Hybrid Control of Multi-robot Systems Using Embedded Graph Grammars

Meng Guo, Magnus Egerstedt and Dimos V. Dimarogonas

***Abstract*— We propose a distributed and cooperative motion and task control scheme for a team of mobile robots that are subject to dynamic constraints including inter-robot collision avoidance and connectivity maintenance of the communication network. Moreover, each agent has a local high-level task given as a Linear Temporal Logic (LTL) formula of desired motion and actions. Embedded graph grammars (EGGs) are used as the main tool to specify local interaction rules and switching control modes among the robots, which is then combined with the model-checking-based task planning module. It is ensured that all local tasks are satisfied while the dynamic constraints are obeyed at all time. The overall approach is demonstrated by simulation and experimental results.**

I. Introduction

The control of multi-robot systems could normally consist of two goals: the first is to accomplish high-level system-wise tasks, e.g., formation and flocking [21], task assignment [20] and collaboration [19]; the second is to cope with constraints that arise from the inter-robot interactions, e.g., collision avoidance [5] and communication maintenance [21]. These two goals are often dependent and heavily coupled since it is essential to consider one when trying to fulfill another. For instance, it is unlikely that a multi-robot formation method would work if the inter-robot collision is not addressed, nor a collaborative task assignment scheme would work if the communication network among the robots is not ensured to be connected. Thus in this work, we tackle some aspects of both goals at the same time.

Regarding the high-level task, we rely on Linear Temporal Logic (LTL) as the formal language that can describe planning objectives more complex than the well-studied point-to-point navigation problem. The task is specified as a LTL formula with respect to an abstraction of the robot motion [1], [3]. Then a high-level discrete plan is found by off-the-shelf model-checking algorithms [2], which is then implemented through the low-level continuous controller [6], [7]. [10] extends this framework by allowing both robot motion and actions in the task specification. Similar methodology has also been applied to multi-robot systems [4], [12], [20]. Two different formalisms have appeared that one focuses on decomposing a global temporal task into bisimilar local ones in a *top-down* approach, which can be then assigned and implemented by individual robots in a synchronized [4] or partially-synchronized [11] manner; another is to assume that there is no pre-specified global task and individual temporal tasks are assigned locally to each robot [8], [9], [19], which favors a *bottom-up* formulation. These local tasks can be independent [9] or dependent [8] due to collaborative tasks. We favor the second formulation as it is useful for multi-robot systems where the number of robots is large, the robots are heterogeneous and each robot has a specific task assignment.

However, most of the aforementioned work neglects the second goal to cope with inter-robot dynamic constraints, e.g, inter-robot collision is not handled formally in [9], [19] and connectivity of the communication network is taken for granted in [8], [9], [20]. Here we take advantage of Embedded Graph Grammars (EGGs) to tackle these constraints, as initially introduced in [14], [15]. It allows us to encode the robot dynamics, local information exchange and switching control modes in a unified hybrid scheme. Successful applications to multi-robot systems can be found in, e.g., coverage control [15], self-reconfiguration of modular robots [17], and autonomous deployment [18]. Only local interactions or communication are needed for the execution of EGGs, making it suitable for large-scale multi-robot systems.

The proposed solution combines the temporal-logic-based task planning with the EGGs-based hybrid control, which overall serves as a distributed and cooperative control scheme for multi-robot systems under local temporal tasks and motion constraints. The main contribution lies in the proposed EGGs that ensure the fulfillment of all local tasks, while guaranteeing no inter-robot collision and the communication network being connected at all time, given the robots' limited capabilities of communication and actuation.

The rest of the paper is organized as follows: Section II briefly introduces essential preliminaries. In Section III, we formally state the problem. Section IV presents the proposed solution. Numerical and experimental examples are shown in Section V. We conclude in Section VI.

II. Preliminaries

A. Embedded Graph Grammar

Here we review some basics of Embedded Graph Grammars (EGGs). For a detailed description, see [14], [15]. Let Σ be a set of pre-defined labels. A labeled graph is defined as the quadruple $G = (V, E, l, e)$, where V is a set of vertices, $E \subset V \times V$ is a set of edges, $l : V \to \Sigma$ is a vertex labeling function, and $e : E \to \Sigma$ is an edge labeling function. Given a continuous state space X for the vertices,

The first and third authors are with the KTH Centre for Autonomous Systems and ACCESS Linnaeus Center, EES, KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden. mengg, dimos@kth.se. The second author is with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA. magnus@gatech.edu. This work was supported by the Swedish Research Council (VR). The work by the second author was supported by Grant N0014-15-1-2115 from the U.S. Office of Naval Research.

an embedded graph is given by $\gamma = (G, x)$, where G is a labeled graph and $x : V \to X$ is a realization function. We use G_γ, x_γ to denote the labeled graph and continuous states associated with γ. The set of allowed embedded graphs being considered is denoted by Γ. Furthermore, an embedded graph transition is a relation $\mathcal{A} \subset \Gamma \times \Gamma$ such that $(\gamma_1, \gamma_2) \in \mathcal{A}$ implies $x_{\gamma_1} = x_{\gamma_2}$ and $G_{\gamma_1} \neq G_{\gamma_2}$. The rules and conditions associated with the transitions are called graph grammars.

B. Linear Temporal Logic

The basic ingredients of a Linear Temporal Logic (LTL) formula are a set of atomic propositions AP and several boolean or temporal operators, formed by the syntax [2]: $\varphi ::= \top \,|\, a \,|\, \varphi_1 \wedge \varphi_2 \,|\, \neg\varphi \,|\, \bigcirc \varphi \,|\, \varphi_1 \mathsf{U} \varphi_2$, where $a \in AP$ and $\top$ (*True*), $\bigcirc$ (*next*), U (*until*). Other operators like $\Box$ (*always*), $\Diamond$ (*eventually*), $\Rightarrow$ (*implication*) and the semantics of LTL formulas can be found in Chapter 5 of [2]. There is a union of infinite words that satisfy φ: $\texttt{Words}(\varphi) = \{\sigma \in (2^{AP})^\omega \,|\, \sigma \models \varphi\}$, where $\models \subseteq (2^{AP})^\omega \times \varphi$ is the satisfaction relation. LTL formulas can be used to specify various control tasks, such as safety ($\Box\neg\varphi_1$, globally avoiding φ_1), ordering ($\Diamond(\varphi_1 \wedge \Diamond(\varphi_2 \wedge \Diamond\varphi_3))$, φ_1, φ_2, φ_3 hold in sequence), response ($\varphi_1 \Rightarrow \varphi_2$, if φ_1 holds, φ_2 will hold in future), repetitive surveillance ($\Box\Diamond\varphi$, φ holds infinitely often).

III. Problem Formulation

A. Robot Dynamics

Consider a team of N mobile robots (agents) in an obstacle-free 2D workspace, indexed by $\mathcal{N} = \{1, 2, \cdots, N\}$. Each agent $i \in \mathcal{N}$ satisfies the unicycle dynamics:

$$\dot{x}_i = v_i \cos(\theta_i), \quad \dot{y}_i = v_i \sin(\theta_i), \quad \dot{\theta}_i = w_i, \qquad (1)$$

where $s_i = (x_i, y_i, \theta_i) \in \mathbb{R}^3$ is the state with position $p_i = (x_i, y_i)$ and orientation θ_i; and $u_i = (v_i, w_i) \in \mathbb{R}^2$ is the control input as linear and angular velocities, bounded by $v_{\max}$ and $w_{\max}$. Agent i has reference linear and angular velocities $V_i < v_{\max}$ and $W_i < w_{\max}$, respectively. Each agent occupies a disk area of $\{p \in \mathbb{R}^2 \,|\, \|p - p_i\| \leq r\}$, where $r > 0$ is the radius of its physical volume. A *safety distance* $\underline{d} > 2r$ is the minimal inter-agent distance to avoid collisions. Moreover, agents $i, j \in \mathcal{N}$ can only communicate if $\|p_i - p_j\| \leq \overline{d}$, where $\overline{d} > \underline{d}$ is the *communication radius*.

