2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

Multi-robot Deployment From LTL Specifications with Reduced Communication

Marius Kloetzer, Xu Chu Ding, and Calin Belta

Abstract— **In this paper, we develop a computational framework for fully automatic deployment of a team of unicycles from a global specification given as an LTL formula over some regions of interest. Our hierarchical approach consists of four steps: (i) the construction of finite abstractions for the motions of each robot, (ii) the parallel composition of the abstractions, (iii) the generation of a satisfying motion of the team; (iv) mapping this motion to individual robot control and communication strategies. The main result of the paper is an algorithm to reduce the amount of inter-robot communication during the fourth step of the procedure.**

I. INTRODUCTION

Motion planning and control is a fundamental problem that have been extensively studied in the robotics literature [1]. Most of the existing works have focused on point-to-point navigation, where a mobile robot is required to travel from an initial to a final point or region, while avoiding obstacles. Several solutions have been proposed for this problem, including cell decomposition based approaches that use graph search algorithms [1], continuous approaches involving navigation functions and potential fields [2], and sampling-based methods such as Rapidly-Exploring Random Trees (RRTs) [3]. However, the above approaches cannot accommodate complex task specifications, where a robot might be required to satisfy some temporal and logic constraints. In recent years, there has been an increased interest in using temporal logics to specify robot missions [4]–[9]. Temporal logics [10] are appealing because they provide formal, rich, and high level languages for describing complex missions.

To use formal languages and model checking techniques for robot motion planning and control, a fundamental challenge is to construct finite models that accurately capture the robot motion and control capabilities. Most current approaches are based on the notion of abstraction [11]. Enabled by recent developments in hierarchical abstractions of dynamical systems [12]–[14], it is now possible to model the motions of several types of robots as finite transition systems over a cell-based decomposition of the environment. By using equivalence relations such as simulations and bisimulations [15], the motion planning and control problem can be reduced to a model checking or formal synthesis problem for a finite transition system, for which several techniques are readily available [10], [16].

Some recent works suggest that such single-robot techniques can be extended to multi-robot systems through the use of parallel composition (synchronous products) [8], [17]. The main advantage of such a bottom-up approach is that the motion planning problem can be solved by off-the-shelf model checking on the parallel composition followed by canonical projection on the individual transition systems. The two main limitations are the state space explosion and the need for inter-robot synchronization (communication) every time a robot leaves its current region. In our previous work, we proposed bisimulation-type techniques to reduce the size of the synchronous product when the robots are identical and derived classes of specifications that do not require any inter-robot communication [17]. By drawing inspiration from distributed formal synthesis [18], we have also proposed top-down approaches that do not require the parallel composition of the individual transition systems [19]. While cheaper, this method restricts the specifications to regular expressions.

In this paper, we focus on bottom-up approaches based on parallel composition and address one of the limitations mentioned above. Specifically, we develop an algorithm that takes as input a satisfying run of the parallel composition and returns a team control and communication strategy based on a significantly reduced amount of inter-robot communication. Our approach is suboptimal because of two main reasons. First, we assume that each communication moment involves all the agents in the team, rather than allowing for subgroup communication. Second, we use a trial-and-error approach that reduces the total amount of communication, rather than minimizing it by exhaustively exploring all possible communications. Although our method does not consider communication constraints, it reduces the amount of necessary communication during robot deployment.

We integrate this algorithm into a software tool for automatic deployment of unicycles with polyhedral control constraints from specifications given as Linear Temporal Logic (LTL) formulas over the regions of an environment. The tool is freely downloadable from `http://hyness.bu.edu/~software/MRRC.htm`, it has as input a user-defined environment, an LTL formula over some polytopes, the number of unicycles, and their forward and angular velocity constraints. It returns a control and communication strategy for each robot in the team. The tool also implements triangulation and polyhedral operation algorithms from [5], [20], [21], LTL to Büchi conversion [22], and robot abstraction by combining affine vector field computation [14] with input-output regulation [23].

This work was partially supported by the CNCS-UEFISCDI grant PN-II-RU PD 333/2010 at the Technical University of Iasi, Romania, and by ONR MURI N00014-09-1051, ARO W911NF-09-1-0088, AFOSR YIP FA9550-09-1-020 and NSF CNS-0834260 at Boston University.

