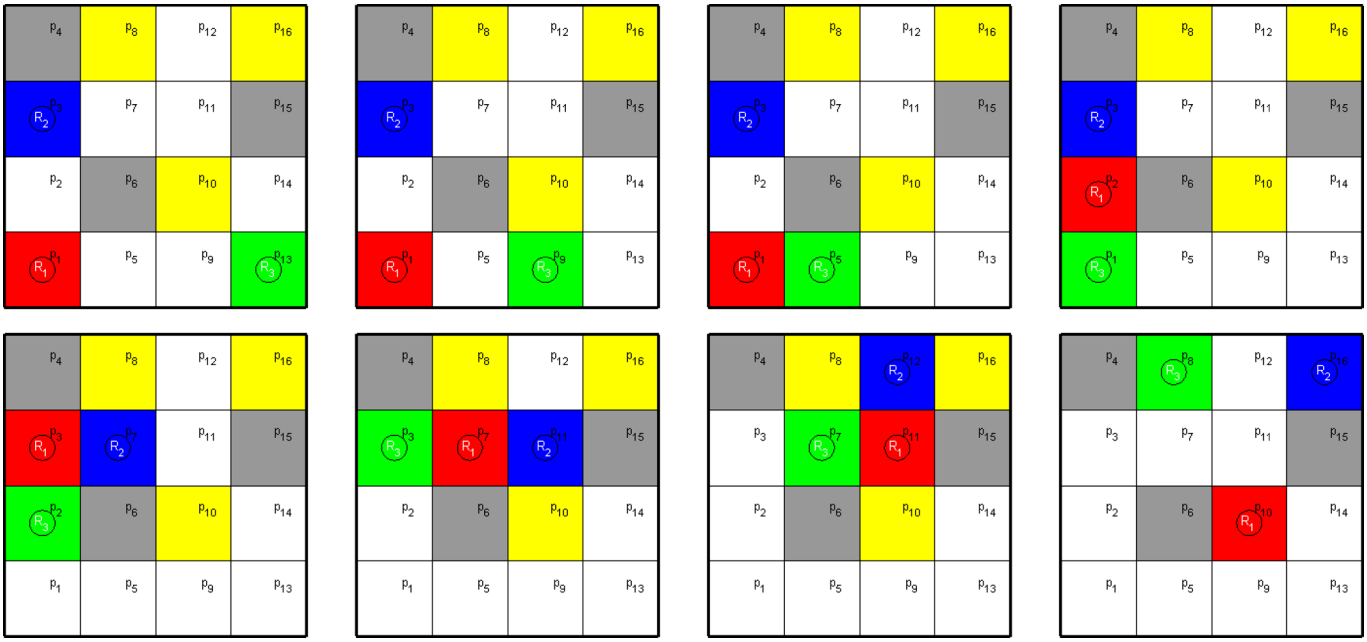

Fig. 2. Successive configurations of the team for the proposed example. The regions occupied by the three robots are in red, blue, and green, respectively. The regions to be simultaneously reached are in yellow (p_8, p_{10}, p_{16}), and the regions to be always avoided are in gray.

observing the communication constraints. The implementation will consist of a generic algorithm running on each robot. Given a run r, robot-specific inputs will be generated for the algorithm running on each robot. After the agents are programmed and placed in their initial vertices, the (local) executions of the algorithms will produce a team behavior, which coincides with the word produced by the run r on T_g.

Algorithm 1 presents the steps that are infinitely iterated by each robot. The main idea in the decentralized implementation of a run r is to have a *token* owned by a single robot at any given time. The robot owning the token initiates the next synchronous movement of some communicating robots (i.e., next transition in the run r). Thus, there will be no simultaneous movements of noncommunicating robots. Each robot continuously broadcasts its identity and the vertex it occupies in its communication range and listens for possible commands. Note that even the robot with token has to listen for commands, because it can send a command to itself. There are two types of commands that a robot can receive: a synchronized moving command that shows the next location to be reached and the group of robots with which to synchronize and a token command that shows which robot will next own the token. Unless a moving command is received, each agent remains in the current location.

The memory of each agent is organized as a first-in first-out (FIFO) queue. An entry in this queue stores the following information: *position*, which can be "prefix" or "suffix," shows if the current configuration of the team is in the prefix or the suffix of the run; $\mathscr{M} \subseteq \{1, \ldots, n\}$, which gives the set (identities) of moving robots for the next transition from r to be taken; $\mathscr{N}_p$ (an ordered tuple with $|\mathscr{M}|$ elements from P), which contains the next location that each robot from $\mathscr{M}$ should visit; and $\mathrm{next}_{\mathrm{tok}} \in \{1, \ldots, n\}$, which gives the index of the robot that will receive the token once the currently scheduled transition from r is taken. A robot reads the first entry of its queue each time it receives the token and then sends moving commands to robots from the current set $\mathscr{M}$, such that the next transition from r occurs. Once the transition is taken, the robot passes the token to robot $\mathrm{next}_{\mathrm{tok}}$ (which may be the same robot), and it updates the queue memory as follows: If $position = prefix$, the first entry (which is the one just treated) is deleted, and if $position = suffix$, the first entry is moved to the end of the queue (thereby implying infinitely many repetitions of suffix of r). The robot that initiated the transition detects its completion based on the following: Either it receives the new locations occupied by robots in $\mathscr{M}$, or if it can no longer communicate with some robots from $\mathscr{M}$, it concludes that those robots completed their moving command.