Definition 1: Agents $i, j \in \mathcal{N}$ are: in collision if $\|p_i(t) - p_j(t)\| \leq \underline{d}$; neighbors if $\|p_i(t) - p_j(t)\| \leq \overline{d}$. ■

Given the agent states, an embedded graph $\gamma(t)$ is defined as $\gamma(t) = (G(t), p(t))$, where $G(t) = (\mathcal{N}, E(t))$ with $(i, j) \in E(t)$ if $\|p_i(t) - p_j(t)\| < \overline{d}$, $\forall i, j \in \mathcal{N}$, $i \neq j$; $p(t)$ is the stack vector of all $p_i(t)$. Then we define the set of allowed embedded graphs Γ_d as follows:

Definition 2: An embedded graph $\gamma(t) = (G(t), p(t))$ is *allowed* that $\gamma(t) \in \Gamma_d$ if (i) $\|p_i(t) - p_j(t)\| > \underline{d}$, $\forall i, j \in \mathcal{N}$, $i \neq j$; and (ii) the graph $G(t)$ is connected. ■

B. Local Task Specification over Motion and Actions

For each agent $i \in \mathcal{N}$, there is a set of points of interest in the workspace, denoted by $Z_i = \{z_{i1}, z_{i2}, \cdots, z_{iM_i}\}$, where $z_{i\ell} \in \mathbb{R}^2$, $\forall \ell = 1, 2, \cdots, M_i$, where $M_i > 0$. Each point satisfies different properties. Furthermore, it is capable of performing a set of actions, described by the action primitives $\Sigma_i = \{a_1, a_2, \cdots, a_{K_i}\}$. Each action has conditions on the workspace property that should be satisfied to perform it and also an effect on the workspace after performing it. Combining these two aspects, we can derive a complete motion and action model for agent i as a finite transition system (FTS) $\mathcal{M}_i = (\Pi_i, \to_i, \Pi_{i,0}, AP_i, L_i)$, where $\Pi_i = Z_i \times \Sigma_i$ is the set of states; $\to_i$: $\Pi_i \to 2^{\Pi_i}$ is the transition relation; $\Pi_{i,0} \subset \Pi_i$ is the set of initial states; AP_i is the set of atomic propositions over workspace property and action primitives; $L_i : \Pi_i \to 2^{AP_i}$ is the labeling function that returns the set of propositions satisfied at each state. We omit the details about how to construct $\mathcal{M}_i$ here due to limited space and refer the readers to [8] and [10]. Then the local task for each agent $i \in \mathcal{N}$ is specified as an LTL formula φ_i over AP_i as described in Section II-B.

Definition 3: The task φ_i is satisfied if there *exists* a sequence of time instants $t_{i0}\, t_{i1}\, t_{i2} \cdots$ and a sequence of states $\pi_{i\ell_0}\pi_{i\ell_1}\pi_{i\ell_2} \cdots$ such that: $\pi_{i\ell_k} = (z_{i\ell_k}, a_{i\ell_k})$ where $z_{i\ell_k} \in Z_i$ and $a_{i\ell_k} \in \Sigma_i$; at time t_{ik}, $\|p_i(t_{ik}) - z_{i\ell_k}\| \leq c_i$ where $c_i > 0$ is a given threshold for reaching a point of interest and the action $a_{i\ell_k}$ is performed at $z_{i\ell_k}$, $\forall k = 0, 1, 2, \cdots$; and $L_i(\pi_{i\ell_0})L_i(\pi_{i\ell_1})L_i(\pi_{i\ell_2}) \cdots \models \varphi_i$. ■

Some examples of considered tasks are: "Infinitely often pick up object `A` in point 1 and then drop it to point 2"; "Surveil points 3 and 4 by taking pictures there"; "Go to point 5 and operate machine `M`, then go to point 6 and charge the battery", which all involve robot motion and actions.

C. Problem Statement

Design a distributed motion control scheme such that φ_i is satisfied, $\forall i \in \mathcal{N}$, while at the same time $\gamma(t) \in \Gamma_d$, $\forall t \geq 0$.

IV. Solution

The proposed solution consists of two major parts: the embedded graph grammars (EGGs) design and the local task coordination, of which the details are given in the sequel. Then we combine them as the complete solution, where we also prove the correctness formally.

A. EGGs Design

The design of EGGs involves three parts: (i) the workspace discretization, (ii) the essential building blocks, and (iii) the graph grammars.

1) Workspace Discretization: The 2-D workspace is discretized into uniform grids by a quantization function, through which we transform the collision avoidance and connectivity constraints into relative-grid positions.

Definition 4: Given a point $(x, y) \in \mathbb{R}^2$, its grid position is given by the function $\textsc{Grid} : \mathbb{R}^2 \to \mathbb{Z}^2$:

$$(g_x, g_y) \triangleq \textsc{Grid}(x, y) \triangleq ([\frac{x}{\underline{d}}], [\frac{y}{\underline{d}}]), \qquad (2)$$

where $[\cdot]$ is the *round* function that returns the closest integer ($[0.5] = 1$) and $\underline{d}$ is the safety distance introduced earlier. ■

Given that $p_i(t) = (x_i(t), y_i(t))$ at time $t > 0$, the grid position of agent i is given by $g_i(t) \triangleq (g_i^x(t), g_i^y(t)) =$

GRID$(x_i(t),\, y_i(t))$. Now consider two agents i and j whose grid positions are given by $g_i(t)$ and $g_j(t)$.

Definition 5: The collision function COLLIDE : $\mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{B}$ satisfies: COLLIDE$(g_i(t),\, g_j(t)) \triangleq \bot$ if $|g_i^x - g_j^x| \geq 2$ or $|g_i^y - g_j^y| \geq 2$; otherwise, COLLIDE$(g_i(t),\, g_j(t)) \triangleq \top$. The neighboring function NEIGHBOR : $\mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{B}$ satisfies: NEIGHBOR$(g_i(t),\, g_j(t)) \triangleq \top$ if it holds that $\|(|g_i^x - g_j^x| + 1,\, |g_i^y - g_j^y| + 1)\| \leq \lambda_d$, where $\lambda_d \triangleq \overline{d}/\underline{d} > 1$; otherwise, NEIGHBOR$(g_i(t),\, g_j(t)) \triangleq \bot$. ■

Lemma 1: By Definition 1 agents i and j are collision-free at time $t > 0$ if COLLIDE$(g_i(t),\, g_j(t)) = \bot$; they are connected if NEIGHBOR$(g_i(t),\, g_j(t)) = \top$.

Proof: For $p_i(t),\, p_j(t) \in \mathbb{R}^2$, by (2) it holds that if $|g_i^x(t) - g_j^x(t)| \geq 2$, then $|x_i - x_j| > \underline{d}$ and $\|p_i(t) - p_j(t)\| \geq |x_i - x_j| + |y_i - y_j| > \underline{d}$, i.e., agents i and j are collision-free by Definition 1. The same holds when $|g_i^y - g_j^y| \geq 2$. For any $p_i(t),\, p_j(t) \in \mathbb{R}^2$, by (2) it holds that $|x_i - x_j| < \underline{d} \cdot (|g_i^x - g_j^x| + 1)$ and $|y_i - y_j| < \underline{d} \cdot (|g_i^y - g_j^y| + 1)$. Then $\|p_i(t) - p_j(t)\| < \underline{d} \cdot \|(|g_i^x - g_j^x| + 1,\, |g_i^y - g_j^y| + 1)\| < \underline{d} \cdot \lambda_d = \overline{d}$, i.e., agents i and j are neighbors. ■

2) Building Blocks: We introduce five building blocks in this part that are essential for the construction of EGGs.

(I) Labels on vertices and edges. The *first* building block is the modified embedded graph $\gamma(t) = (G(t),\, p(t))$ where $G(t) = (\mathcal{N},\, E(t),\, l,\, e)$, where l and e are the vertex and edge labeling functions. Each vertex has a label with three named fields $\{\texttt{id},\, \texttt{mode},\, \texttt{data}\}$, where `id` is the agent ID; `mode` is the agent status, including $\{\texttt{check},\, \texttt{static},\, \texttt{move}\}$; and `data` stores data for the execution, which has three sub-fields $\{\texttt{nb},\, \texttt{pt},\, \texttt{gi}\}$, where `nb` saves the a set of other agents' IDs; `pt` saves a tentative path; and `gi` saves a positive gain parameter. Moreover, the edge between neighbors has the named field `id`, i.e., the edge from agent i to j has the `id` as $(i,\, j)$. For brevity, we omit the definitions of l and e that map $\mathcal{N}$ and $E(t)$ to the set of labels, which is a cartesian product of the named fields above. We use dot notation to indicate the value of label fields. For instance, "$i.\texttt{mode} = \texttt{move}$" means that agent i has the mode being `move`. We call an agent *static* if its mode is `static` and *active* if its mode is `move`.