M. Kloetzer is with the Department of Automatic Control and Applied Informatics, Technical University "Gheorghe Asachi" of Iasi, Romania `kmarius@ac.tuiasi.ro`

X. C. Ding and C. Belta are with the Department of Mechanical Engineering, Boston University, Boston, MA 02215, USA `{xcding,cbelta}@bu.edu`

II. PRELIMINARIES

Definition 2.1: A deterministic finite transition system is a tuple $T = (Q, q_0, \rightarrow, \Pi, \rho)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\rightarrow \subseteq Q \times Q$ is a transition relation, Π is a finite set of atomic propositions (observations), and $\rho : Q \rightarrow 2^{\Pi}$ is the observation map.

A *trajectory* or *run* of T starting from q_0 is an infinite sequence $r = r(1)r(2)\ldots$ with the property that $r(1) = q_0$, $r(i) \in Q$, and $(r(i), r(i+1)) \in \rightarrow$, $\forall i \geq 1$. A run $r = r(1)r(2)\ldots$ defines an infinite *word* over set 2^{Π}, $w = w(1)w(2)\ldots$, where $w(i) = \rho(r(i))$. With a slight abuse of notation, we denote by $\rho(r)$ the word generated by run r. The set of all words that can be generated by T is called the (ω-) language of T.

In this paper, we consider motion specifications given as formulas in Linear Temporal Logic without the "next" temporal operator (LTL$_{-\mathrm{X}}$) [10], which, for simplicity, we will call LTL. A formal definition for the syntax and semantics of LTL formulas is beyond the scope of this paper. Informally, LTL formulas are recursively defined over a set of atomic propositions Π, by using the standard Boolean operators and a set of temporal operators, which include U ("until"), $\Box$ ("always"), and $\Diamond$ ("eventually"). LTL formulas are interpreted over infinite words over 2^{Π}, as are those generated by transition system T. We note that the class of LTL$_{-\mathrm{X}}$ is not restrictive for our purpose, since for continuous systems it captures the full expressivity power of LTL [5].

Definition 2.2 (Generalized Büchi automaton): A generalized Büchi automaton is a tuple $\mathscr{B} = (S, S_0, \Sigma, \rightarrow_{\mathscr{B}}, F)$, where:

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- Σ is the input alphabet,
- $\rightarrow_{\mathscr{B}} \subseteq S \times \Sigma \times S$ is a transition relation,
- $F \subseteq 2^S$ is the *set of sets* of accepting (final) states.

An infinite input word is accepted by automaton $\mathscr{B}$ if and only if there exists a run produced by that word with the property that all sets from F are visited infinitely often. Due to the complicated acceptance condition (multiple sets have to be infinitely often visited), a generalized Büchi automaton is usually converted into a regular (degeneralized) Büchi automaton, which has only one set of final states, *i.e.* $F \in 2^S$. A conversion algorithm can be found in [22].

For any LTL formula ϕ over a set of atomic propositions Π, there exists a (generalized or regular) Büchi automaton $\mathscr{B}_\phi$ with input alphabet $\Sigma \subseteq 2^{\Pi}$ accepting *all and only* infinite words over Π satisfying formula ϕ [24].

Given a transition system T with set of observations Π and an LTL formula ϕ over Π, a trajectory r of T generating a word that satisfies ϕ can be found by adapting LTL model checking tools [5]. Although r is infinite, it has a finite representation in the form of a finite string called *prefix*, followed by infinite repetitions of another finite string called *suffix* (such a run is said to be in the prefix-suffix form). A run r minimizing a given criterion, such as the necessary storage memory, or the cumulative cost along the transitions of T for finitely many repetitions of the suffix, can be easily found [5].

III. PROBLEM FORMULATION AND APPROACH

We are interested in developing a computational framework for automatic deployment of a team on n identical unicycle robots in a convex polygonal environment. We assume that such a robot is described by (x, y, θ), where $(x, y) \in \mathbb{R}^2$ gives the position vector of the robot's center of rotation, and θ is its orientation. The control $w = [v, \omega]^T \in W \subseteq \mathbb{R}^2$ consists of forward driving (v) and steering (ω) speeds, where W is a polyhedral set capturing control bounds. The kinematics of each unicycle are given by:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{1}$$

The mission is given as a temporal logic statement about a set Π of non-overlapping convex polygonal regions in the environment[1]. Specifically, we consider the following problem:

Problem 3.1: Given a team of n unicycles and a task specification in the form of an LTL formula ϕ over a set of regions of interest Π, design individual communication and control strategies for the mobile robots such that the motion of the team satisfies the specification.