We find robot-specific inputs for Algorithm 1 by scanning the positions of run r. Because of the prefix–suffix structure of the run, we only have to scan a finite number of positions. The memory queues are created based on the schedule for robots receiving the token. Let $r(i)$, $i = 1, \ldots$ be a position from run r, and assume that $r(i) \neq r(i+1)$ and that $r(i+1) \neq r(i+2)$. This means that, if the suffix consists from a single infinitely repeating configuration, the current position $r(i)$ is at least two transitions before the suffix of run. For each $i = 1, \ldots$, we first choose the agent that will receive the token to initiate transition $r(i)$ to $r(i+1)$. This can be any agent from the set $\mathscr{C}(r(i), j) \cap \mathscr{C}(r(i+1), k)$, for arbitrary $j \in \mathscr{M}(r(i), r(i+1))$, and $k \in \mathscr{M}(r(i+1), r(i+2))$ (property (6) guarantees the nonemptiness of this set). In our implementation, the robot with the smallest index from the mentioned set is picked. Let us denote the identity of this robot by t_i. Using run r, we create the inputs *position* $\mathscr{M}$ and $\mathscr{N}_p$ to be added at the end of the queue memory of robot t_i. The value for $\mathrm{next}_{\mathrm{tok}}$ is determined when we find robot t_{i+1} initiating the next transition [from $r(i+1)$ to $r(i+2)$].

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

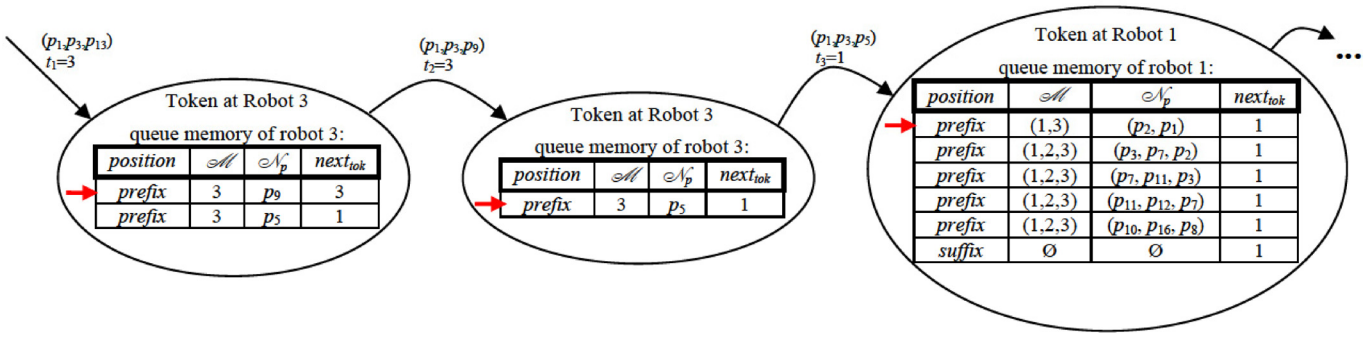

Fig. 3. Token passing and the queue memory of robots receiving the token. Edges between the big ellipses represent the team configuration when the token is passed and the robot receiving the token.

Algorithm 1 Operations iterated by each agent $i = 1, \ldots, n$

► listen and execute received commands
► broadcast index i and location occupied
if agent has *token* **then**
  ► read first entry from queue memory: *position*, $\mathscr{M}$, $\mathscr{N}_p$, $next_{tok}$
  ► send to agents from $\mathscr{M}$ the next location to be visited (corresponding position from $\mathscr{N}_p$), and set to synchronize with ($\mathscr{M}$)
  ► wait for agents from $\mathscr{M}$ to complete their task
  if *position* = *prefix* **then**
    ► delete first entry from queue memory
  else
    ► move first entry from queue memory to end
  end if
  ► send *token* to agent $next_{tok}$
end if

Note that, when the team reaches configuration $r(i+1)$, robot t_i communicates with t_{i+1}; therefore, it can pass the token $(t_i, t_{i+1} \in \mathscr{C}(r(i+1), k), k \in \mathscr{M}(r(i+1), r(i+2)))$. If run r has a suffix consisting from only one state, once we reached a position $r(i)$ that is only one transition far from the suffix, the token stays with the same robot ($t_{i-1} = t_i = t_{i+1}$). This is because the aforementioned approach would not work, since there are no moving robots from configurations $r(i+1)$ to $r(i+2)$ (suffix). Moreover, once such a suffix is reached, the sets $\mathscr{M}$ and $\mathscr{N}_p$ are empty in the last entry of the queue memory of agent t_i. In this case, only the token is passed to the same robot to ensure the correctness of Algorithm 1.

A. Case Study Revisited

Fig. 3 shows a global overview of the token passing among agents and the queue memory of each agent receiving the token. For simplicity, only the diagram guiding through the first three configurations of the team is presented. Initially, robot 3 has the token ($t_1 = 3$), since it is the first one to move, as already shown in Fig. 2. Once configuration (p_1, p_3, p_9) is reached, robot 3 deletes the first entry in its queue memory and sends the token back to itself ($t_2 = 3$). One can observe that agent 2 never receives the token; therefore, it has an empty queue memory (all its movements are initiated by agent 1). ■

VI. Conservatism and Complexity

Our approach to solving Problem 1 is conservative for two main reasons. First, we assume that only one group of communicating agents can move at a time, rather than allowing multiple noncommunicating groups of locally communicating agents to move simultaneously (an approach that allows us to relax this assumption is presented in Section VII). Second, by reducing T_g to T_r in Section IV, it is possible that no solution that satisfies property (6) can be found, even though such a solution might exist in T_g. Also, an optimal solution with respect to the number of robot movements in T_r does not guarantee optimality with respect to the same criterion for T_g.