To start with, we need the notion of a *local sub-graph* for agent $i \in \mathcal{N}$, denoted by $G_i(t) = (V_i(t),\, E_i(t))$, where (i) $V_i(t) = \{i\} \cup \mathcal{N}_i(t)$, where $\mathcal{N}_i(t) = \{j \in \mathcal{N} \,|\, (i,\, j) \in E(t)\}$; (ii) $(j,\, k) \in E_i(t)$ if $(j,\, k) \in E(t)$, $\forall j,\, k \in V_i(t)$. Clearly, $G_i(t)$ is a sub-graph of $G(t)$ and it can be constructed locally by agent i. Clearly if $G(t)$ is connected, then $G_i(t)$ is connected, $\forall i \in \mathcal{N}$.

(II) Neighbor marking scheme. The *second* building block is the mechanism to maintain graph connectivity while the agents are moving. The main idea is to choose locally some agents to be static and the others be active; and more importantly ensure that the active agents remain connected to its static neighbors while moving. The most straightforward way is to allow only one agent move at a time, which is extremely inefficient as a system. Here we propose a local *marking scheme* to choose static and active agents, which allows more agents being active simultaneously.

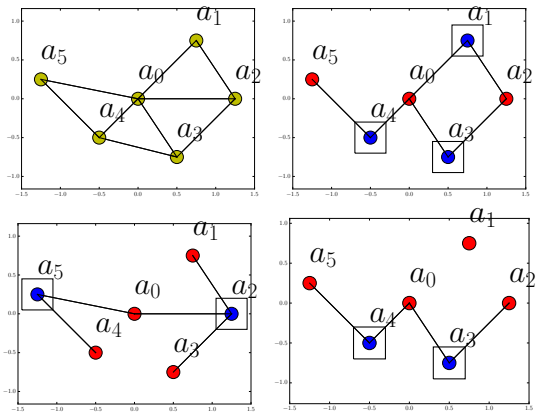

Fig. 1. Examples of marking schemes for agent a_0 locally: (a) Local graph G_0, consisting of neighbors a_1, a_2, a_3, a_4, a_5; (b) one allowed marking scheme where a_1, a_3, a_4 are marked, with the associated marked sub-graph of G_0^m,G_2^m,G_5^m; (c) another allowed marking scheme where a_2, a_5 are marked; (d) an *not*-allowed marking scheme where a_3, a_4 are marked as a_1 is neither marked nor connected to a marked agent.

Assume that agent $i \in \mathcal{N}$ satisfies $i.\texttt{mode} = \texttt{active}$. Given its local graph $G_i(t)$ at time $t > 0$, agent i can communicate with its neighbor $j \in \mathcal{N}_i(t)$ regarding its `mode`. We denote by $\mathcal{N}_i^s(t) = \{j \in \mathcal{N}_i(t) \,|\, j.\texttt{mode} = \texttt{static}\}$ the set of static neighbors; $\mathcal{N}_i^a(t) = \{j \in \mathcal{N}_i(t) \,|\, j.\texttt{mode} = \texttt{move}\}$ the set of active neighbors; and the others are in the `check` mode. A marking scheme of agent i at time $t > 0$ marks a subset of its neighbors, denoted by $\mathcal{N}_i^m(t) \subseteq \mathcal{N}_i(t)$, as the potential agents to become static. Given the above three categories, a marking scheme should satisfy the following:

Definition 6: The marked set of neighbors $\mathcal{N}_i^m(t)$ is *allowed* if: (i) for any neighbor $j \in \mathcal{N}_i(t)$, it holds that either $j \in \mathcal{N}_i^m(t)$ or there exists $g \in \mathcal{N}_i^m(t)$ that $(j,\, g) \in E_i(t)$; (ii) $\mathcal{N}_i^s(t) \subseteq \mathcal{N}_i^m(t)$ and $\mathcal{N}_i^m(t) \cap \mathcal{N}_i^a(t) = \emptyset$. ■

The first condition requires that any neighbor is either marked or *directly* connected to a marked agent, while the second condition says that all static and no active neighbors should be marked. Examples of different marked schemes are shown in Figure 1. Given the set of marked agents $\mathcal{N}_i^m(t) \subseteq \mathcal{N}_i(t)$, the *marked* sub-graph of $G_i(t)$ is defined as:

Definition 7: The marked sub-graph $G_i^m(t) \triangleq (V_i^m(t),\, E_i^m(t))$ has $V_i^m(t) = \{i\} \cup \mathcal{N}_i^m(t)$ and $(j,\, k) \in E_i^m(t)$ if $(j,\, k) \in E_i(t)$, $\forall j,\, k \in V_i^m$. ■

(III) Potential path synthesis. The *third* building block is the synthesis algorithm to derive a local path for an active agent $i \in \mathcal{N}$ to move towards its point of interest $z_{i\ell} = (z_{i\ell}^x,\, z_{i\ell}^y) \in Z_i$ while keeping connected and collision-free to all its marked neighbors in $\mathcal{N}_i^m$.

Denote by $\mathbf{p}_i$ the *tentative* discrete path of agent i with length $L_i \geq 1$ that obeys the following structure:

$$\mathbf{p}_i = q_i^0\, q_i^1 \cdots q_i^l \cdots q_i^{L_i} \tag{3}$$

where $q_i^l = (s_i^l,\, t_i^l,\, v_i^l)$ is a 3-tuple with the desired state $s_i^l = (x_i^l,\, y_i^l,\, \theta_i^l) \in \mathbb{R}^3$, the approximated time t_i^l when s_i^l will be reached, and the linear velocity v_i^l at q_i^l when heading towards q_i^{l+1}, $\forall l = 0, 1, \cdots, L_i$. Notice that

$q_i^0 \triangleq (s_i(t),\, 0,\, V_i)$ initially, where V_i is the reference linear velocity. Moreover, the position $p_i^l = (x_i^l,\, y_i^l)$ of s_i^l should correspond to the center of a grid $g_i^l = \textsc{Grid}(p_i^l)$ and two consecutive positions $p_i^l,\, p_i^{l+1}$ correspond to two adjacent grids, $\forall l = 0, 1, \cdots, L_i - 1$. Given the current state $s_i(t)$ of agent i, the potential *cost* of $\mathbf{p}_i$ is defined as:

$$\textsc{Cost}(\mathbf{p}_i) \triangleq \sum_{l=0}^{L_i-1} \big(\|p_i^l - p_i^{l+1}\| + \alpha \cdot |\theta_i^l - \theta_i^{l+1}|\big), \quad (4)$$

where the first term is the total traveled distance and the second term is the total turned angles; $\alpha > 0$ is the chosen weight on turning cost. To synthesize the tentative path $\mathbf{p}_i$, we consider the following optimization problem:

$$\begin{aligned} \mathbf{min}_{\mathbf{p}_i} \quad & \|(p_{ix}^{L_i} - z_{i\ell}^x,\, p_{iy}^{L_i} - z_{i\ell}^y)\| + \beta \cdot \textsc{Cost}(\mathbf{p}_i) \\ \mathbf{s.t.} \quad & G_i^m(t) \text{ remains connected if } p_i = p_i^l, \\ & \textsc{Collide}(g_i^l,\, g_j(t)) = \bot, \\ & \forall l = 0, 1, \cdots, L_i \text{ and } \forall j \in \mathcal{N}_i^m(t), \end{aligned} \quad (5)$$

where the first term is the tentative progress as the distance from $p_i^{L_i} = (p_{ix}^{L_i}, p_{iy}^{L_i})$ to $(z_{i\ell}^x,\, z_{i\ell}^y)$; $\beta > 0$ is a tuning parameter; and the conditions that along $\mathbf{p}_i$ agent i should remain connected and collision-free to all agents in G_i^m.

