Decentralized Multi-Agent Control from Local LTL Specifications

Ioannis Filippidis, Dimos V. Dimarogonas and Kostas J. Kyriakopoulos

Abstract— **We propose a methodology for decentralized multi-agent control from Linear Temporal Logic (LTL) specifications. Each agent receives an independent specification to formally synthesize its own hybrid controller. Mutual satisfiability is not a priori guaranteed. Due to limited communication, the agents utilize meeting events to exchange their controller automata and verify satisfiability through model checking. Local interaction only when common atomic propositions exist reduces the overall computational cost, facilitating scalability. Provably correct collision avoidance and convergence is ensured by Decentralized Multi-Agent Navigation Functions.**

I. INTRODUCTION

There have been multiple approaches to multi-agent control. Both classic motion planning [1], [2], [3] and task related methods [4], [5] have been developed. The current effort targets unification [6], [7], [8], [9], [10], [11], [12], [13]. Since we currently deal with increasingly complex and heterogeneous systems, decentralization is desired for scalability, while safe and guaranteed results are required. Formal methods for specification and automatic synthesis of provably correct controllers can ensure this. The system's specification can be provided in a logic sufficiently expressive for the desired tasks.

In [13] centralized multi-agent systems with perfect information are considered. Synthesis of a centralized controller is performed from a global LTL specification. This requires a globally connected multi-agent system to ensure information availability. Necessary and sufficient conditions for a global specification to be decomposable to bisimilar local ones are derived in [11]. This provides a distributed method from top to bottom. Execution with no communication limitations is assumed. In [14] the issue of communication and synchronization is analyzed and a solution is proposed, which diagnoses whether an LTL specification needs communication or not. Similarly to [15], where only communicating agents are allowed to move, this check characterizes the subset of realizable specifications.

We extend here the application of formal methods to decentralized multi-agent systems. The method proposed enables each agent to independently synthesize safe controllers, verify its plans versus those of others upon meeting them and execute them in a continuous state space using Navigation Functions. The main innovations with respect to previous works are that tasks are defined independently in a bottom up manner, there is no global specification for all agents, the verification is decentralized and the motion planning controllers are NFs.

In particular, LTL specifications provided to the agents are not produced in a centralized way, hence they may be contradicting each other. The solution proposed for this aims at gradually verifying that agent specifications are mutually satisfiable. Events of path-connectedness enable exchange of their alphabets and automata, to allow model checking [16]. Moreover, even if mutually satisfiable, we are interested in cases in which long-range communication is not available. If path-connectivity is absent when required by agents, the controllers will fail to act according to their specifications, due to lack of information. We embed in LTL communication requests when information is needed and implement them using additional follower agents under connectivity maintenance control.

For interfacing the discrete controllers to the continuous system state we choose Navigation Functions (NFs) [17], [2]. NFs are continuous feedback motion planning control laws [3] which ensure collision avoidance and convergence. As a result, the specification is formally satisfied in the discrete control level, which in turn is interfaced to the continuous state space via provably correct NF controllers.

The rest of this paper is organized as follows: preliminaries are covered in § II, the problem defined in § III, the layered architecture is described in § IV, decentralized verification in § V, and the method illustrated by simulation in § VI. Concluding remarks are summarized in § VII where future research is considered.

II. PRELIMINARIES

A. *Linear Temporal Logic*

LTL is an extension of propositional logic suitable for reasoning about infinite sequences of states [18]. Let P be a set of Atomic Propositions (APs) [16]. More complex formulae result from combining propositional ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, i.e., negation, conjunction, disjunction, implication, equivalence) with temporal ($\mathscr{U}, \Box$ and $\Diamond$, i.e., until, always, eventually) operators. Formula $\phi_1 \mathscr{U} \phi_2$ requires that ϕ_1 be true *until* ϕ_2 becomes true, which is required to happen. Formula $\Diamond\phi$ requires that ϕ be true at some future time. Its dual $\Box\phi$ requires that ϕ be true at all future times.

The semantics of LTL are defined with respect to sequences $\sigma : \mathbb{N} \rightarrow 2^P$. A formula is evaluated over σ by starting its interpretation from $\sigma(0)$. Let Φ_P denote the set of

Ioannis Filippidis and Kostas J. Kyriakopoulos are with the Control Systems Lab, Department of Mechanical Engineering, National Technical University of Athens, 9 Heroon Polytechniou Street, Zografou 15780, Greece. E-mail: jfilippidis@gmail.com, kkyria@mail.ntua.gr, Dimos V. Dimarogonas is with the ACCESS Linnaeus Centre, School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden. He is also affiliated with the KTH Centre for Autonomous Systems and is supported by the Swedish Research Council (VR) through contract 2009-3948. E-mail: dimos@kth.se

well-formed formulas [19] over P, $p \in P$ and $\phi_1, \phi_2 \in \Phi_P$. By $\sigma \vDash \phi$ we mean that ϕ is true when evaluated over σ.