From a complexity point of view, the bottlenecks of the presented approach are the construction of T_g, which has $k!/(k-n)!$ states, the reduction of T_g to T_r, and finding satisfying runs in T_r. Indeed, as stated in Section II, as a part of finding a satisfying run in T_g, we need to construct a Büchi automaton for the specification ϕ. This computation scales exponentially in the size of the LTL formula (i.e., number of propositions appearing in the formula).[1] This being said, note that all this expensive computation is performed offline, before the deployment of the robots. The algorithm that is run by each robot during the execution (see Algorithm 1) of the task is quite simple and requires a small amount of memory.

A. Example

To illustrate the increase of the computation time with the size of the problem, here, we consider a more complex example. We consider a team of four agents initially deployed in a partitioned environment, as shown in Fig. 4. The motion and communication capabilities are possible between rectangular regions sharing an edge (as shown in Section III-A). The task requires that "always avoid gray regions eventually visit the yellow regions and do not visit any green region until all of them are simultaneously entered," and it is translated to the LTL formula $\phi = \square\neg(p_7 \vee p_{15} \vee p_{22} \vee p_{25}) \wedge \Diamond(p_4 \wedge$

[1]This theoretical upper bound is rarely achieved in applications.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

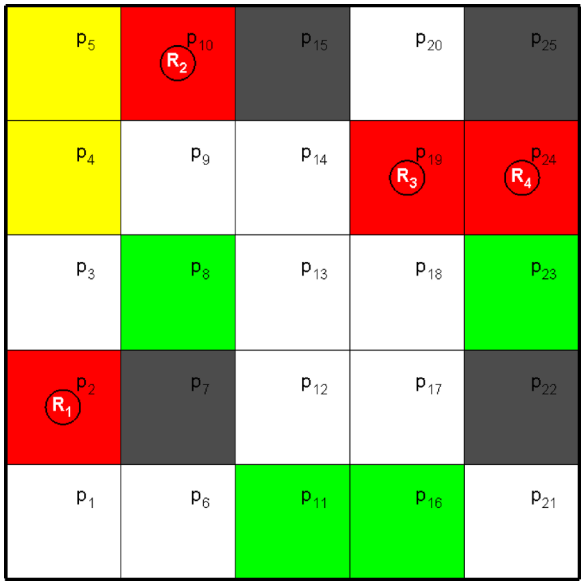

Fig. 4. Environment, regions of interest (in gray, yellow, and green), and initial deployment of agents (red-filled regions) for the example given in Section VI-A.

$p_5) \wedge \neg(p_8 \vee p_{11} \vee p_{16} \vee p_{23}) U (p_8 \wedge p_{11} \wedge p_{16} \wedge p_{23})$. Transition system T_g has 303 600 states. As compared with the previous case study, T_g has 1000 times more states, and the Büchi automaton has twice the number of states.

A run that satisfies the formula in a reduced transition system T_r corresponding to the situation in which the token was initially given to R_1 (see Fig. 4) is shown in Fig. 5. This run consists of ten different configurations of the team: the first configuration is the one from Fig. 4, and the last one is the suffix, which has length 1 and is repeated infinitely often. The overall computation took about 80 h, from which the construction of T_g, the construction of T_r, and finding a satisfying run in T_r took about 8%, 33%, and 58%, respectively. ■

It is important to note that a significant decrease in complexity can be achieved for the particular case when the robots are all identical and can all communicate with each other (they maintain a connected communication graph for all times). In such a case, the only difference between the transition systems T_i, $i = 1, \ldots, n$ is given by their initial states q_{0_i}, since they have identical motion capabilities $\rightarrow_i$. Requirement c) of Definition 3 becomes redundant, because the fully centralized communication architecture guarantees its fulfilment. In addition, there is no need to reduce T_g to T_r to guarantee property (6), because it is already satisfied by any run of T_g. In our previous work [27], by using bisimulation quotients, we show that the transition system T_g can be equivalently reduced to a system with $k!/((k-n)!n!)$ states. Another situation in which the amount of computation can decrease significantly is when no synchronization is necessary to accomplish the task, which is discussed in Section VII.

It is important to note that the speed of computation is not the emphasis in this study. All the given computation times, which are too large to be useful in a real deployment problem, should be interpreted as relative to one another, just to give an idea about how the methods scale with the size of the problem and how they compare with each other. Most importantly, note that we used a "classical" in-house model checker. A significant decrease in the computation time (which is of several orders of magnitude) can be obtained if a symbolic model checker is used [28]. Alternatively, a significant decrease can be obtained with on-the-fly LTL model checking [29] or by restricting the specifications to fragments of LTL for which more efficient algorithms exist. As already mentioned, all the computation was performed on a medium-performance laptop. The use of a more powerful computer would result in significantly reduced computation times. If several processors are available, the computation can be also accelerated through parallelizing the construction of the several possible T_r's. This being said, we expect that the method developed in this paper can only be relevant for "simple" applications (e.g., three robots moving in an environment with 20 features of interest).