The above problem can be solved in four steps: (i) determine the general search area. Given the positions of the marked agents, the general search area $\mathcal{S}_i \subset \mathbb{Z}^2$ satisfies that $g_s = (g_s^x, g_s^y) \in \mathcal{S}_i$ if $\textsc{Neighbor}(g_b,\, g_j(t)) = \top$, for at least one neighbor $j \in \mathcal{N}_i^m(t)$; (ii) remove any grid $g_s \in \mathcal{S}_i$ that $G_i^m(t)$ is not connected if $g_i = g_s$ or $\textsc{Collide}(g_s,\, g_j(t)) = \top$ for any neighbor $j \in \mathcal{N}_i^m(t)$. Thus all elements of $\mathbf{p}_i$ should belong to this general search area; (iii) the augmented-graph construction. Construct a graph $\Xi = (\mathbf{n}_i,\, \mathbf{e}_i,\, \mathbf{w}_i)$ where $\mathbf{n}_i = \mathcal{S}_i \times \{0, \pm\frac{\boldsymbol{\pi}}{2}, \boldsymbol{\pi}\}$ is the set of nodes; $\mathbf{e}_i \subset \mathcal{S}_i \times \mathcal{S}_i$ is the set of edges $(n_1,\, n_2) \in e_i$ if $n_1 = (g_1,\, \theta_1)$, $n_1 = (g_2,\, \theta_2)$ where the grids g_1 and g_2 are adjacent; $\mathbf{w}_i : e_i \to \mathbb{R}^+$ is the weighting function, where $\mathbf{w}_i((g_1,\, \theta_1),\, (g_2,\, \theta_2)) = \underline{d} + \alpha \cdot |\theta_1 - \theta_2|$, where α is defined in (4); (iv) shortest path search. Firstly, locate the initial node $n_0 = (g_0,\, \theta_0) \in \mathbf{n}_i$ that is closest to the current agent state $s_i(t)$. Then construct the shortest paths from n_0 to every other node in $\mathbf{n}_i$ by Dijkstra's algorithm. At last, find the destination $n_d^\star \in \mathbf{n}_i$ that minimizes the cost in (5). Denote the shortest path from n_0 to $n_d^\star$ by $\mathbf{p}_i^\Xi = n_0 n_1 n_2 \cdots n_{L_i-1} n_d^\star$, where $n_l = (g_l,\, \theta_l) \in \mathbf{n}_i$ and L_i is the length of this path. An example is shown in Figure 2.

Give the shortest path $\mathbf{p}_i^\Xi$ above, each element $q_i^l = (s_i^l,\, t_i^l,\, v_i^l)$ of $\mathbf{p}_i$ can be derived by setting $s_i^l = (g_l^x \cdot \underline{d},\, g_l^y \cdot \underline{d},\, \theta_l)$ and $v_i^l = V_i$, $\forall l = 0, \cdots, L_i$, and t_i^l is computed by:

$$t_i^{l+1} = t_i^l + \frac{\underline{d}}{v_i^l} + \frac{|\theta_i^{l+1} - \theta_i^l|}{W_i}, \quad \forall l = 1, 2, \cdots, L_i, \quad (6)$$

which accumulates the time for agent i to move from s_i^l to s_i^{l+1} with linear velocity v_i^l and angular velocity W_i.

If a solution to (5) exists, the resulting $\mathbf{p}_i$ is the tentative path of agent i with the associated marked set $\mathcal{N}_i^m$. Moreover, its tentative *gain* is given by $\chi_i = \|p_i^{L_i} - z_{i\ell}\| - \|p_i(t) - z_{i\ell}\|$. For the ease of notation, we denote this local path synthesis procedure by a single function:

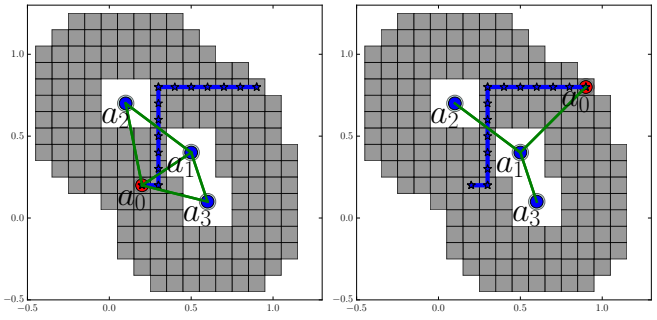

Fig. 2. Grey grids indicate the allowed search area. The blue star-marked path is the optimal path $\mathbf{p}_0$ by (5), for agent a_0 given its marked neighbors a_1, a_2, a_3 and its goal. Notice the change of graph topology of $G_0^m(t)$ and the fact that it remains connected while a_0 moves along $\mathbf{p}_0$.

$$(\mathbf{p}_i,\, \chi_i) = \textsc{Check}(s_i(t),\, \mathcal{N}_i(t),\, z_{i\ell},\, \mathcal{N}_i^m). \quad (7)$$

As a result, agent i *executes* its tentative path $\mathbf{p}_i$ by following and staying within the sequence of grids along $\mathbf{p}_i$.

Lemma 2: Assume that (5) has a solution at time $t_0 > 0$. If all marked neighbors in $\mathcal{N}_i^m$ remain static and agent i executes $\mathbf{p}_i$ until $t_1 > t_0$, then $G_i^m(t)$ remains connected and all agents within $V_i^m(t)$ are collision-free, $\forall t \in [t_0,\, t_1]$.

Proof: Since all marked neighbors in $\mathcal{N}_i^m$ stay static, agent i is the only moving agent within $\mathcal{V}_i^m$. Initially $G_i^m(t_0)$ is connected and all agents within V_i^m are collision-free. While agent i executes $\mathbf{p}_i$, the formulation of (5) ensures that $G_i^m(t)$ remains connected and agent i is collision-free with any marked neighbor. This holds until agent i finishes executing $\mathbf{p}_i$ by reaching $q_i^{L_i}$ at time $t_1 > t_0$. ■

(IV) Path adaptation. The *fourth* building block is the path adaptation algorithm for any active agent while executing its tentative path. Assume that at time $t > 0$ an active agent i may detect another agent $j \in \mathcal{N}$ that does not belong to $\mathcal{N}_i^m$, when its state $s_i(t)$ corresponds to $q_i^{w_0} \in \mathbf{p}_i$ in (3), where $0 < w_0 < L_i$. We consider two cases below:

If $j.\texttt{mode} = \texttt{static}$, then agent i only needs to check if its future path segment is in collision with this static agent j. Its future path segment is given by $\mathbf{p}_i[w_0{:}L_i] = q_i^{w_0} q_i^{w_0+1} \cdots q_i^{L_i}$, where $q_i^l = (s_i^l,\, t_i^l,\, v_i^l)$ is defined in (3). Therefore if $\textsc{Collide}(g_i^w,\, g_j(t)) = \bot$, $\forall w = w_0, w_0 + 1, \cdots, L_i$, it means they will *not* collide and $\mathbf{p}_i$ remains unchanged; otherwise, $\mathbf{p}_i$ is adapted by repeating the synthesis procedure by (7), but with the new neighboring set $\mathcal{N}_i(t)$.