We assume that the unicycles are small, *i.e,* their size is negligible when compared to the environment and the predefined regions in Π. To fully specify Problem 3.1, we need to define what it means for the motion of the team to satisfy an LTL formula ϕ over Π. This means that the word generated during the motion satisfies ϕ. The word generated by a set of n continuous trajectories is a straightforward generalization of the definition of the word generated by a single trajectory [5]. Informally, the word generated by the team motion consists of a sequence of elements of 2^{Π} containing the satisfied propositions (visited regions) as time evolves. In a generated word, there are no finite successive repetition of the same element of 2^{Π}, and infinite successive repetitions of the same element appear if and only if each robot trajectory remains inside a region for all times.

Case study: Consider $n = 3$ unicycles moving in the environment illustrated in Fig. 1 and the following task specification: "Visit regions π_1 and π_4 and π_6 simultaneously, and visit regions π_2 and π_5 simultaneously, infinitely often, while always avoiding π_3." This specification translates to the following LTL formula:

$$\phi = \Box\neg\pi_3 \wedge \Box\Diamond\left((\pi_1 \wedge \pi_4 \wedge \pi_6) \wedge \Diamond(\pi_2 \wedge \pi_5)\right). \tag{2}$$

Note that, since the regions are disjoint, at least three robots are necessary to accomplish the task. Indeed, $\pi_1 \wedge \pi_4 \wedge \pi_6$ requires that π_1, π_4, and π_6 are simultaneously occupied. ■

Our solution to Problem 3.1 combines various techniques from computational geometry, motion planning, and model checking to obtain a sequence of tuples of regions and robot feedback control laws in each of these regions. The produced sequence minimizes the overall number of robot movements

[1]Note that convex non-polygonal regions can be bounded by convex polygonal regions with arbitrary accuracy, and non-convex regions can be divided into adjacent convex regions

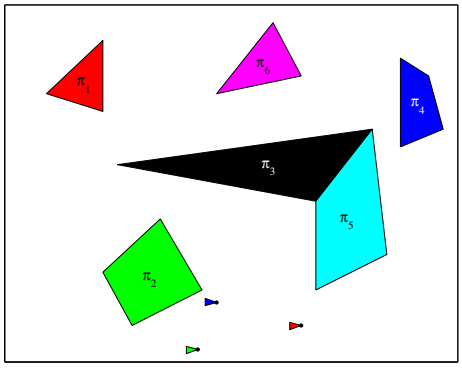

Fig. 1. A polygonal environment, six regions of interest, and the initial deployment of three unicycle robots. Robot 1 is green, robot 2 is blue, and robot 3 is red; there is no relation between the robot and the region colors.

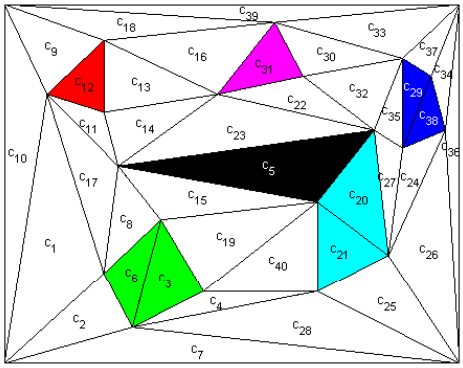

Fig. 2. Triangular partition for the environment from Fig. 1.

among the cells, which corresponds to a minimizing the used robot memory during the deployment. This procedure is described in the next two subsections. After this, we focus on the main contribution of the paper, namely how to use such a sequence to find a team control and communication strategy such that the set of communication (synchronization) moments among the robots is reduced (possibly minimized), while making sure the that the specification is still satisfied.

A. Robot Abstraction

We begin by abstracting the motion capabilities of each robot to a finite transition system. To this end, the environment is first partitioned into convex regions (cells) such that two adjacent cells share exactly one facet, and each region from Π consists of a set of adjacent cells. Such a partition can be constructed with cell decomposition algorithms used in motion planning and computational geometry, *e.g.* constraint triangulations [14] or polytopal partitions [5]. We denote the set of partition elements by $C = \{c_1, c_2, \ldots, c_{|C|}\}$. For example, Fig. 2 shows a triangular partition with 40 cells for the environment from Fig. 1. Then, we use approaches from [14], [23], [25] for reducing the problem of controlling each unicycle with kinematics (1) to control a fully actuated point (the "nose" of the robot) with speed constraints, with details described in technical report [26].