- For all σ it is $\sigma \vDash$ true and $\sigma \nvDash$ false;
- $\sigma \vDash p$ if and only if (iff) $p \in \sigma(0)$;
- $\sigma \vDash \neg p$ iff $p \notin \sigma(0)$;
- $\sigma \vDash \phi_1 \wedge \phi_2$ iff $\sigma \vDash \phi_1$ and $\sigma \vDash \phi_2$;
- $\sigma \vDash \phi_1 \vee \phi_2$ iff $\sigma \vDash \phi_1$ or $\sigma \vDash \phi_2$;
- $\sigma \vDash \phi_1 \mathscr{U} \phi_2$ iff $\exists i \in \mathbb{N} : \sigma^i \vDash \phi_2$ and $\sigma^j \vDash \phi_1$ for all $j < i$, where $\sigma^m(k) \triangleq \sigma(m+k)$ for all $k \in \mathbb{N}$.

B. Navigation Functions

Let $x_i \in \mathbb{R}^n, i \in I \subset \mathbb{N}$ be the continuous states of a set of agents indexed by I. Navigation Functions (NF) are potential fields free of local minima introduced in [17]. NFs were extended to decentralized multi-agent systems in [2]. Connectivity maintenance constraints for NFs were proposed in [20]. Each agent is holonomic, $\dot{x}_i(t) = u_i(t)$. The control input u_i is chosen as $u_i \triangleq -\nabla_{x_i}\varphi_i\left(x(t), x_{di}\right)$ where $\varphi_i \triangleq \frac{\gamma_i}{(\gamma_i^k + G_i)^{\frac{1}{k}}}$ is the NF of agent i and $x(t)$ is the stack vector of all x_i. Function φ_i is maximal in collision sets and has a unique minimum at x_{di}.

Let N_i be the set of neighbors to which agent i should remain connected. The destination function $\gamma_i \triangleq \sum_{j \in N_i} \|x_i - x_j\|^2$ is used later for followers and $\gamma_i \triangleq \|x_i - x_{di}\|^2$ for leaders, described in § III-A. The destination x_{di} for a leader is selected by the discrete control layer. Function G_i measures the proximity between agents, in order to avoid collisions. Also, if connectivity between two agents is maintained, then G_i prevents them from separating. For its definition, please refer to [20], [2].

In order for ϕ_i to acquire the properties of a NF, the parameters k, λ and h should be larger than thresholds which depend on each problem, as proved in [20]. Leaders constrained by connectivity can reach their destinations when sufficiently many followers are available to connect them over that distance.

III. PROBLEM DEFINITION

A. Agent Continuous and Discrete States

We consider a set of $N \in \mathbb{N}^* \triangleq \mathbb{N}\backslash\{0\}$ agents, indexed by $I_a \triangleq \mathbb{N}^*_{\leq N}$. Each agent a_i is characterized by a continuous state $x_i \in X_i \subseteq \mathbb{R}^{n_i}$ where $n_i \in \mathbb{N}^*$. In this work the continuous state is the position of each agent on the Euclidean plane, $X_i = \mathbb{R}^2$. In addition, it may have a discrete state $q_i \in Q_i$ associated with it, where $Q_i \subseteq \mathbb{N}^{m_i}$ and $m_i \in \mathbb{N}$. For example, an agent may carry a red beacon described by the discrete state q_1. The beacon can be either on or off, so $q_1 \in Q_1 = \{0, 1\}$. The combination of continuous and discrete state forms a hybrid state $H_i = x_i \times q_i$.

Each agent is either a leader l_i, or a follower f_i. Each leader receives its own LTL specification ϕ_i. Followers do not receive specifications. There are N_l leaders indexed by $I_l \subseteq I_a$ and N_f followers indexed by $I_f \triangleq I_a \setminus I_l$.

Define the ball $B_\delta(x_0) \triangleq \{x \in \mathbb{R}^n | \ \|x - x_0\| < \delta\}$ with center x_i and radius $\delta > 0$. We assume that each agent occupies $B_{\rho_i}(x_i)$, where $\rho_i > 0$.

If $x_j \in B_{R_s}(x_i)$, then agents a_i and a_j can communicate *directly*, Fig. 3a. We call R_s the communication radius. Agents a_i and a_j are *path-connected* if there exists a chain connecting them, comprised of pairs of directly connected agents, Fig. 3b. This chain relays information between a_i and a_j, so they can communicate.

B. Atomic Propositions observing the Continuous State

We are now going to discuss the problem's modeling. We want to specify the movement of leaders between points using LTL. To "speak" in terms of "point" entities, the proximity to each point y is expressed by an AP associated with y. Let p^o_{ij} be the j^{th} observer AP of agent a_i, observing the state of agent a_k, then

$$p^o_{ij} \triangleq \begin{cases} \text{true}, \|x_k - y\| < \varepsilon \\ \text{false}, \|x_k - y\| \geq \varepsilon \end{cases} \quad i, k \in I_l,\ y \in X_k \qquad (1)$$

where $\varepsilon > 0$ is some constant such that $\varepsilon < R_s$ and y is some point of interest to a_i. For sufficiently small ε at most one observer can be true at any time t. Let $P^o_i \triangleq \left\{p^o_{ij}\right\}_{j \in I^o_i}$, be the observer APs of agent a_i, where $I^o_i \triangleq \mathbb{N}_{\leq n^o_i}$ and $n^o_i \in \mathbb{N}$.