If $j.\texttt{mode} = \texttt{move}$, then agent j is also moving and executing its path $\mathbf{p}_j$. In this case, it is more complicated to check whether they will be in collision. We assume that agent j's position $s_j(t)$ corresponds to $q_j^{v_0} \in \mathbf{p}_j$, where $0 < v_0 < L_j$. Its future path segment is given by $\mathbf{p}_j[v_0{:}L_j] = q_j^{v_0} q_j^{v_0+1} \cdots q_j^{L_j}$, where $q_j^l = (s_j^l,\, t_j^l,\, v_j^l)$ from (3). Given $\mathbf{p}_i[w_0{:}L_i]$ and $\mathbf{p}_j[v_0{:}L_j]$, a potential collision between agents i and j can be detected by the function:

$$\textsc{CollidePath}(\mathbf{p}_i,\, \mathbf{p}_j) = \bot, \quad (8)$$

if $\textsc{Collide}(p_i^w,\, p_j^v) = \perp$ and $|t_i^w - t_j^v| < \Delta_t$, for any $p_i^w \in \mathbf{p}_i[w_0{:}L_i]$ and any $p_j^v \in \mathbf{p}_j[v_0{:}L_j]$, where $\Delta_t > 0$ is a design parameter as the allowed time difference, which depends on the estimation accuracy of the time sequences $\{t_i^w\}$ and $\{t_j^v\}$ by (6). Then agents i and j keep their current paths unchanged; otherwise, $\textsc{CollidePath}(\mathbf{p}_i,\, \mathbf{p}_j) = \top$, meaning that they may collide by executing their respective paths. Thus at least one of them should modify its current path, the choice of which agent will be presented later in the EGGs. For now, we assume that agent i is chosen to change its path $\mathbf{p}_i$. Let $w_c \in \{w_0, w_0+1, \cdots, L_i\}$ be the *smallest* index within $\mathbf{p}_i[w_0{:}L_i]$ that a potential collision could happen by (8) and the associated index within $\mathbf{p}_j[v_0{:}L_j]$ is $v_c \in \{v_0, v_0+1, \cdots, L_j\}$. Then agent i would avoid this collision by reducing its speed within the segment $\mathbf{p}_i[w_0{:}w_c]$, while $\mathbf{p}_i[w_c{:}L_i]$ remains unchanged. To find a suitable linear velocity $\nu < v_{\max}$ for elements in $\mathbf{p}_i[w_0{:}w_c]$, we consider the following optimization problem:

$$\begin{aligned} &\textbf{min}_{0<\nu<v_{\max}} \quad |V_i - \nu| \\ &\textbf{s.t.} \quad v_i^l = \nu,\ \forall l = w_0, \cdots, w_c. \\ &\qquad \textsc{Collide}(p_i^w,\, p_s^v) = \perp, \text{ and } |t_i^w - t_s^v| < \Delta_t, \\ &\qquad \forall p_i^w \in \mathbf{p}_i[w_0{:}L_i],\ \forall p_j^v \in \mathbf{p}_j[v_0{:}L_j]. \end{aligned} \tag{9}$$

where V_i is the reference velocity. The conditions above ensure that after adjusting the linear velocity, $\mathbf{p}_i$ and $\mathbf{p}_j$ will not collide by (8). The above problem can be solved as follows: firstly, choose $\nu = \max_{l\in[w_0:w_c]}\{v_i^l\}$ and a proper step size $\delta_v > 0$. Then gradually decrease ν by δ_v and check if the conditions within (9) are fulfilled. If not, repeat this procedure until $\nu = \nu^\star$ is small enough and all conditions within (9) are fulfilled. As a result, $\nu^\star$ is the suitable linear velocity for $\mathbf{p}_i[w_0{:}w_c]$. Moreover, the time instants $\{v_i^w\}$ within $\mathbf{p}_i[w_0{:}L_i]$ are updated according to (6). If $\nu < 0$ and no solution can be found, it means that the initial position of agent i is in collision with parts of agent j's path. Then it changes its mode according to the EGGs defined later.

Then consider that while executing the adjusted path, agent i may meet with *another* moving agent, say $k \in \mathcal{N}_i(t_1)$ at time $t_1 > 0$. Now its corresponding index within $\mathbf{p}_i$ is $w_0' > w_0$. Similar as before, agents i and k exchange their paths $\mathbf{p}_i$ and $\mathbf{p}_k$. Function $\textsc{CollidePath}(\mathbf{p}_i,\, \mathbf{p}_k)$ can be used to check if they will collide in the future. If so, assume that agent i is chosen to adapt its path again and the potential collision is estimated to happen at index w_c' of $\mathbf{p}_i$. Consider the relative position of $q_i^{w_c'}$ and $q_i^{w_c}$ from the previous adjustment: (i) if $w_c' \le w_c$, agent i would reduce its linear velocity within $\mathbf{p}_i[w_0'{:}w_c']$ by the same formulation as (9); (ii) if $w_c' > w_c$, agent i would instead reduce its linear velocity within $\mathbf{p}_i[w_c{:}w_c']$ by the same formulation as (9).

For the ease of notation, we denote this process of adjusting linear velocity by a single function:

$$\mathbf{p}_i = \textsc{SlowDown}(s_i(t),\, \mathbf{p}_i,\, \mathbf{p}_j), \tag{10}$$

which is only applied to the agent that adapts its path. Figure 3 shows an example of applying the above function.

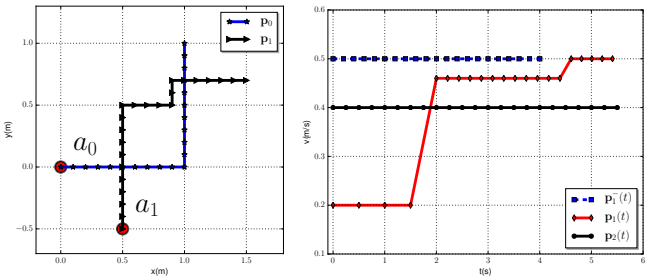

Fig. 3. The left image shows that agents a_0, a_1 have a potential collision given their paths $\mathbf{p}_1, \mathbf{p}_2$ with velocities $0.5m/s$, $0.4m/s$. After applying $\textsc{SlowDown}(\cdot)$ by (10), the velocity profiles of a_0 before (by blue square) and after (by red diamond) are shown in the right image, by which the potential collision is avoided.

(V) Continuous control for tracking. The *fifth* building block is the continuous controller for an active agent to track its tentative path. We rely on the nonlinear control scheme from [13] for unicycle models that handles bounded control inputs and ensures the tracking of a reference trajectory with a provable bounded tracking error. In particular, consider that an active agent i is executing its path $\mathbf{p}_i$ by (3) from q_i^l to q_i^{l+1} at time $t_0 > 0$, where $l \in [0,\, L_i - 1]$. We first construct the reference trajectory $(x_r(t),\, y_r(t),\, \theta_r(t))$ as follows: (i) rotate to the desired orientation while staying at the same position. For $t \in [t_0,\, t_1)$, we set $x_r(t) = x_i^l$, $y_r(t) = y_i^l$, $\theta_r(t) = \theta_i^l + W_i \cdot \text{sgn}(\theta_i^{l+1} - \theta_i^l)(t - t_0)$ and $w_r(t) = W_i$, $v_r(t) = 0$, where $t_1 = t_0 + |\theta_i^{l+1} - \theta_i^l|/W_i$; (ii) forward towards the next grid while keeping the same orientation. For $t \in [t_1,\, t_2]$, we set $x_r(t) = x_i^l + v_i^l \cdot \cos(\theta_r) \cdot (t - t_1)$, $y_r(t) = y_i^l + v_i^l \cdot \sin(\theta_r) \cdot (t - t_1)$, $\theta_r(t) = \theta_i^{l+1}$ and $w_r(t) = 0$, $v_r(t) = v_i^l$, where $t_2 = \underline{d}/v_i^l$. Denote by the saturation function $\text{Sat}_\delta(x) = x$, $\forall |x| \le \delta$ and $\text{Sat}_\delta(x) = \text{sgn}(x)\delta$, $\forall |x| > \delta$. Then the nonlinear control laws are given by: $v_i = v_r \cos(\theta_e) - \text{Sat}_a(k_0 x_e)$ and $w_i = w_r + \frac{f_1(x_e, y_e, \theta_e, t)}{f_2(x_e, y_e, t)} + \text{Sat}_b(k_1 \bar{\theta}_0)$, where $a = v_{\max} - v_i^l$; $x_e = \cos(\theta)(x - x_r) + \sin(\theta)(y - y_r)$; $y_e = -\sin(\theta)(x - x_r) + \cos(\theta)(y - y_r)$; $\theta_e = \theta_r - \theta$; $b > 0$ is chosen such that $|w_i| < w_{\max}$; $k_1,\, k_2 > 0$; $\bar{\theta}_0 = \theta_0 + f_3(x_e,\, y_e,\, t)\, y_e$; the actual expressions of functions $f_1(\cdot)$, $f_2(\cdot)$ and $f_3(\cdot)$ can be found in Section III of [13]. The guarantees for convergence and bounded tracking error are shown in Theorem 1 of [13]. For brevity, we denote this control scheme by the function:

$$(v_i(t),\, w_i(t)) = \textsc{Move}(s_i(t),\, \mathbf{p}_i). \tag{11}$$

3) Graph Grammars: With the above building blocks, we now present the complete graph grammars for the embedded graph $\gamma(t)$, which includes the set of local transition rules with the associated conditions and control modes. We emphasize that they can be applied locally by each agent.