Definition 3.1: The transition system abstracting the motion capabilities of unicycle i, $i = 1, \ldots, n$ has the form $T_i = (Q_i, q_{0i}, \rightarrow_i, \Pi \cup \{\emptyset\}, \rho)$, where:

- $Q_i = C$, (the set of cells from the environment partition);
- $q_{0i} \in C$ is the initial cell of unicycle i;
- The transition relation $\rightarrow_i \in C \times C$ is defined as follows:
  - $(c, c) \in \rightarrow_i$ if we can design a feedback control law keeping robot i in cell c for all times, and
  - $(c, c') \in \rightarrow_i$, if c and c' are adjacent and we can design a feedback control law such that robot i leaves cell c in finite time by crossing the common facet of c and c';
- Π is the set of labels for the regions of interest and $\emptyset$ corresponds to the environmental regions not labeled by symbols from Π,
- ρ is the observation map that associates each cell from the partition with the corresponding proposition from Π or with $\emptyset$.

B. Satisfying Behavior of the Team

Definition 3.2: The transition system $T_G = (Q_G, q_{G0}, \rightarrow_G, \Pi, \rho_G)$ capturing the behavior of the group of n robots is defined as the synchronous product of T_i, $i = 1, \ldots, n$:

- $Q_G = Q_1 \times \ldots \times Q_n$,
- $q_{G0} = (q_{01}, \ldots, q_{0n})$,
- $\rightarrow_G \subset Q_G \times Q_G$ is defined by $((q_1, \ldots, q_n), (q'_1, \ldots, q'_n)) \in \rightarrow_G$ if and only if $(q_i, q'_i) \in \rightarrow_i$, $i = 1, \ldots, n$,
- Π is the observation set,
- $\rho_G : Q_G \rightarrow 2^{\Pi}$, with $\rho_G((q_1, \ldots, q_n)) = \cup_{i=1}^{n} \{\rho(q_i)\}$.

To find a run R of T_G such that the generated word $\rho_G(R)$ satisfies ϕ, we use the the tool developed in [5]. In [17], such a global run was projected to individual robot runs. A run was implemented on a robot by using the feedback controllers corresponding to the transitions (see Sec. III-A). When deploying the team, to ensure the correct transitions in T_G, the robots synchronize (communicate) with each other and wait until every member finishes the previous transition. The synchronization occurs on the boundaries of the cells, when crossing from one cell to another.

Case study revisited: For our example with three unicycles required to accomplish mission specification (2) in the partitioned environment from Fig. 2, we found a minimal satisfying run of T_G, $R =$ prefix,suffix,suffix,..., as follows:

$$\begin{aligned}
\text{prefix} = &\begin{pmatrix} c_7 \\ c_4 \\ c_{28} \end{pmatrix} \begin{pmatrix} c_2 \\ c_3 \\ c_{28} \end{pmatrix} \begin{pmatrix} c_1 \\ c_6 \\ c_{28} \end{pmatrix} \begin{pmatrix} c_{10} \\ c_8 \\ c_{28} \end{pmatrix} \\
&\begin{pmatrix} c_9 \\ c_{17} \\ c_{25} \end{pmatrix} \begin{pmatrix} c_{18} \\ c_{11} \\ c_{26} \end{pmatrix} \begin{pmatrix} c_{16} \\ c_{11} \\ c_{24} \end{pmatrix} \\
\text{suffix} = &\begin{pmatrix} c_{31} \\ c_{12} \\ c_{38} \end{pmatrix} \begin{pmatrix} c_{31} \\ c_{11} \\ c_{38} \end{pmatrix} \begin{pmatrix} c_{31} \\ c_{17} \\ c_{24} \end{pmatrix} \begin{pmatrix} c_{31} \\ c_{8} \\ c_{27} \end{pmatrix} \\
&\begin{pmatrix} c_{31} \\ c_{6} \\ c_{20} \end{pmatrix} \begin{pmatrix} c_{31} \\ c_{8} \\ c_{27} \end{pmatrix} \begin{pmatrix} c_{31} \\ c_{17} \\ c_{27} \end{pmatrix} \begin{pmatrix} c_{31} \\ c_{11} \\ c_{24} \end{pmatrix}
\end{aligned} \tag{3}$$