VII. On the Need for Synchronization

As already emphasized, the main limitation of the approach presented previously is that robots can move (i.e., make progress toward the completion of the task) only if they can communicate. In this section, we present an approach that allows us to determine if the agents can satisfy the LTL task without communicating anytime they change locations. They will at most need to communicate locally to avoid collisions. The main idea is the following. First, we find a run of T_g that satisfies the LTL formula. Second, we find the set of all possible agent executions that, without synchronization, map to the satisfying run of T_g. This will be described as the language of a transition system. Then, we test if this language violates the LTL formula (by using our model-checking tool [15]). If there is no word that violates the formula, we conclude that synchronization is not necessary and the individual agent runs provide a solution to Problem 1. If the formula is violated by at least one word, then we decide that synchronization is necessary, and the approach that involves the reduction of T_g to T_r (see Section IV) has to be followed to find a solution. Alternatively, we could try to find another run of T_g that can be implemented by the team without the need for synchronization. This would follow the same lines, as described in Remark 6, but where step 2) would involve using the current run (the one that could not be implemented) to adjust T_g before producing a new candidate run. However, this extra step would add more computational overhead to an already expensive method, and we decided not to pursue it.

In the following, we describe the earlier ideas formally. Assume that by using the tool from [15] with inputs T_g and ϕ, we obtain a run r of T_g that starts from the initial state and satisfies the LTL formula ϕ. If no such run exists, we conclude that Problem 1 is unfeasible (by using T_r, as given in Section IV, we will also get no run, since the language of T_r is a subset of the language of T_g). Furthermore, for simplicity of exposition, we assume that the run r has a suffix of length one (if this is not the case, the definitions for Ts_i and T_p given shortly become more complicated).

If the run r is of form $r = (q_1^{(1)}, q_2^{(1)}, \ldots, q_n^{(1)})(q_1^{(2)}, q_2^{(2)}, \ldots, q_n^{(2)}) \ldots (q_1^{(s)}, q_2^{(s)}, \ldots, q_n^{(s)}) \ldots$, where state $(q_1^{(s)}, q_2^{(s)},$

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

Fig. 5. Successive configurations of the team for the example in Section VI-A (the initial configuration is shown in Fig. 4). The regions occupied by the four agents are in red. Gray regions should be avoided, yellow regions should be visited, and green regions should be entered simultaneously.

$\ldots, q_n^{(s)})$ is the suffix and is infinitely repeated, then the individual run of agent i is given by $r_i = q_i^{(1)}, q_i^{(2)}, \ldots, q_i^{(s)}, q_i^{(s)}, \ldots$, $i = 1, \ldots, n$. Next, in each individual run, we collapse all finite successive repetitions of the same symbol (except for the suffix, which consists of infinitely many repetitions) into a single occurrence (i.e., if $\exists i \in \{1, \ldots, n\}, j \in \{1, \ldots, s-1\}$ such that $q_i^{(j)} = q_i^{(j+1)}$, then we remove $q_i^{(j+1)}$ from r_i and repeat until no such i and j can be found). This is motivated by the fact that the agents do not synchronize while moving, and the removed repetitions correspond to synchronizations in run r modeling an agent that waits in the same location until other agents reach some specific locations. For simplicity, let each individual run (without successive repetitions) be denoted by $r_i = q_{i_1}, q_{i_2}, \ldots, q_{i_{s_i}}, \ldots$, where $q_{i_{s_i}}$ is infinitely repeated (the runs might now have prefixes of different lengths because of the performed collapsing).

Next, we construct the set of all possible n-tuples that can be generated by the team of agents, while each agent i individually (without synchronization) follows run r_i. For this, we first construct a small transition system Ts_i for each agent, and then, we construct a special kind of product automaton of all these transition systems, with the guarantee that the set of words

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

generated by the obtained product will be exactly the desired set of tuples.

Definition 4: The transition system $Ts_i = (Qs_i, Qs_{i0}, \rightarrow_{s_i}, \Pi_{s_i}, \models_{s_i})$, $i = 1, \ldots, n$, is defined as follows.

1) $Qs_i = \{q_{i_1}, q_{i_2}, \ldots, q_{i_{s_i}}\} \subset P$ is the set of states.
2) $Qs_{i0} = q_{i_1} = q_{0_i}$ is the initial location of agent i.
3) $\rightarrow_{s_i} \subset Qs_i \times Qs_i$ is defined by $(q_{i_j}, q_{i_j}) \in \rightarrow_{s_i} \forall j \in \{1, \ldots, s_i\}$, and $(q_{i_j}, q_{i_{j+1}}) \in \rightarrow_{s_i} \forall j \in \{1, \ldots, s_i - 1\}$.
4) $\Pi_{s_i} = P$.
5) $\models_{s_i} \subset Qs_i \times \Pi_{s_i}$ is the trivial satisfaction relation $(q, \pi) \in \models_{s_i}$ if and only if $q = \pi$.

Each transition system Ts_i, $i = 1, \ldots, n$, corresponds to agent i following run r_i (transitions exist only between successive states of r_i, together with self-loops in any state). The self-loops are included to correctly create the product transition system from Definition 5. Informally, agent i takes a self-transition if it is slower than other moving agents—this is necessary since there is no synchronization.

Definition 5: The product of Ts_i, $i = 1, \ldots, n$, is defined as $T_p = (Qp, Qp_0, \rightarrow_p, \Pi_p, \models_p)$, where the following conditions hold.

1) $Qp \subset Qs_1 \times \ldots \times Qs_n$ is defined by $(q_1, \ldots, q_n) \in Qp$ if and only if $q_i \neq q_j$ for $i \neq j$.
2) $Qp_0 = (q_{1_1}, \ldots, q_{n_1})$.
3) $\rightarrow_p \subset Qp \times Qp$ is defined by $(q, q') \in \rightarrow_p$, with $q = (q_1, \ldots, q_n)$, and $q' = (q'_1, \ldots, q'_n)$, if and only if
   a) $(q_i, q'_i) \in \rightarrow_{s_i}$, $i = 1, \ldots, n$;
   b) $q \neq q'$, or $q = q' = (q_{1_{s_1}}, \ldots, q_{n_{s_n}})$.
4) $\Pi_p = P$.
5) $\models_p \subset Qp \times \Pi_p$ is defined by $((q_1, \ldots, q_n), \pi) \in \models_p$ if $\pi \in \{q_1, \ldots, q_n\}$.