[R.0] At $t = 0$, each agent $i \in \mathcal{N}$ initializes its label by setting $i.\texttt{id} = i$, $i.\texttt{mode} = \texttt{check}$ or $i.\texttt{mode} = \texttt{static}$ randomly, and $i.\texttt{data.nb} = \emptyset$, $i.\texttt{data.pt} = [\,]$, $i.\texttt{gi} = 0$, where $[\,]$ denotes an empty sequence. Moreover, for any agent $j \in \mathcal{N}_i(0)$, it sets $(i,\, j).\texttt{id} = (i,\, j)$.

After the system starts at $t > 0$, *each* agent $i \in \mathcal{N}$ reconstructs its local graph $G_i(t)$ and applies the rules below:

[R.1] If $i.\texttt{mode} = \texttt{check}$, agent i first communicates with *every* neighbor $j \in \mathcal{N}_i(t)$ and checks if $j.\texttt{mode} = \texttt{active}$ and $i \in j.\texttt{data.nb}$. If so, it sets $i.\texttt{mode} = \texttt{static}$ and adds agent j to $i.\texttt{data.nb}$.

After that, if $i.\texttt{mode} = \texttt{check}$ still holds, agent i chooses an allowed marked scheme $\mathcal{N}_i^m$ given $G_i(t)$ and calls the function $\textsc{Check}(s_i(t), \mathcal{N}_i(t), z_i, \mathcal{N}_i^m)$ in (7). If (5) has a solution as the tentative path $\mathbf{p}_i$ and the potential gain χ_i. If $\chi_i > 0$, then it sets $i.\texttt{mode} = \texttt{move}$ and $i.\texttt{data.nb} = \mathcal{N}_i^m(t)$, $i.\texttt{data.gi} = \chi_i$, $i.\texttt{data.pt} = \mathbf{p}_i$. Otherwise if $\chi_i \leq 0$, it sets $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \emptyset$. Otherwise if no solutions to (5) exist or $\chi_i \leq 0$, it sets $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \emptyset$.

[R.2] If $i.\texttt{mode} = \texttt{static}$, agent i stays static by setting $v_i = w_i = 0$. Then it communicates with each neighbor $j \in \mathcal{N}_i(t)$ and checks that if $j.\texttt{mode} = \texttt{active}$, $i \in j.\texttt{data.nb}$, and $j \notin i.\texttt{data.nb}$ hold. If so, it adds agent j to $i.\texttt{data.nb}$. Moreover, for each agent $j \in i.\texttt{data.nb}$, it checks whether $i \in j.\texttt{data.nb}$ still holds. If not, it removes agent j from $i.\texttt{data.nb}$. At last, it checks if $i.\texttt{data.nb} = \emptyset$. If so, it sets $i.\texttt{mode} = \texttt{check}$.

[R.3] If $i.\texttt{mode} = \texttt{move}$, agent i first checks if $j.\texttt{mode} = \texttt{static}$, $\forall j \in i.\texttt{data.nb}$. If not, it stops moving by setting $i.\texttt{mode} = \texttt{check}$ and $i.\texttt{data.nb} = \emptyset$. Otherwise, it executes its tentative path $\mathbf{p}_i$ via the motion controller $(v_i, w_i) = \textsc{Move}(s_i(t), \mathbf{p}_i)$ by (11). As discussed earlier, agent i may encounter other agents, e.g., $j \in \mathcal{N}_i(t)$:

(i) if $j.\texttt{mode} = \texttt{move}$, they exchange their respective gains and tentative paths. Then the agent with higher gain is given higher priority. Assume for now that $i.\texttt{data.gi} < j.\texttt{data.gi}$, implying agent j has higher priority. Then the agent with lower priority, i.e., agent i, calls $\textsc{CollidePath}(\mathbf{p}_i, \mathbf{p}_j)$ by (8) to check if $\mathbf{p}_i$ and $\mathbf{p}_j$ will collide. If so, agent i calls $\textsc{SlowDown}(s_i(t), \mathbf{p}_i, \mathbf{p}_j)$ by (10). If it has a solution, agent i updates its path $\mathbf{p}_i$ by slowing down; otherwise, agent i stops moving by setting $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \emptyset$.

(ii) if $j.\texttt{mode} = \texttt{static}$, agent i checks if it would collide with agent j given its current path $\mathbf{p}_i$. If so, it stops moving by setting $i.\texttt{mode} = \texttt{check}$ and $i.\texttt{data.nb} = \emptyset$.

[R.4] If $i.\texttt{mode} = \texttt{move}$ and $\|p_i(t) - z_{i\ell}\| < c_i$, where $c_i > 0$ is the threshold from Definition 3, agent i has reached its goal point. Then agent i stops moving and resets $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \emptyset$.

It is worth mentioning that the gain comparison in **[R.3]** introduces a fixed priority among the active agents. It means that in the worst-case scenario all agents will slow down or be static except the one with the highest gain.

B. Local Discrete Plan Synthesis

The previous section solves how each agent could move to its current goal point, while obeying the motion constraints. Here we tackle how each agent should choose and update its goal point, in order to fulfill its local task φ_i. The solution relies on the automaton-based model checking algorithm [2], [9]: (i) recall that the complete motion and action model $\mathcal{M}_i$ is given in Section III-B, (ii) then we derive the Büchi automaton $\mathcal{A}_{\varphi_i}$ associated with φ_i [2] by fast translation tools [16]; (iii) we construct the product automaton $\mathcal{P}_i = \mathcal{T}_i \times \mathcal{A}_{\varphi_i}$ by Definition 4.62 of [2]; (iv) lastly a nested Dijkstra's shortest path algorithm [9] is applied to $\mathcal{P}_i$, to find its strongly connected component [2] with the minimal summation cost. We refer the interested readers to [9] for algorithms and implementation details. The infinite discrete plan denoted by τ_i has the prefix-suffix structure:

$$\tau_i = \pi_{i,0}\pi_{i,1}\cdots\pi_{i,k_i-1}(\pi_{i,k_i}\pi_{i,k_i+1}\cdots\pi_{i,K_i})^{\omega}, \qquad (12)$$

where $\pi_{i,k} = (z_{i,k}, a_{i,k}) \in \Pi_i$ where $z_{i,k} \in Z_i$ and $a_{i,k} \in \Sigma_i$, $\forall i = 0, 1, \cdots, K_i$ and $K_i > 0$ is the total length of the prefix and suffix. Note that the k-th element $\pi_{i,k}$ of τ_i for $k > K_i$ can be easily derived from the fact that the suffix is repeated infinitely often. Given the locally-synthesized plans from all agents, we impose the assumption below:

Assumption 1: The plans $\{\tau_i, i \in \mathcal{N}\}$ are *feasible* if $\gamma(t)$ is allowed by Definition 2 when $p(t)$ satisfies $p_i(t) = z_{i,k}$, $\forall i \in \mathcal{N}$ and $\forall k = 0, 1, \cdots$. ■

C. The Complete Solution

When the system starts, each agent $i \in \mathcal{N}$ derives its local plan τ_i from (12) and sets its current goal point $z_{i\ell} = z_{i,0}$; then it follows the transition rules and control laws from the EGGs; by **[R.4]** after it reaches $z_{i,0}$, it becomes static. Then it performs the action $a_{i,k}$ according to the plan τ_i; after the action is accomplished, it remains static until all other agents have reached their respective goal points and finished the corresponding actions. It can be detected through the communication network that all agents are static. Then each agent updates its goal point by $z_{i\ell} = z_{i,1}$ and sets $i.\texttt{mode} = \texttt{check}$, $\forall i \in \mathcal{N}$. Then all agents follow the EGGs to make progress towards this new goal point. This procedure repeats indefinitely as the discrete plans have infinite length. Note that after agent $i \in \mathcal{N}$ reaches z_{i,K_i}, it should set $z_{i\ell} = z_{i,k_i}$ to repeat the plan suffix by (12).