The robots can be deployed such that their motion corresponds to run R if they synchronize (communicate) whenever

they cross a region boundary [17]. ■

C. Minimizing Communication

The synchronization-based deployment strategy from [17] has two main disadvantages. First, it requires a large amount of communication. Second, it can lead to significant delays, since the moving robots have to wait for each other to synchronize on the boundaries. To deal with these limitations, in this paper we consider the following problem:

Problem 3.2: Given a run R of T_G in prefix-suffix form that satisfies the LTL specification ϕ, find a team control and communication strategy that requires a reduced number of inter-robot synchronizations than in the synchronization-based deployment, while at the same time guaranteeing that the produced motion of the team satisfies ϕ.

Central to our approach to Problem 3.2 is an algorithm that takes as input the satisfying run $R = R(1)R(2)\ldots$ and returns a reduced number of necessary *synchronization moments*, where the i^{th} moment along R is defined as the index i corresponding to $R(i)$. Motivated by the fact that synchronization by stopping and waiting at region boundaries is not always necessary to produce a desired tuple, we consider two types of synchronizations: in a *weak* synchronization, a certain tuple is generated because there exists an instant of time at which the robots are in the corresponding cells; a *strong* synchronization ensures that a sequence of two successive tuples from R is observed. The latter is the stop-and-wait strategy from [17]. Since the run is given in the prefix-suffix form, our algorithmic framework will return a finite set of moments that require synchronization, as well as the type for each synchronization moment.

In our previous work [17], we developed an algorithm to test if an unsynchronized motion of the team can lead to a violation of the specification. If the answer was yes (this is the case in the example considered in this paper), then strong synchronization at each moment along R was the only available option. The algorithm proposed in this paper is therefore a significant improvement over [17].

IV. SOLUTION TO PROBLEM 3.2

Assume that the prefix of R has length $k-1$ and the suffix has length $l-k+1$, with $R(j) = \left(c_j^1, c_j^2, \ldots, c_j^n\right)^T$, for $j = 1,\ldots,l$ (and $R(l+1) = R(k)$, $R(l+2) = R(k+1)$, ...). For simplicity of notation, we assume that whenever an index along R exceeds l, that index is automatically mapped to the set $\{k,\ldots,l\}$, *e.g.,* if $j = l$, then index $j+1$ is replaced with k. We use superscripts for identifying a robot and subscripts for indexing the cells.

For any robot $i = 1,\ldots,n$ and for any moment $j = 1,\ldots,l-1$, cells c_j^i and c_{j+1}^i are either adjacent or identical, and the same is true for cells c_l^i and c_k^i. Recall that from the abstraction process of continuous robot trajectories, R does not contain any successive and finite repetition of a n-tuple.

A. Finding Synchronization Moments

Let $S \subseteq \{1,\ldots,l\}$ be an arbitrary set of synchronization moments. We impose the type of each moment from S by creating a map $\tau : S \to \{weak, strong\}$, where $\tau(j) = weak$ means a *weak* synchronization at moment j, and $\tau(j) = strong$ means a *strong* synchronization at moment j, $\forall j \in S$.

As mentioned in Sec. III-C, a *weak* synchronization at moment j along run R means that the tuple $R(j)$ is reached by the robots, *i.e.,* there is a moment when the robots are in cells $c_j^1, c_j^2, \ldots, c_j^n$, respectively. A *strong* synchronization at moment j along run R means that there is a weak synchronization at j, and additionally the robots synchronously enter the next tuple ($R(j+1)$). In other words, all moving robots i cross from cells c_j^i to cells c_{j+1}^i at the same time.

Note that a strong synchronization at moment j is not equivalent to two weak synchronizations at j and $j+1$. The strong synchronization guarantees that in the resulted team run the tuple $R(j)$ is immediately followed by the tuple $R(j+1)$. However, weak synchronizations at j and $j+1$ guarantee that $R(j)$ and $R(j+1)$ are generated, but there may be a sequence of tuples between them.