Transition system T_p captures all possible transitions that can appear in any Ts_i, i.e., all possible configurations that can be attained by the team, while each robot follows its run on its own (without synchronization). This fact is guaranteed by requirement a) (which is given in Definition 5) from the transition relation of T_p. As already mentioned, the self-loops at all states of all Ts_i allow some agents to remain in their locations (take self-loops), while other agents move from one location to another. Requirement b) (which is given in Definition 5) from the transition relation of T_p eliminates self-loops in all states of T_p, except for the state corresponding to suffixes of runs r_i. This is because we use LTL_{-X} formulas (see Remark 1), and there is no need to capture finitely many successive repetitions of the same tuple in the generated word.

Note that in the construction of Qp in Definition 5, we exclude the situations in which two or more agents occupy the same location, thus achieving collision avoidance in the formalism that we use. However, to satisfy this requirement when implementing an obtained strategy, the agents must have some local collision-avoidance controllers. More specifically, all agents must to be able to sense if the next (adjacent) location is free before moving from the current location. If the next location is occupied, the agent needs to wait until the location becomes free. This sensing capacity can, of course, be implemented if the agents can communicate when in adjacent regions. Since there is no synchronization when different agents change locations, the probability of two agents taking transitions to exactly the same location at exactly the same time is zero, and thus, the described simple protocol guarantees collision avoidance. Such a protocol was not necessary in the approach from Sections IV and V, since collision avoidance was guaranteed by the global construction of the team run. Finally, note that the "move-only-when-communicate" restriction from the run r of T_g disappears as a result of collapsing identical successive repetitions from its projections to individual agents.

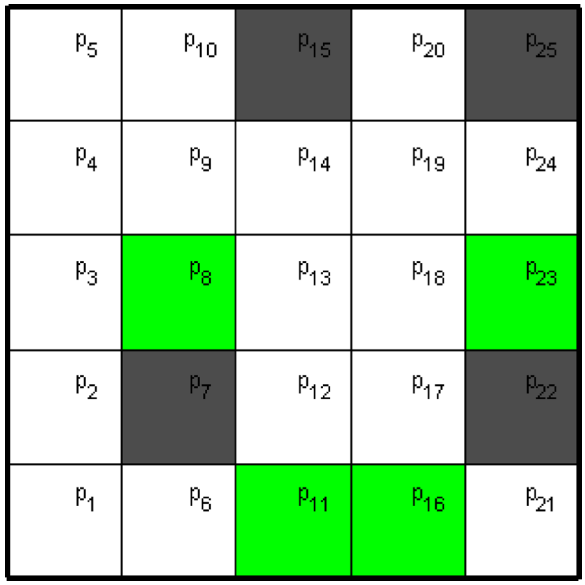

Fig. 6. Partition for the example given in Section VII-A.

By construction, the language of T_p contains all and only the words that can be generated by the unsynchronized team movement when each agent i follows its run r_i, $i = 1, \ldots, n$. If the language of T_p satisfy formula ϕ, then we conclude that the unsynchronized movement (with collision avoidance, as mentioned earlier) is a solution to Problem 1. If this is not the case, we conclude that synchronization is necessary, and therefore, the approach from Sections IV and V is necessary.

A. Example

We consider a team of four agents evolving in the environment from Fig. 6. The agents are initially placed in locations p_2, p_{10}, p_{19}, and p_{24} (see Section VI-A), and the motion and communication capabilities are as given Section III-A (i.e., motion and communication are possible between rectangles sharing an edge). The task is "always avoid gray regions and, eventually, occupy all the green regions at the same time (but not necessarily simultaneously)," and it can be translated to the LTL formula $\phi = \Box\neg(p_7 \vee p_{15} \vee p_{22} \vee p_{25}) \wedge \Diamond(p_8 \wedge p_{11} \wedge p_{16} \wedge p_{23})$. Transition system T_g is the same, as given in Section VI-A (it has 303 600 states). However, note that this task is different from the task given in Section VI-A (even ignoring the yellow regions), because here, the agents may enter the green regions at different time instants and in any order, whereas in the example given in Section VI-A, the robots were required to enter the green regions simultaneously.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

If we use the approach from Sections IV and V, with token initially given to agent 1, the overall computational time is about 50 h from which the construction of T_g, the construction of T_r, and finding a satisfying run in T_r takes about 14%, 55%, and 30%, respectively. Note that these numbers are different from those given in Section VI-A, because the LTL task is different. The decentralized solution consists of eight different team configurations (first, agent 1 moves to locations p_3, p_4, and p_9 and establishes communication with agent 2; then, they synchronously move to locations p_{14} and p_9, respectively, and then, the entire team communicates and synchronously moves through configurations $(p_{14}, p_9, p_{19}, p_{24})$, $(p_{13}, p_8, p_{18}, p_{19})$, $(p_{12}, p_{13}, p_{17}, p_{18})$, and $(p_{11}, p_8, p_{16}, p_{23})$ by remaining in the last configuration).