Lemma 3: If $G(0)$ is connected, then $G(t)$ remains connected for $t \geq 0$.

Proof: Since $G(0)$ is connected, there exists at least one path of length N that connects all agents in $G(0)$. Denote by this path $\zeta_0 = a_0 a_1 \cdots a_N$, where agents a_i and a_{i+1} are directly connected by an edge and $a_i \in \mathcal{N}_{i+1}(0)$, $\forall i = 0, 1, \cdots, N-1$. Denote by $t_1 > 0$ the *smallest* time instance that one consecutive pair within ζ_0 is not directly connected anymore. Without loss of generality, let the pair be agents i and j. In other words, agents i and j are the *first* pair within ζ_0 that becomes disconnected directly. Notice that $j \notin \mathcal{N}_i(t_1)$ can only happen in one of the following cases: (i) agent i is moving while agent j is static during $[0, t_1]$. Given the marked neighbors $\mathcal{N}_i^m(0)$, by Definition 6 it holds that $j \in \mathcal{N}_i^m(0)$. Given agent i's path $\mathbf{p}_i$ as derived by (5), Lemma 2 ensures that the sub-graph $G_i^m(t)$ remains connected for $t \in [0, t_1]$ while agent i executes $\mathbf{p}_i$. Thus even though agents i and j are not connected directly at time t_1, they are still connected indirectly within $G_i^m(t_1)$; (ii) both agents i and j are moving during $[0, t_1]$. Given their marked neighbors $\mathcal{N}_i^m(0)$ and $\mathcal{N}_j^m(0)$, by Definition 6 there

must exist a static agent $k \in \mathcal{N}_i(0)$ such that $k \in \mathcal{N}_i^m(0)$ and $k \in \mathcal{N}_j^m(0)$. Given the paths $\mathbf{p}_i$, $\mathbf{p}_j$ as derived by (5), by the same analysis as in case (i), agents i, k remain connected and agents j, k remain connected during $[0,\, t_1]$, yielding that agents i, j is connected indirectly at time t_1. Since the other consecutive pairs in ζ_0 remain connected directly during $[0,\, t_1]$, $G(t)$ remains connected for $t \in [0,\, t_1]$. Now denote by ζ_1 the new path of length N that connects all agents within $G(t_1)$ at time t_1. By the same arguments above for ζ_0, we can show that $G(t)$ remains connected for $t \in [t_1,\, t_2]$, where t_2 is the smallest time instance that one consecutive pair in ζ_1 is disconnected directly. Thus recursively we show that $G(t)$ remains connected, $\forall t \geq 0$. ■

Theorem 4: All local tasks φ_i, $i \in \mathcal{N}$ are satisfied while $\gamma(t) \in \Gamma_d$, $\forall t > 0$.

Proof: Since the workspace is assumed to be unbounded and free of obstacles, at least one agent within $\mathcal{N}$ can be active and make a progress towards its current goal point. The connectivity of $G(t)$ is proved above and the collision avoidance is ensured by the formulation of (5) and (9). Moreover, Assumption 1 ensures that the intermediate configuration of all agents' goal points is feasible and can be reached. At last, by correctness of the discrete plan synthesis process, the execution of the discrete plan τ_i guarantees the satisfaction of φ_i, and thus ensures that the local task φ_i is satisfied, $\forall i \in \mathcal{N}$. ■

V. Simulation and Experimental Study

This section presents the simulation and experimental results of applying the proposed scheme to both simulated and physical multi-robot systems. All algorithms are implemented in Python 2.7. The message passing among the robots are handled by the Robot Operating System (ROS) and each robot is launched as a ROS node. All simulations are carried out on a laptop (3.06GHz Duo CPU and 8GB of RAM).

A. Workspace and Agent Description

The six robots are labeled $\mathfrak{R}_0, \mathfrak{R}_1, \cdots, \mathfrak{R}_5$ and each occupies a disk area of radius $0.05m$. As shown in Figure 4, the communication range $\overline{d}$ is uniformly set to $0.9m$, while the safety distance $\underline{d}$ is set to $0.15m$. Moreover, their reference linear velocity is set to between $0.1m/s$ and $0.3m/s$, under the maximal $0.4m/s$. The angular velocity is set to between $0.4rad/s$ and $0.5rad/s$, under the maximal $0.7rad/s$.

The robots' motion and action model along with their local task specifications are defined as follows: robots $\mathfrak{R}_0$, $\mathfrak{R}_1$ have the local task as surveillance. Robot $\mathfrak{R}_0$ has four points of interest at $(1.5, 1.5)$, $(\text{-}0.2, 1.5)$, $(0, 0)$, $(1.6, 0)$ with labels $\{r_1\}$, $\{r_2\}$, $\{r_3\}$, $\{r_4\}$ and action $\{a_0\}$ as "take photos". Its local task is to surveil r_1, r_2, r_3, r_4 in any order, which can be specified as the LTL formula $\varphi_0 = \wedge_{i=1,\cdots,4}\Box(\Diamond r_i \wedge a_0)$. Robot $\mathfrak{R}_1$ has points of interest close to $\mathfrak{R}_0$'s and its local tasks is similar to φ_0. Robots $\mathfrak{R}_2$, $\mathfrak{R}_3$ have the local tasks for providing services. Robot $\mathfrak{R}_2$ has three points of interest at $(1.2, 0.4)$, $(0.6, 0.6)$, $(0.6, 0.9)$ with labels $\{p_1\}$, $\{p_3\}$, $\{p_2\}$ and action $\{a_1\}$ as "provide services". Its local task is to provide services to p_1, p_2, p_3 in sequence, namely $\varphi_2 = \Box\Diamond((p_1 \wedge a_1) \wedge \Diamond(((p_2 \wedge a_1) \wedge \Diamond(p_3 \wedge a_1)))$. Robot $\mathfrak{R}_3$ has points of interest close to $\mathfrak{R}_3$'s and its task is similar to φ_2. At last, robots $\mathfrak{R}_4$, $\mathfrak{R}_5$ are responsible for transporting goods between goal points. Robot $\mathfrak{R}_4$ has three points of interest $(1.1,\, 1.0)$, $(1.5,\, 1.5)$, $(1.0, 1.0)$ with labels $\{b\}$, $\{s_1\}$, $\{s_2\}$ and actions $\{a_2, a_3\}$ as "load and unload goods". Its local task is to transport goods "A" from storage s_1 to base b and "B" from storage s_2 to base b, i.e., $\varphi_4 = \wedge_{i=1,2}\Box\Diamond((s_i \wedge a_2) \Rightarrow (\neg s_i \,\mathsf{U}\, (b \wedge a_3)))$. Robot $\mathfrak{R}_5$ has three points of interest close to $\mathfrak{R}_4$'s and its local task is similar to φ_4. Initially, the agents start from a line graph.

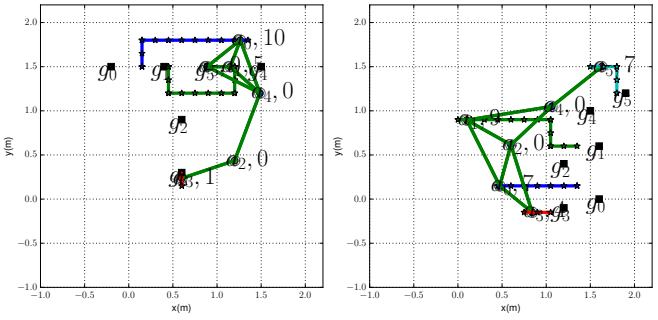

Fig. 4. Snapshots of the simulation results. Moving robots are denoted by red circles while static ones are in blue, labeled by its ID and current gain. Lines marked by stars are tentative paths of the active robots. Black squares represent the goal points, labeled by g_i, $\forall i \in \mathcal{N}$.

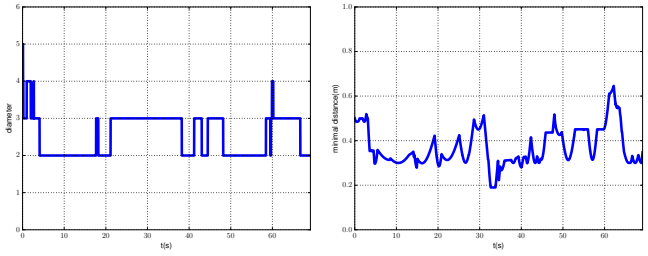

Fig. 5. Evolution of the graph diameter (left) and the minimal distance among the robots. $G(t)$ remains connected as its diameter is always lower than 6; no collision occur as the minimal distance is always above $0.15m$.