Alternatively, an observer may be defined to measure the distance between the continuous states of two different agents a_i and a_k, with $i \neq k$

$$p^o_{ij} \triangleq \begin{cases} \text{true}, \|x_i - x_k\| < \varepsilon \\ \text{false}, \|x_i - x_k\| \geq \varepsilon \end{cases} \quad i \neq j \qquad (2)$$

The value of p^o_{ij} cannot be instantly controlled by agent a_i. However, if p^o_{ij} depends on x_i, then a_i can change it at some later time by using a continuous controller.

Each observation p^o_{ij} on the state of some other agent $a_k, k \neq i$ introduces a dependency of a_i on the behavior of a_k. Consider the directed graph defined by the dependencies. Only the subgraph which is reachable from a_i can affect it and needs to be modeled by a_i during verification in § V.

C. Atomic Propositions controlling the Continuous State

The continuous movement of each agent is controlled by a decentralized Navigation Function, defined in § II-B. The destination used in the NF can change in time, according to the specification ϕ_i. Let x_{dij} be the destinations used by agent a_i, where $j \in I^c_i \triangleq \mathbb{N}_{\leq n^c_i}$ and $n^c_i \in \mathbb{N}$. We will refer to each NF $u_{ij} = -\nabla_{x_i}\varphi_i(x, x_{dij})$ as a different continuous controller, associated with x_{dij}. Define the continuous controller AP p^c_{ij} to be *True* when u_{ij} is active, *False* otherwise. Each NF is asymptotically stable from almost all initial conditions (apart from a subset of measure zero). So it reaches $B_\varepsilon(x_{dij})$ in finite time, which motivated the definition of p^o_{ij}. The value of p^c_{ij} is controllable only by agent a_i and no other. At most one NF is active at any time, expressed as $\Box\neg\left(p^c_{ij} \wedge p^c_{ik}\right)$ for all $j \neq k$. The discrete control layer selects which p^c_{ij} to activate, as described in § IV-A.

Note that by defining $x_{dij} = x_k + c_{dij}$ for some relative position $c_{dij} \in \mathbb{R}^2$ with respect to some other agent a_j, formation control can also be achieved. Let $P^c_i \triangleq \left\{p^c_{ij}\right\}_{j \in I^c_i}$.

D. Atomic Propositions observing Path-Connectedness

If a_i cannot communicate with a_k, then the value of each p_{ij}^o which depends on x_k is unobservable. In this case, the discrete control layer designed later cannot decide what action to take. Define the AP p_{ik}^w to be *True* when a_i and a_k are path-connected, *False* otherwise.

If $p_{ik}^w = \text{false}$ implies that the value of ϕ_i is independent of all p_{ij}^o which depend on x_k, then limited communication cannot result in undecidable situations for the discrete controller. So tasks which require communication can "wait".

Let ϕ_i^t be some task which requires communication of a_i with a_k. Then, if ϕ_i requires that eventually ϕ_i^t be satisfied, it assumes a priori that a_i will communicate with a_k in the future. So ϕ_i should either include the assume-guarantee subformula $\Diamond p_{ij}^w \to \phi_i^t$, or the assumption about eventual path-connectedness be modeled during verification. Let $P_i^w \triangleq \{p_{ij}^w\}_{j \in I_w}$ and $I_w \subseteq I_a \setminus \{i\}$.

E. Atomic Propositions maintaining Connectivity

As already discussed, certain subformulas of ϕ_i may require information about other agents. So a_i needs a means to remain connected with selected other agents as shown in Fig. 3. This is achieved by using the connectivity maintenance algorithm from [21]. We define the APs p_{ik}^m which control the connectivity. Agent a_i can request from the network to maintain its connection with a_k by setting p_{ik}^m to *True*, or disconnect by setting it to *False*. Note that disconnection requires that both agents a_i and a_k broadcast a disconnection request. Connections are requested only when path-connected, specified as $\Box(\neg p_{ik}^m \vee p_{ik}^w)$, and connectivity maintenance is modeled by $\Box((p_{ik}^w \wedge p_{ik}^m) \to \bigcirc p_{ik}^w)$. Let $P_i^m \triangleq \{p_{ij}^m\}_{j \in I_w}$.

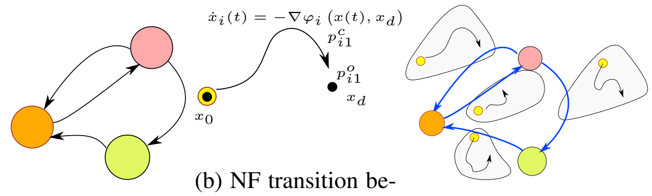
F. Problem Statement

Let $P_i \triangleq P_i^c \cup P_i^m \cup P_i^o \cup P_i^w$ be all the APs of leader l_i. Our aim is to make the agent visit the points of its APs and connect with other agents as specified by φ_i.