If, instead of computing T_r, we use the approach from this section in order to decide if synchronization is necessary, the overall time is 34 h. The run has eight states in prefix and one in suffix, namely, $(p_2, p_{10}, p_{19}, p_{24})$, $(p_1, p_{10}, p_{19}, p_{24})$, $(p_1, p_9, p_{19}, p_{24})$, $(p_1, p_8, p_{19}, p_{24})$, $(p_1, p_8, p_{18}, p_{23})$, $(p_1, p_8, p_{17}, p_{23})$, $(p_1, p_8, p_{16}, p_{23})$, $(p_6, p_8, p_{16}, p_{23})$, and $(p_{11}, p_8, p_{16}, p_{23})$ (the last configuration being the suffix). By projecting this run to individual runs and by collapsing the identical successive repetitions, we obtain $r_1 = p_2, p_1, p_6, p_{11}$, $r_2 = p_{10}, p_9, p_8$, $r_3 = p_{19}, p_{18}, p_{17}, p_{16}$, and $r_4 = p_{24}, p_{23}$. The transition system T_p has 96 states, and by using the model-checking approach, we conclude that all its generated words satisfy the formula. Thus, each agent can individually follow its run, without synchronizing with other agents (note that in this example, no collision is possible). All steps after finding run r took about 5 s.

By using the approach from this section for the case studies given in Sections III and VI, we obtain that synchronization is necessary, which is in accordance with the nature of the LTL tasks (they both require to simultaneously enter some regions). The involved computation time was about 30 s for the example given in Section III-A and 49 h for the example given in Section VI-A, from which more than 99% was used to find a run of T_g. ■

To conclude, our proposed approach to Problem 1 is a two-step process. First, we check if synchronization is necessary by using the approach proposed in this section. If the answer is negative, then a solution is automatically generated. In this solution, the robots do not have to wait for each other, except for collision avoidance. If the answer is positive, then the (more expensive) approach that is described in Sections IV and V can provide a solution to the problem. Note that there is still a gap in between the two approaches and we do not accommodate cases in which several noncommunicating groups of communicating robots move independently and communicate only when necessary. While this gap could be reduced by extending the ideas from this section to groups of robots, this would add some computation overhead to an already computationally expensive framework.

VIII. Deployment of Robots With Continuous Dynamics and Experimental Results

As already suggested, by using results on the construction of finite quotients for continuous and hybrid systems [11], [13], [15], the framework developed here for finite models of robot motion (i.e., finite-transition systems) can be directly used to deploy robots described by "realistic" continuous dynamical models. The underlying idea is a certain notion of "equivalence" between an infinite control system ($\dot{x} = f(x, u)$), which describes the motion of a robot in a partitioned environment, and a finite-transition system, which describes the motion of a robot between the vertices of a graph. The graph on which the transition system evolves is the partition quotient of the environment. The transitions $\rightarrow_i$ of each T_i are constructed based on computationally efficient procedures to design feedback controllers in each region such that either the region is left in finite time through a desired facet, or the region is an invariant for the trajectories of the continuous control system [12], [13]. If only such feedback controllers are used for the continuous control system, then the finite-transition system is a bisimulation quotient for an infinite-transition system embedding the closed-loop continuous control system.

Once a finite representation of the form (3) is constructed for each robot, the discrete strategy, which is presented in this paper, can be almost directly applied to the initial robots described by continuous control systems. Indeed, a "discrete" transition between two adjacent vertices p_1 and p_2 corresponds to the application of a feedback controller driving all initial states of the continuous system from the region labeled by p_1 to the region labeled by p_2 through their separating facet in finite time. Robots that are not required to move in the discrete scenario apply feedback controllers making the corresponding regions invariants. Of course, the motion is not instantaneous anymore in the continuous case. The synchronization will occur on the boundaries of the regions when crossing from a region to another, and the synchronizing robots will have to "wait" for each other on the boundaries.

In the rest of this section, we present experimental results showing the automatic deployment of two Khepera III robots ( ) in a 80 in $\times$ 60 in rectangular environment with polygonal regions of interest, such as targets and obstacles (see Fig. 7). Each robot is a differential drive, and it is about the size of a CD. A central computer performs all the computation and sends the individual plans (runs) to the robots through Bluetooth. An overhead video camera is used for robot localization. The robots are covered with white disks, which contain certain dot patterns for robot identification. When computing the robot-control inputs, each robot is abstracted to its observable, which is an off-axis point on the white disk. To design desired velocities for the reference point, we use triangulations and construction of affine vector fields, as described in [14]. To generate velocities for the robot wheels, we use a unicycle model and solve an input–output regulation problem, as proposed in [30]. We assume that the robots can communicate for all times. Since they are also identical, the offline computation before deployment is relatively cheap by using the reduction, as described at the end of Section VI.

We consider a task specification given by the following LTL formula:

$$\phi = \neg(\text{blue}_{\text{triangle}} \vee \text{blue}_{\text{pentagon}} \vee \text{red} \vee \text{green}) \mathscr{U} (\text{red} \wedge \text{green} \wedge \diamondsuit(\text{blue}_{\text{triangle}} \vee \text{blue}_{\text{pentagon}})). \quad (7)$$