For testing the correctness of a set of synchronization moments, we developed a procedure $test\_feasibility_\phi(R,S,\tau)$, which takes as inputs the formula ϕ, the run R, a set S of synchronization moments, and a map τ. The returned output is either "feasible" (set S with map τ guarantees the satisfaction of the formula, no matter how the robots move in between synchronization moments) or "not feasible" (it is possible to violate the formula by imposing just the moments from S with type τ). We postpone the details on $test\_feasibility_\phi(R,S,\tau)$ until Sec. IV-B.

We use Alg. 1 for finding a set S of necessary synchronization moments and a map τ. The intuition behind this algorithm is to start with no synchronization moment ($S = \emptyset$) and iteratively increase S until we obtain a feasible set together with a corresponding map τ. We refer to our technical report [26] for the proof of correctness of Alg. 1, as well as further discussions.

Remark 4.1 (Complexity): Alg. 1 is guaranteed to finish, because in the worst case it returns the set $S = \{1,\ldots,l\}$, meaning that strong synchronizations are needed at every moment (see [26]). The worst case complexity requires $3l(l+3)/2$ iterations of the *test_feasibility* procedure.

Remark 4.2 (Optimality): Alg. 1 can be tailored such that it returns an optimal solution (with respect to a cost defined by weighting the weak and the strong synchronization moments). This can be done by first constructing all possible pairs S,τ (there are 3^l such pairs). Then, these pairs should be ordered according to their associated costs. Finally, the pairs should be tested (in the found order) against the *test_feasibility* procedure, until a feasible response is obtained. The worst case ($S = \{1,\ldots,l\}$ is the only feasible solution) would require 3^l iterations of *test_feasibility*.

B. Testing a Set of Synchronization Moments

Procedure $test\_feasibility_\phi(R,S,\tau)$ consists of the following main steps: (i) construction of an automaton $A_{R,S,\tau}$ generating all the words (infinite sequences of observed propositions) that can result while the robots evolve and obey synchronization moments from S ($A_{R,S,\tau}$ has the form of a Büchi automaton with an observation map); (ii) conversion of

Algorithm 1 Find synchronization moments

Inputs: Run R, formula ϕ

Outputs: Set S, map τ

for $synch\_type \in \{weak, strong\}$ **do**
  $S = \emptyset$, τ undefined
  $lower_{bound} = 1$, $moment = l$
  while $moment \geq lower_{bound}$ **do**
    if $test\_feasibility_{\phi}(R, S, \tau)$ = "feasible" **then**
      Return set S and map τ
    end if
    $S_{temp} = S \cup \{moment, moment+1, \ldots, l\}$
    $\tau_{temp}(i) = \tau(i), \forall i \in S$
    $\tau_{temp}(i) = synch\_type, \forall i \in \{moment+1, \ldots, l\}$
    for $\tau_{temp}(moment) \in \{weak, strong\}$ **do**
      if $test\_feasibility_{\phi}(R, S_{temp}, \tau_{temp})$ = "feasible" **then**
        $S := S \cup \{moment\}$
        $\tau(moment) = \tau_{temp}(moment)$
        $lower_{bound} = moment$
        $moment = l$
        Break "for" loop on τ_{temp}
      else
        $moment := moment - 1$
      end if
    end for
  end while
end for

$A_{R,S,\tau}$ into a degeneralized form, if necessary; (iii) construction of the product automaton between $A_{R,S,\tau}$ and the Büchi automaton $\mathscr{B}_{\neg\phi}$ corresponding to $\neg\phi$; (iv) if the language of this product automaton is empty, the procedure returns "feasible" and otherwise it returns "not feasible".

For step (i), the run R is projected to n individual runs, each corresponding to a specific robot. In each of these individual runs, we collapse the finite successive repetitions of identical states (cells) into a single occurrence (such repetitions mean that the individual robot stays inside a cell). Let us denote the resulted runs by $R^i = q^i_1 q^i_2 \ldots \left[q^i_{k_i} \ldots q^i_{l_i}\right] \ldots$, where prefix has length $k_i - 1$ ($k_i \leq k$) and suffix has length $l_i - k_i + 1$ ($l_i \leq l$), $i = 1, \ldots, n$. Together with individual projections and collapsing, we construct a set of maps $\beta_i : \{1, 2, \ldots, l\} \to \{1, 2, \ldots, l_i\}$, $i = 1, \ldots, n$, mapping each index from run R to the corresponding index from the individual run R_i (*e.g.* $\beta_i(j) = \beta_i(j+1)$ if we have the same i^{th} element in tuples $R(j)$ and $R(j+1)$).