Each leader receives an LTL specification $\phi_i \in \Phi_{P_i}$. We are interested in each agent independently synthesizing its own hybrid controller, such that it will satisfy safety properties specified by ϕ_i and check liveness properties when exchanging information with other agents.

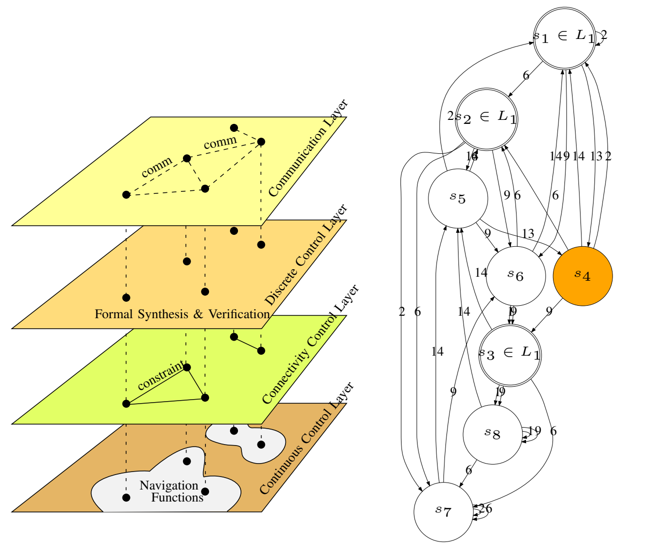
IV. LAYERED ARCHITECTURE

Each agent communicates with others, maintains connectivity with selected neighbors as needed, takes decisions on a discrete level and moves in the continuous state space. So its controller is hybrid, Fig. 1c, comprising of a discrete and a continuous control layer, Figs. 1a and 1b. These are augmented by a layer controlling connectivity and the communication layer, Fig. 2a. We describe this layered architecture briefly here, while more details can be found in [22].



(a) Automaton. (b) NF transition between points. (c) Hybrid controller.

Fig. 1: Each agent is controlled by the composition of a finite state machine with continuous controllers.



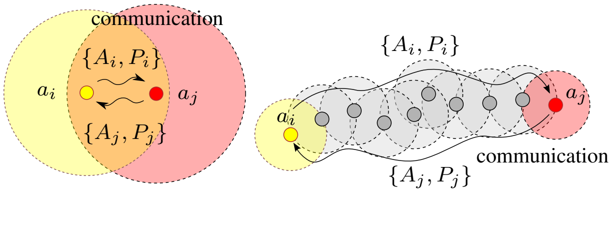
(a) Layered architecture. (b) Controller for ϕ_1 in § VI.

Fig. 2: The multi-agent system composition. Orange denotes S_0. Decimals encode guards as binary valuations of APs, e.g., 2 stands for $\neg p_{11} \wedge p_{12} \wedge \neg p_{13} \wedge \neg p_{14}$.

A. Discrete Control Layer

The specification ϕ_i is in LTL and can always be converted to a non-deterministic Büchi Automaton (NBA) [23], [16]. However, a real-world controller cannot be non-deterministic, because it would require an oracle. So the NBA is converted to a Deterministic Rabin Automaton (DRA) by Safra's construction [24] using off-the-shelf software [25]. The computational complexity of this conversion is exponential. Nonetheless, the proposed modeling using points of interest for each agent (instead of a grid) improves the complexity by reducing the number of states.

The DRA is non-deterministic due to the possible alternative selections of control actions among p_{ij}^c and p_{ij}^m. The non-determinism is resolved by selecting a fixed control action for each observation, at each state. The heuristic criterion is a cost which reflects the ordering of the progress, accept and neutral states of the DRA, in the active pair of progress-acceptance conditions. In other words, the controller tries to move towards the progress state and avoid the accept state of the current pair. Moreover, deadlock states in the DRA representing safety violations are removed, Fig. 2b. This results in a finite state machine which selects control actions p_{ij}^c and p_{ij}^m, given observations p_{ij}^o and p_{ij}^w, and can



(a) Direct connectedness. (b) Indirect, path connectedness.

Fig. 3: When two leaders a_i, a_j meet, they exchange their controllers A_i, A_j and alphabets P_i, P_j. Disks depict communication radii R_s and gray agents are followers.

be used as the discrete control layer. The produced strategy is guaranteed to be safe, but liveness is not guaranteed. This approximates the partial solution of an adversarial game.

B. Continuous Control Layer

The continuous control layer is responsible for implementing the NF $\phi_i(x, x_{dij})$ which is associated with the currently active p^c_{ij} selected by the discrete control layer. So each NF executes a transition between points, when required by ϕ_i, as shown in Fig. 1b. Each transition requires some execution time, during which the automaton executes and evaluates observation propositions.