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

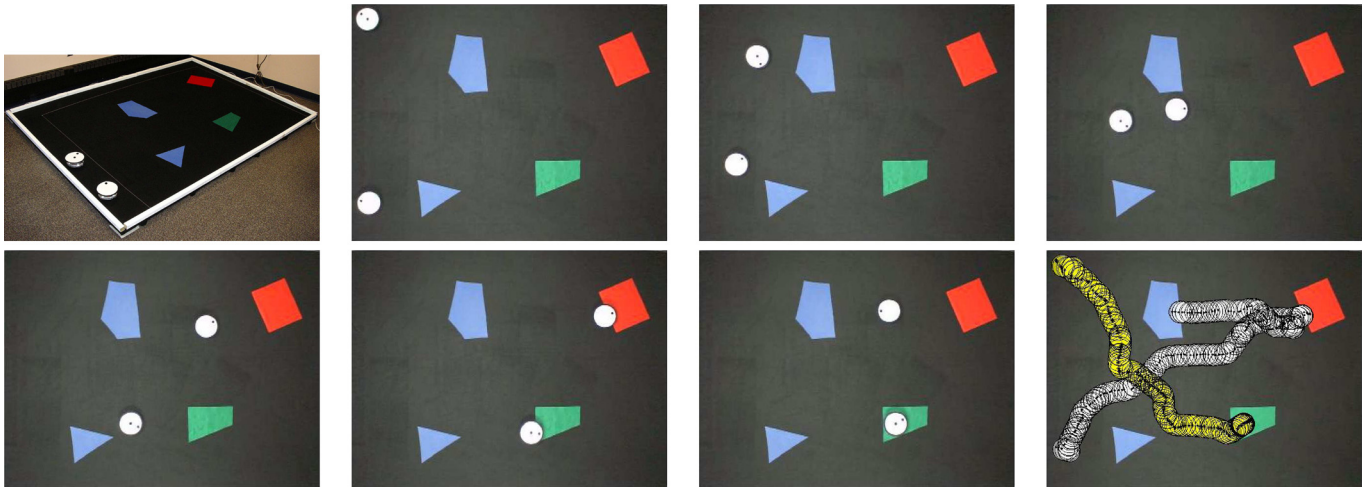

Fig. 7. Experimental results for the deployment of two robots in a rectangular environment with polygonal regions of interest. (Left to right and top to bottom) (First figure) Testbed. (Second figure) Initial deployment of the two robots, while the next ones show snapshots of their motion. (Last figure) Motion traces (white for the first robot and yellow for the second).

Formula ϕ requires that all regions be avoided until the green and red regions are simultaneously entered and, after that, either of the blue regions be visited. Several snapshots taken with the overhead video camera during the motion of the robots, as well as motion traces of both robots, are presented in Fig. 7. The movies corresponding to this experiment can be downloaded from ∼ or from the multimedia attachment of this paper.

IX. Discussion and Future Work

We presented a fully automated computational framework for decentralized deployment of a team of robots from task specifications given in high-level and rich language consisting of temporal-logic statements about visiting regions of interest in a partitioned environment. While we focused on a purely discrete problem, the implementation of the algorithm is immediate for robots with continuous dynamics moving in environments with triangular or rectangular partitions and with communication constraints induced by physical proximity. The approach was implemented as a software package under MATLAB, which is freely downloadable from ∼, and tested in an experimental platform based on Khepera III.

While allowing for provably correct distributed implementations from rich, human-like motion specifications, our approach has five main disadvantages. First, it is very expensive. This is mainly due to the construction of the synchronous product capturing the motion of the team. To deal with this issue, in the future, we will pursue a hierarchical approach in which the global specification will be first mapped to local (abstract) behaviors (following the ideas reviewed in [22]). The latter will serve as motion specifications for each agent. However, even in this approach, we expect that the applicability of these types of techniques will be restricted to "simple" environments and small teams of robots with significant computation capabilities. Second, our approach is not completely decentralized in the sense that the solution is found in a "centralized" manner and is executed in a distributed manner. Of course, it would be desirable to have a completely decentralized solution in which the global task is given to all agents and they find the distributed solution through a negotiation process during the execution of the task. We expect that "on-the-fly" model-checking algorithms [29] for distributable temporal logics will be relevant to this issue.

Third, our approach is conservative, or incomplete, in the sense that even if a solution exists, we might fail to find it. As already stated, we could approach this issue through some extra computation. However, this would add to an already expensive algorithm. Fourth, it cannot accommodate changes in the environment and external events, and it is not robust to failures, e.g., token loss, incorrect messages, etc. A reactive approach [21] can be used to accommodate "well-behaved" external events, as recommended in [19]. Mission changes could be handled by restricting the specification to flat LTL [31], as recommended in [32].

Fifth, the method proposed here captures no measurement and control uncertainty. Indeed, we assume that the robots can determine exactly when crossing from one region to another and what happens after the application of a control action, i.e., the transition to the next region is guaranteed. Relaxing these assumptions leads to the problem of controlling a partially observed Markov decision process (POMDP) from a temporal-logic specification, which is currently not well-understood. Note that control strategies for MDP from formulas of probabilistic temporal logic can be, in principle, obtained by adapting existing probabilistic model checkers, such as PRISM [33].

Acknowledgment

This paper contains a new algorithm that allows for distributed executions with no communication. In addition, it contains revised notation, definitions, more technical details, examples, and experimental results.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] M. Kloetzer and C. Belta, "Distributed implementation of global temporal logic motion specifications," in *Proc. IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, 2008, pp. 1–6.

[2] J. C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.

[3] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston, MA: MIT, 2005.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ., 2006.

[5] V. Gazi and K. M. Passino, "Stability analysis of swarms," *IEEE Trans. Autom. Control*, vol. 48, no. 4, pp. 692–697, Apr. 2003.

[6] N. E. Leonard and E. Fiorelli, "Virtual leaders, artificial potentials, and coordinated control of groups," in *Proc. 40th IEEE Conf. Decis. Control*, Orlando, FL, Dec. 2001, pp. 2968–2973.