Next, we obtain a generalized Büchi automaton $A_{R,S,\tau}$ (see Def. 2.2) whose accepting runs are the possible sequences of tuples of cells visited during the team evolution. The language (the set of accepted words) of $A_{R,S,\tau}$ contains all possible sequences of elements from 2^{Π} observed by the motion of the team, where each robot moves without synchronization except for the moments from S with map τ.

Definition 4.1: The automaton $A_{R,S,\tau} = (Q_A, q_{A_0}, \to_A, F_A, \Pi, \rho)$, is defined as:

- $Q_A = \{q^1_1, q^1_2, \ldots, q^1_{l_1}\} \times \{q^2_1, q^2_2, \ldots, q^2_{l_2}\} \times \ldots \times \{q^n_1, q^n_2, \ldots, q^n_{l_n}\}$ is the set of states,
- $q_{A_0} = (q^1_1, q^2_1, \ldots, q^n_1)$ is the initial state,
- $\to_A$: $Q_A \to 2^{Q_A}$ is the transition relation,
- $F_A \subset 2^{Q_A}$ is the set of sets of accepting (final) states,
- Π is the observation set,
- $\rho_A : Q_A \to 2^{\Pi}$ is the observation map, $\rho_A(q_1, q_2, \ldots, q_n) = \cup_{i=1}^{n} \{\rho(q_i)\}$.

The *transition relation* $\to_A$ is defined as follows: $\forall q, q' \in Q_A$, with $q = (q^1_{j_1}, q^2_{j_2}, \ldots, q^n_{j_n})$ and $q' = ({q'}^1_{j_1}, {q'}^2_{j_2}, \ldots, {q'}^n_{j_n})$, $(q, q') \in \to_A$ if and only if the following rules are simultaneously satisfied:

(a) $q = q'$ if and only if $j_i = k_i$ and $k_i = l_i$, $i = 1, \ldots, n$;
(b) ${q'}^i_j \in \{q^i_j, q^i_{j+1}\}$ if $j \in \{1, 2, \ldots, l_i - 1\}$, and ${q'}^i_j \in \{q^i_{l_i}, q^i_{k_i}\}$ if $j = l_i$, $i = 1, 2, \ldots, n$;
(c) if $\exists s \in S$ such that $j_i = \beta_i(s)$ for $i \in I \subseteq \{1, \ldots, n\}$, where I is the largest possible such subset, then:
  (1) if $I \neq \{1, \ldots, n\}$, then ${q'}^i_{j_i} = q^i_{j_i}$, $\forall i \in I$;
  (2) if $I = \{1, \ldots, n\}$ and $\tau(s) = strong$, then ${q'}^i_{j_i} = q^i_{\beta_i(s+1)}$, $\forall i \in I$.

Informally, requirements (a) and (b) capture the global movement along the individual runs, by also capturing all the possible interleavings of individual robot motions. Requirement (c) restricts transitions based on synchronization moments. The set F_A is constructed according to Alg. 2, and the details of this procedure can be found in [26].

Algorithm 2 Set of final sets of automaton $A_{R,S,\tau}$

$S_{suffix} = S \cap \{k, \ldots, l\}$
if $S_{suffix} = \emptyset$ **then**
  $F_A = \{q^1_{k_1}, \ldots, q^1_{l_1}\} \times \{q^2_{k_2}, \ldots, q^2_{l_2}\} \times \ldots \times \{q^n_{k_n}, \ldots, q^n_{l_n}\}$
else
  Assume $S_{suffix} = \{s_1, s_2, \ldots, s_{|S_{suffix}|}\}$
  $F_A = \{F_1, F_2, \ldots, F_{|S_{suffix}|}\}$
  for $j = 1, 2, \ldots, |S_{suffix}|$ **do**
    $F_j = \{q^1_{\beta_1(s_j)}, q^2_{\beta_2(s_j)}, \ldots, q^n_{\beta_n(s_j)}\}$
  end for
end if

Once $A_{R,S,\tau}$ is constructed, we check if there exists a generated word of $A_{R,S,\tau}$ that violates the LTL formula (by satisfying the negation of ϕ). This is carried out by the model-checking resembling steps (ii)-(iv) of our procedure. More details on the construction of $A_{R,S,\tau}$, together with an example, are given in [26].