C. Communication Layer

When agents a_i and a_j become path-connected, they can communicate. For the purpose of deciding if there is no logical conflict between their specifications, the two agents should exchange information about their behavior. The behavior of each agent comprises of its discrete control layer. This is defined by the set P_i of AP and the controller automaton A_i. Note that the AP in P_i define the continuous and connectivity control layers. So, each agent transmits $\{P_i, A_i\}$ when meeting another agent.

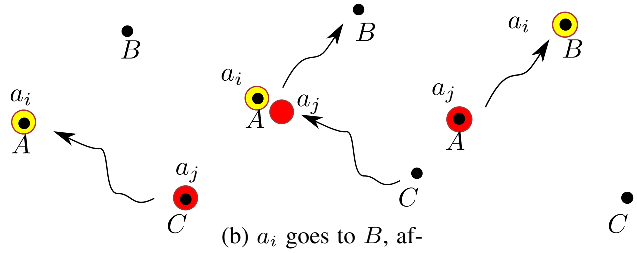
Some observers p^o_{ij} of a_i cannot be "eventually" controlled by its NFs, because their value depends on the state x_k of *another* agent. So a_i does not control x_k, but is affected by it. For this reason it needs information about their behavior. Furthermore, connectivity maintenance requests are broadcast and received through the communication layer.

D. Connectivity Maintenance Layer

If agents a_i and a_j are not path-connected, then a_i cannot observe state x_j. Observables of a_i depending on x_j cannot be evaluated in this case. So the specification of a_i should be independent of x_j, when x_j is unobservable.

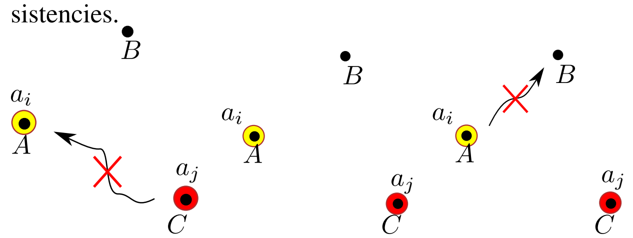
However, certain tasks may require maintenance of connectivity. The role of followers is to function as communication relays that re-transmit information between leaders which have requested to remain connected. We use the distributed connectivity maintenance algorithm from [21]. The connectedness criterion has been adapted here to maintain the connectedness between only those leaders requesting it.

A sufficient number of followers is needed to allow the leaders to stretch as far away from each other as their specifications require. A conservative estimate of the required number of followers per pair of leaders can be obtained by finding the largest distance between points over which propositions in P_i and P_j are defined and divide it by R_s.



(a) a_j goes to A. (b) a_i goes to B, after a_j reaches A. (c) a_i reaches B.

Fig. 4: Simple example with agents to study logical inconsistencies.



(a) a_j will not go to A. (b) a_j remains indefinitely long at C. (c) a_i will fail to go to B.

Fig. 5: The simple example of Fig. 4. Here the agent a_j received a specification ϕ_j telling it to remain indefinitely long at point C, so agent a_i will never start going to point B and fail to fulfill ϕ_i.

V. DECENTRALIZED VERIFICATION

A. Logical conflicts between specifications

As detailed in § III, the agents receive independent specifications in a decentralized manner. Therefore, it is not guaranteed that ϕ_i and ϕ_j will agree, they may be logically inconsistent with each other.

For example ϕ_i may tell a_i to wait at point A until a_j appears, then go to point B, as shown in Fig. 4. Assume that a_j starts at point C. If ϕ_j tells a_j to remain at C forever, then it will never appear at A. So a_i will never start going to B and ϕ_i cannot be satisfied, Fig. 5.

This result is due to the logical conflict between the specifications of the two agents. It becomes apparent that in order for all agents to be able to satisfy their specifications, their specifications should in some way agree with each other. Rich specifications are still possible (e.g. a_j visiting another 10 points before A). However, logical compatibility of ϕ_i and ϕ_j is not ensured a priori. For this reason, it is necessary for the agents to decide whether there is a logical conflict between their specifications while executing them.

This elicits the question of what happens in case the specifications are not mutually satisfiable, due to logical conflicts. In that case it is necessary that some of the specifications be changed, e.g. relaxed in some way. This procedure will be a refinement of one, or both, of the specifications, so that they become logically consistent. This reconfigurability of the system constitutes an interesting direction of future research, related to [26].

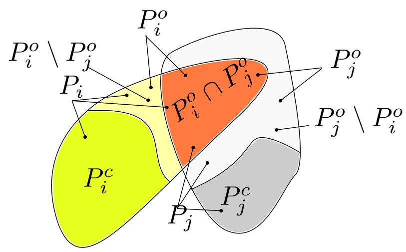


Fig. 6: In order to identify common atomic propositions, communicating agents should interpret them similarly. For simplicity P_i^w, P_j^w are not shown.

B. Verification Procedure

After appropriately modeling agent behaviors, all possible executions of the system can be examined using model checking. Here we describe how the system is modeled in the Promela programming language, in order to be checked with the SPIN model checking software [27]. More details about the modeling and the custom interface between MATLAB and SPIN can be found in [22].