[7] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Trans. Autom. Control*, vol. 48, no. 6, pp. 988–1001, Jun. 2003.

[8] R. Olfati-Saber and R. M. Murray, "Distributed cooperative control of multiple vehicle formations using structural potential functions," presented at the IFAC World Congr., Barcelona, Spain, Jul. 2002.

[9] E. M. M. Clarke, D. Peled, and O. Grumberg, *Model Checking*. Cambridge, MA: MIT, 1999.

[10] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Berlin, Germany: Springer-Verlag, 1992.

[11] L. Habets and J. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, pp. 21–35, 2004.

[12] L. Habets, P. Collins, and J. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Trans. Autom. Control*, vol. 51, no. 6, pp. 938–948, Jun. 2006.

[13] C. Belta and L. Habets, "Control of a class of nonlinear systems on rectangles," *IEEE Trans. Autom. Control*, vol. 51, no. 11, pp. 1749–1759, Nov. 2006.

[14] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot planning and control in polygonal environments," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 864–874, Oct. 2005.

[15] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. Autom. Control*, vol. 53, no. 1, pp. 287–297, Feb. 2008.

[16] M. Antoniotti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1995, pp. 1441–1446.

[17] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *Proc. 43rd IEEE Conf. Decis. Control*, Dec. 2004, pp. 153–158.

[18] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot motion planning: A timed automata approach," in *Proc. IEEE Int. Conf. Robot. Autom.*, New Orleans, LA, Apr. 2004, pp. 4417–4422.

[19] H. K. Gazit, G. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *Proc. IEEE Conf. Robot. Autom.*, Rome, Italy, 2007, pp. 3116–3121.

[20] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proc. IEEE Conf. Decis. Control*, Seville, Spain, Dec. 2005, pp. 4885–4890.

[21] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive(1) designs," in *Proc. VMCAI*, Charleston, SC, 2006, pp. 364–380.

[22] M. Mukund, "From global specifications to distributed implementations," in *Synthesis and Control of Discrete Event Systems*. Norwell, MA: Kluwer, 2002, pp. 19–34.

[23] R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems*. Boston, MA: Kluwer, 1995.

[24] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv version 2: An open-source tool for symbolic model checking," in *Proc. Int. Conf. Comput. Aided Verification* (Lecture Notes in Computer Science 2404). Copenhagen, Denmark: Springer-Verlag, Jul. 2002.

[25] G. Holzmann, "The model checker spin," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 279–295, May 1997.

[26] G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*. Reading, MA: Addison-Wesley, 2004.

[27] M. Kloetzer and C. Belta, "LTL planning for groups of robots," in *Proc. IEEE Int. Conf. Netw., Sens., Control*, Ft. Lauderdale, FL, 2006, pp. 578–583.

[28] J. R. Burch, E. M. Clarke, K. L. Mcmillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10 20 states and beyond," *Inf. Comput.*, vol. 98, no. 2, pp. 142–170, 1990.

[29] J. Geldenhuys and A. Valmari, "More efficient on-the-fly LTL verification with Tarjan's algorithm," *Theor. Comput. Sci.*, vol. 345, no. 1, pp. 60–82, 2005.

[30] J. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, Leuven, Belgium, 1998, pp. 2864–2869.

[31] D. Dams, "Flat fragments of CTL and CTL?: Separating the expressive and distinguishing powers," *Logic J. IGPL*, vol. 7, no. 1, pp. 55–78, 1999.

[32] G. Fainekos, S. Loizou, and G. J. Pappas, "Translating temporal logic to controller specifications," in *Proc. IEEE Conf. Decis. Control*, San Diego, CA, 2006, pp. 899–904.

[33] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "Prism: A tool for automatic verification of probabilistic systems," in *Proc. 12th Int. Conf. Tools Algorithms Constr. Anal. Syst.* (Lecture Notes in Computer Science 3920). New York: Springer-Verlag, 2006, pp. 441–444.

Marius Kloetzer received the B.S. and M.Sc. degrees in computer science from the Technical University of Iasi, Iasi, Romania, and the Ph.D. degree in systems engineering from Boston University, Boston, MA.

He is currently an Assistant Professor with the Technical University of Iasi. His current research interests include symbolic motion planning, discrete abstractions, linear temporal logic, and hybrid systems.

Calin Belta (M'03) received the B.S. and M.Sc. degrees in control and computer science from the Technical University of Iasi, Iasi, Romania, the M.Sc. degree in electrical engineering from Louisiana State University, Baton Rouge, and the M.Sc. and Ph.D. degrees in mechanical engineering from the University of Pennsylvania, Philadelphia.

He is currently an Assistant Professor with Boston University, Boston, MA. He is an Associate Editor for the *Society for Industrial and Applied Mathematics Journal on Control and Optimization*. His current research interests include analysis and control of hybrid systems, motion planning and control, and biomolecular networks.

Dr. Belta is an Associate Editor for the Robotics and Automation Society (RAS) and Control Systems Society (CSS) Conference Editorial Boards. He received the best Poster Award at the International Conference on Systems Biology in 2004 and was a finalist for the American Society of Mechanical Engineers (ASME) Design Engineering Technical Conference Best Paper Award in 2002 and for the Anton Philips Best Student Paper Award at the IEEE International Conference on Robotics and Automation in 2001. He received the Air Force Office of Scientific Research Young Investigator Award in 2008 and the National Science Foundation CAREER Award in 2005.

Authorized licensed use limited to: BOSTON UNIVERSITY. Downloaded on December 30,2023 at 00:34:48 UTC from IEEE Xplore. Restrictions apply.