B. Simulation Results

After the system starts, each robot first synthesizes its discrete plan τ_i as described in Section IV-B. For instances, robot $\mathfrak{R}_0$'s discrete plan is to visit r_1, r_2, r_3, r_4 in sequence and perform action a_0 at each point, which is then repeated, while robot $\mathfrak{R}_4$'s plan is to load goods "A" at g_1 and unload it at b, then load goods "B" at g_2 and unload it at b, in sequence and repeat. Then they follow the EGGs as described in Section IV-A.3. Most of the time there are three to four robots moving. Figures 4 show some snapshots of how $G(t)$ changes with time. After each robot reaches its current goal point, it performs the planned action. It waits until all other robots become static and then updates its goal point. This procedure continues indefinitely and we simulate the system until $t = 72.5s$ when they have reached the forth goal point. Figure 5 verifies that all motion constraints are fulfilled by showing the evolution of the maximal length of the shortest

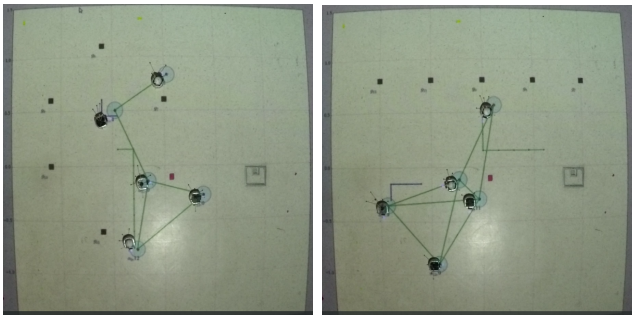

Fig. 6. Snapshots of the experiments at $25s$ and $230s$, where the set of goal points (black squares) are changed. Projected lines are the inter-robot communication link and the tentative paths.

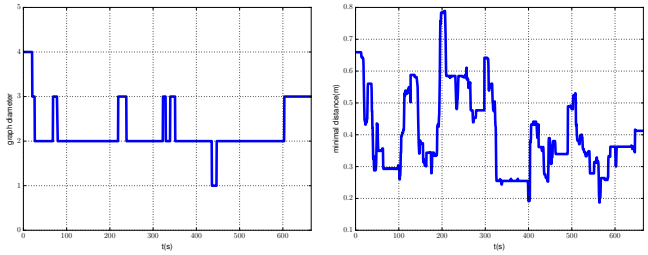

Fig. 7. Evolution of the graph diameter (left) and the minimal distance among the robots (right). Same conclusions can be drawn as in Figure 5.

path between any two vertices within $G(t)$ (i.e., its diameter) and the evolution of the minimal distance between any two robots. The complete simulation video can be found in [22].

C. Experimental Results

We implement the proposed scheme on a team of five Khepera II robots at the GRITS Lab of Georgia Tech, as shown in Figure 6. They are differential-driven wheeled robots that communicates wirelessly with the base station computer. Their position and orientation are tracked in real-time by the OptiTrack system. The message exchange among the robots, between the robots and OptiTrack system are handled by ROS. The robots' points of interest are scattered within the $3m \times 3m$ workspace and designed to be feasible by Assumption 1. The communication radius and safety distance are set to $0.9m$ and $0.15m$. The navigation controller from (11) is tuned properly to ensure that the robots track the synthesized path. We omit the task description here due to limited space, which is similar to the simulation case but no robot actions are modeled in this experiment.

We run the system for 11 minutes and the robots have reached the fourth goal point in their respective plans. The whole experiment is recorded by an overhead camera and communication links among the robots are projected onto the ground. Some snapshots of the experiments are shown in Figure 6, where the robots are heading for different goal points. To verify that both continuous constraints are satisfied, we also plot the diameter of $G(t)$ and the minimal distance between any two robots during the experiment in Figure 7. The complete experiment video can be found in [23], where more detailed descriptions are given.

VI. Conclusion and Future Work

We have presented a hybrid control scheme for multi-robot systems with local tasks, under collision avoidance and connectivity maintenance constraints. Our solution relies on embedded graph control grammars and imposes only local communication and interactions. Future work includes the consideration of static obstacles and dependent local tasks.

References

[1] A. Bhatia, L. E. Kavraki and M. Y. Vardi. Sampling-based motion planning with temporal goals. *IEEE International Conference on Robotics and Automation(ICRA)*, 2689–2696, 2010.

[2] C. Baier and J.-P Katoen. Principles of model checking. *The MIT Press*, 2008.

[3] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins and G. J. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine*, 14(1): 61–70, 2007.

[4] X. Ding, M. Kloetzer, Y. Chen and C. Belta. Automatic deployment of robotic teams. *IEEE Robotics Automation Magazine*, 18: 75–86, 2011.

[5] D. V. Dimarogonas and K. J. Kyriakopoulos. Distributed cooperative control and collision avoidance for multiple kinematic agents. *IEEE Conference on Decision and Control*, 721-726, 2006.

[6] G. E. Fainekos, A. Girard, H. Kress-Gazit and G. J. Pappas. Temporal Logic Motion Planning for Dynamic Mobile Robots. *Automatica*, 45(2): 343–352, 2009.

[7] G. E. Fainekos. Revising temporal logic specifications for motion planning. *IEEE Conference on Robotics and Automation*, 2011.

[8] M. Guo and D. V. Dimarogonas. Bottom-up Motion and Task Coordination for Loosely-coupled Multi-agent Systems with Dependent Local Tasks. *IEEE International Conference on Automation Science and Engineering(CASE)*, 2015. To appear.

[9] M. Guo and D. V. Dimarogonas. Multi-agent Plan reconfiguration under local LTL specifications. *International Journal of Robotics Research*, 34(2): 218-235, 2015.

[10] M. Guo, K. H. Johansson and D. V. Dimarogonas. Motion and Action Planning under LTL Specification using Navigation Functions and Action Description Language. *IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, 240–245, 2013

[11] M. Kloetzer, X. C. Ding and C. Belta. Multi-robot deployment from LTL specifications with reduced communication, *IEEE Conference on Decision and Control(CDC)*, 4867–4872, 2011

[12] S. G. Loizou and K. J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on LTL specifications. *IEEE Conference on Decision and Control(CDC)*, 78–83, 2005.

[13] T. C. Lee, K. T. Song, C. C. Teng. Tracking control of unicycle-modeled mobile robots using a saturation feedback controller. *IEEE Transactions on Control Systems Technology*, 9(2): 305-318, 2001.

[14] J. M. McNew and E. Klavins. Locally interacting hybrid sys- tems with embedded graph grammars. *IEEE Conference on Decision and Control(CDC)*, 6080-6087, 2006.

[15] J. M. McNew, E. Klavins, and M. Egerstedt. Solving Coverage Problems with Embedded Graph Grammars. *Hybrid Systems: Computation and Control, Springer-Verlag*, 413-427, 2007.

[16] P. Gastin and D. Oddoux. Fast LTL to Büchi automaton translation. *International Conference on Computer Aided Verification*, 2001.

[17] D. Pickem and M. Egerstedt. Self-Reconfiguration Using Graph Grammars for Modular Robotics. *IFAC Conference on Analysis and Design of Hybrid Systems*, 2012.

[18] B. Smith, A. Howard, J. M. McNew, J. Wang, M. Egerstedt. Multi-robot deployment and coordination with embedded graph grammars. *Autonomous Robots*, 26(1): 79-98, 2009.

[19] J. Tumova and D. V. Dimarogonas. A receding horizon approach to multi-agent planning from local LTL specifications. *American Control Conference (ACC)*, 1775–1780, 2014.

[20] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, D. Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8): 889–911, 2013.

[21] M. M. Zavlanos, M. B. Egerstedt, and G. J. Pappas. Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9): 1525–1540, 2011.

[22] Simulation video. `https://vimeo.com/136210841`

[23] Experiment video. `https://vimeo.com/137872185`