Only those agents on which a_i depends through its observables need to be modeled by a_i. As a_i meets more agents, it can model its environment better.

The issue of state explosion imposes a limitation to the number of agents modeled. This is partially mitigated by the relatively small number of points interesting each agent (as opposed to grids) and the dependencies between agents, which are assumed to be sparse, as will usually be the case in a scenario involving distributed robotic teams.

1) Alphabet Comparison: If an agent a_j has an NF controller p_{j1}^c, then it also has an associated observer p_{j1}^o. If agent a_i observes with p_{i1}^o the same point as p_{j1}^o, then these two observers are identical. In order to find which observer APs are common, Fig. 6, the alphabets P_i^o and P_j^o are compared using their definitions. In more detail, the definitions of p_{ik}^o and p_{jr}^o comprise of the indices of the agents on whose states each AP depends (assuming both agents use the same index set I_a), together with the points of interest. By comparing the indices and points, observers of different agents can be compared. Note that the controllers p_{ij}^c of different agents cannot be the same. The APs in the collected alphabets are initialized to $\mathtt{false}$ for the verification.

2) Modeling agent discrete controllers: The controller automaton constructed in § IV-A is modeled as an independently executing deterministic process, which selects the values of controllable p_{ij}^c and p_{ij}^m based on the current values of p_{ij}^o and p_{ij}^w.

3) Modeling Navigation Function Controllers: NFs guarantee transitions between points in finite time (a priori unknown), abstracting them to the discrete level. This is modeled by introducing each NF controller as a separate process, which monitors when p_{ij}^c becomes $\mathtt{true}$, so then it sets $p_{ik}^o = \mathtt{true}$. Fairness is enforced during model checking, to ensure that the NF controller will eventually execute after a non-deterministic number of iterations, allowing any possible interleaving to be checked.

4) Modeling agents unknown yet: As already mentioned, the agent has initially limited knowledge about its environment. In other words, it does not know what the other agents will do. Although, the verification process can be iteratively executed, we need some way of modeling the state of incomplete information about other agents, on which the one performing the verification, let that be a_i, depends.

This can be achieved by modeling all possible executions (since we do not know what is the particular behavior of the other agent). An independently executing process which changes the associated observable values between $\mathtt{true}$ and $\mathtt{false}$ in a non-deterministic way models this situation.

VI. SIMULATION RESULTS

A case study using the proposed algorithm involving $n_l = 6$ leader agents and $n_f = 3$ followers, illustrated in Fig. 7, in which agents a_1, a_2 (blue,green) should eventually patrol the lower-right area, visiting infinitely often two points one after the other. Agents a_5, a_6 (magenta,yellow) wait for a_4 (cyan) before requesting connectivity and moving to x_{d51}, x_{d61}. Agent a_4 goes first to x_{d41}, then to x_{d42}. Finally, a_3 (red) goes to x_{d31}, waits to see a_2 and then moves to x_{d32}. Followers f_7, f_8, f_9 are available to provide path connectivity where needed.

The specifications are defined as

$$\phi_1 = \Box\big(\neg\,(p_{11}^c \wedge p_{12}^c) \wedge (p_{11}^o \rightarrow (p_{12}^c \,\mathcal{U}\, p_{12}^o)) \wedge (p_{12}^o \rightarrow (p_{11}^c \,\mathcal{U}\, p_{11}^o))\big)$$

$$\phi_2 = \Box\big(\neg\,(p_{21}^c \wedge p_{22}^c) \wedge (p_{21}^o \rightarrow (p_{22}^c \,\mathcal{U}\, p_{22}^o)) \wedge (p_{22}^o \rightarrow (p_{21}^c \,\mathcal{U}\, p_{21}^o))\big)$$

$$\phi_3 = \Box\neg\,(p_{31}^c \wedge p_{32}^c) \wedge (p_{31}^c \,\mathcal{U}\, (p_{31}^o \wedge p_{32}^c))$$

$$\phi_4 = \Box\neg\,(p_{41}^c \wedge p_{42}^c) \wedge p_{41}^c \,\mathcal{U}\, p_{41}^o \wedge \Diamond\,(p_{42}^c \,\mathcal{U}\, p_{42}^o)$$

$$\phi_5 = \Box\,(\neg p_{52}^m \vee p_{53}^w) \wedge \Box\,((p_{52}^m \wedge p_{53}^w) \rightarrow \bigcirc p_{53}^w) \wedge ((\neg p_{51}^c) \,\mathcal{U}\, p_{51}^o) \wedge \Diamond\,(p_{51}^c \,\mathcal{U}\, p_{52}^o) \wedge ((\neg p_{52}^m) \,\mathcal{U}\, p_{51}^o) \wedge (\Diamond p_{51}^o \rightarrow \Diamond p_{52}^m)$$

$$\phi_6 = \Box\,(\neg p_{62}^m \vee p_{63}^w) \wedge \Box\,((p_{62}^m \wedge p_{63}^w) \rightarrow \bigcirc p_{63}^w) \wedge ((\neg p_{61}^c) \,\mathcal{U}\, p_{61}^o) \wedge \Diamond\,(p_{61}^c \,\mathcal{U}\, p_{62}^o) \wedge ((\neg p_{62}^m) \,\mathcal{U}\, p_{61}^o) \wedge (\Diamond p_{61}^o \rightarrow \Diamond p_{62}^m)$$

and the followers constantly execute a NF with neighbor list. The NF controller destinations are $x_{d11} = [0,0]^{\mathrm{T}}, x_{d12} = [2,-2]^{\mathrm{T}}, x_{d21} = [0,-1]^{\mathrm{T}}, x_{d22} = [1,1]^{\mathrm{T}}, x_{d31} = [-1,+1]^{\mathrm{T}}, x_{d32} = [3,-1]^{\mathrm{T}}, x_{d41} = [0,3]^{\mathrm{T}}, x_{d42} = [-2,1]^{\mathrm{T}}, x_{d51} = [-3,5]^{\mathrm{T}}, x_{d61} = [2,4]^{\mathrm{T}}$ and when p_{52}^m, p_{62}^m are active, they issue connectivity requests to link a_5, a_6.

Let the observable APs be defined as follows $p_{11}^o \triangleq (\|x_1 - x_{d11}\| < 0.1)\,, p_{12}^o \triangleq (\|x_1 - x_{d12}\| < 0.1)\,,$ $p_{21}^o \triangleq (\|x_2 - x_{d21}\| < 0.1)\,, p_{22}^o \triangleq (\|x_2 - x_{d22}\| < 0.1)\,,$ $p_{31}^o \triangleq (\|x_2 - x_3\| < 1)\,, p_{41}^o \triangleq (\|x_4 - x_{d41}\| < 0.1)\,,$ $p_{42}^o \triangleq (\|x_4 - x_{d42}\| < 0.1)\,, p_{51}^o \triangleq (\|x_4 - x_{d41}\| < 0.4)\,,$ $p_{52}^o \triangleq (\|x_5 - x_{d51}\| < 0.1)\,, p_{61}^o \triangleq (\|x_4 - x_{d41}\| < 0.4)\,,$ $p_{62}^o \triangleq (\|x_6 - x_{d61}\| < 0.1)$ and p_{53}^w, p_{63}^w detect path-connectivity between a_5, a_6 through followers. APs p_{54}^w, p_{64}^w detect path-connectivity between a_4 and a_5, a_6, respectively, through any agent and function as information availability switches. Note that p_{31}^o requires information about x_2, but it also functions as an information availability switch, because $\|x_2 - x_3\| < 1 < R_s$, so no additional observable is needed.

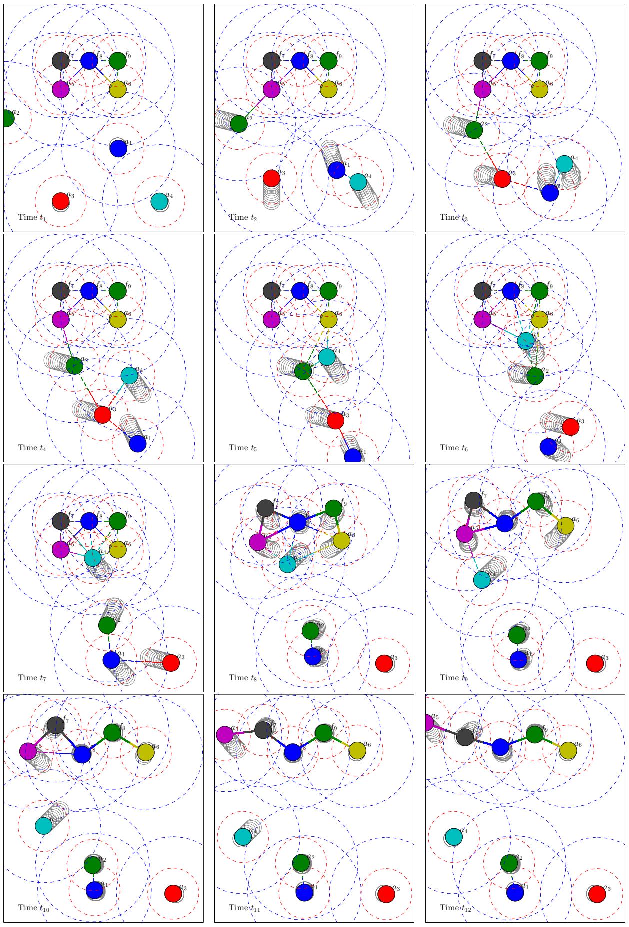


Fig. 7: Decentralized multi-agent scenario with independent LTL specifications and decentralized Navigation Functions with limited sensing R_s (blue dashed), collision avoidance distances d_c (red dashed), thin dashed lines indicate sensing, thick continuous lines denote connectivity links.

In the simulation $a_1, \cdots, a_4$ proceed to their objectives avoiding collisions, so that at t_7 agents a_1, a_2 have started patrolling the lower left area and a_3 is heading towards x_{d32}. At t_8 agent a_4 comes within distance 0.4 of x_{d41}, so that connectivity is triggered (thick continuous lines) and a_5, a_6 begin moving to x_{d51}, x_{d61} respectively, while f_7, f_8, f_9 maintain the path-connectivity between a_5, a_6, as requested by p_{52}^m, p_{62}^m. Note that a_5 will conclude that ϕ_5 is not realizable, until it learns the behavior of a_4 and models the assumption of eventual path-connectivity with a_6.

VII. Conclusions and Future Work

An algorithm for converting independent LTL specifications to agent discrete controller automatons manipulating continuous Navigation Function controllers has been introduced. Execution is decentralized and under limited communication. Mutual satisfiability is verified when agents meet and can exchange their automata. Further directions of research concern the introduction of reconfigurability capabilities, when agent specifications are conflicting.

References

[1] S. G. Loizou and K. J. Kyriakopoulos, "Multirobot navigation functions i," in *Stochastic Hybrid Systems*, ser. LNCIS. Springer, 2006, vol. 337, pp. 171–207.

[2] D. V. Dimarogonas, S. G. Loizou, and K. J. Kyriakopoulos, "Multi-robot navigation functions ii: Towards decentralization," in *Stochastic Hybrid Systems*, ser. LNCIS. Springer, 2006, vol. 337, pp. 209–253.

[3] S. LaValle, *Planning Algorithms*. Cambridge Univ. Press, 2006.

[4] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Translating structured english to robot controllers," *Advanced Robotics Special Issue on Selected Papers from IROS 2007*, vol. 22, no. 12, p. 13431359, 2008.

[5] G. Fainekos, A. Girard, H. Kress-Gazit, and G. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[6] T. Wongpiromsarn, U. Topcu, and R. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proc. 48th IEEE Conf. on Decis. and Cont.*, 2009, pp. 5997–6004.

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

[8] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-uav mission planning," *Int. J. of Robust and Nonlinear Cont.*, vol. 21, no. 12, pp. 1372–1395, 2011.

[9] ——, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. 48th IEEE Conf. on Decis. and Cont.*, 2009, pp. 2222–2229.

[10] S. Karaman, S. Rasmussen, D. Kingston, and E. Frazzoli, "Specification and planning of uav missions: a process algebra approach," in *Proc. Amer. Cont. Conf.*, 2009, pp. 1442–1447.

[11] M. Karimadini and H. Lin, "Guaranteed global performance through local coordinations," *Automatica*, vol. 47, no. 5, pp. 890–898, 2011.

[12] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. on Rob.*, vol. 25, no. 6, pp. 1370 –1381, 2009.

[13] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on ltl specifications," in *Proc. 43rd IEEE Conf. on Dec. and Cont.*, vol. 1, 2004, pp. 153–158.

[14] M. Kloetzer, S. Itani, S. Birch, and C. Belta, "On the need for communication in distributed implementations of ltl motion specifications," in *Proc. IEEE Conf. on Rob. and Aut.*, 2010, pp. 4451–4456.

[15] M. Kloetzer and C. Belta, "Distributed implementations of global temporal logic motion specifications," in *Proc. IEEE Conf. on Rob. and Aut.*, 2008, pp. 393–398.

[16] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT, 2008.

[17] D. E. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary," *Adv. in Applied Math.*, vol. 11, pp. 412–442, 1990.

[18] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*, 1977, pp. 46–57.

[19] P. Wolper, *Constructing automata from temporal logic formulas: a tutorial*. Springer, 2002.

[20] D. Dimarogonas and K. Johansson, "Decentralized connectivity maintenance in mobile networks with bounded inputs," in *Proc. IEEE Conf. on Rob. and Aut.*, 2008, pp. 1507–1512.

[21] M. Zavlanos and G. Pappas, "Distributed connectivity control of mobile networks," *IEEE Trans. on Rob.*, vol. 24, no. 6, pp. 1416 –1428, 2008.

[22] I. Filippidis, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Decentralized multi-agent control from local LTL specifications," NTUA, Tech. Rep., 2012.

[23] R. J. Büchi, "Symp. on decision problems: On a decision method in restricted second order arithmetic," in *Proc. of the 1960 Int. Congr. on Logic, Meth. and Phil. of Sc.*, ser. Studies in Logic and the Foundations of Math. Elsevier, 1966, vol. 44, pp. 1–11.

[24] S. Safra, "On the complexity of ω automata," in *29th Annual Symp. on Foundations of Comp. Sc.*, 1988, pp. 319–327.

[25] J. Klein and C. Baier, "On-the-fly stuttering in the construction of deterministic ω-automata," in *Implementation and Application of Automata*, ser. LNCS. Springer, 2007, vol. 4783, pp. 51–61.

[26] G. E. Fainekos, "Revising temporal logic specifications for motion planning," in *Proc. IEEE Conf. on Rob. and Aut.*, 2011.

[27] G. Holzmann, *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.