Theorem 4.1: An output of Alg. 1 is a solution to Problem 3.2; Alg. 1 returns an output in a finite number of steps.

The proof of Theorem 4.1 can be found in [26].

C. Communication and Control Strategy

The solution to Problem 3.2 is completed by a deployment strategy such that the synchronization moments from set S with type τ are correctly implemented. For obtaining individual strategies, set S and map τ are first adapted to descriptions suitable for each robot, by constructing for each robot i, $i = 1, \ldots, n$, a memory queue that contains the indices along R^i when synchronization should be enforced and the synchronization type. Then, each robot i follows the infinite

run R^i, by applying feedback controllers and by inspecting its memory queue for deciding when to synchronize with other robots. Due to space constraints, we do not include here the algorithms corresponding to individual robot strategies, and we refer to [26] for these procedures.

V. CASE STUDY

This section concludes the case study illustrated throughout the paper, by applying the procedure described in Sec. IV to the run from (3). We obtain only two weak synchronization moments, at indices 8 and 12 of run R (first and fifth positions of every repetition of suffix). This makes sense, since the propositions satisfied by the team at the two synchronization moments are the two sets of regions ($\{\pi_1, \pi_4, \pi_6\}$ and $\{\pi_2, \pi_5\}$) that are required to be visited for the satisfaction of the formula. The individual runs of the robots are given in (4), where the square brackets delimitate each suffix, and the two weak synchronization moments are marked in bold:

$$\begin{aligned} R^1 &= c_7\, c_2\, c_1\, c_{10}\, c_9\, c_{18}\, c_{16}\, [\mathbf{c_{31}}]\, [\mathbf{c_{31}}] \dots \\ R^2 &= c_4\, c_3\, c_6\, c_8\, c_{17}\, c_{11}\, [\mathbf{c_{12}}\, c_{11}\, c_{17}\, c_8\, \mathbf{c_6}\, c_8\, c_{17}\, c_{11}] \dots \\ R^3 &= c_{28}\, c_{25}\, c_{26}\, c_{24}\, [\mathbf{c_{38}}\, c_{24}\, c_{27}\, \mathbf{c_{20}}\, c_{27}\, c_{24}] \dots \end{aligned} \tag{4}$$

A movie illustrating the robot motion for the case study is available at `http://hyness.bu.edu/~software/unicycles.mp4`. For comparison, if we avoided solving Problem 3.2 and instead used the deployment strategy from [17], we would get the team trajectory illustrated by the movie `http://hyness.bu.edu/~software/unicycles-full-synch.mp4`. In this movie, we can see that the motions of the robots are not as "smooth" as in our proposed approach, and the iterations for each suffix require more time. Our approach is more suitable for real experiments, as robots have less frequent stops at region borders. Additional case studies can be found in [26].

Computation time: The most computationally intensive part of the solution to Problem 3.1 is finding a run R (as in Sec. III-A and III-B). For our case study, this took about 100 minutes on a medium performance computer. In contrast, the solution we proposed for Prob. 3.2 (Sec. IV) took only 30 seconds. To generate a solution for Prob. 3.2, 26 iterations of the *test_feasibility* procedure were performed until the set of synchronization moments was found.

VI. CONCLUSIONS

We presented a fully automated framework for deploying a team of unicycles from a task specified as a linear temporal logic formula over some regions of interest. The approach consists of abstracting the motion capabilities of each robot into a finite state representation, using model checking tools to find a satisfying run, and mapping the solution to a communication and control strategy for each unicycle. The main contribution of the paper is the development of an algorithmic procedure that returns a reduced set of moments when the robots should communicate and synchronize, with the guarantee that the specification is satisfied. A secondary contribution is the integration of this algorithm as part of a fully automatic procedure for deployment of teams of unicycles from specifications given as LTL formulas over regions of interest in an environment.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006, available at `http://planning.cs.uiuc.edu`.

[2] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[3] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[4] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *IEEE Conf. on Decision and Control*, Shanghai, China, 2009, pp. 2222 – 2229.

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

[6] H. Kress-Gazit, G. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *IEEE Int. Conf. on Robotics and Automation*, Rome, Italy, 2007, pp. 3116–3121.

[7] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *IEEE Conf. on Decision and Control*, Paradise Island, Bahamas, 2004, pp. 153–158.

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

[9] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *IEEE Conf. on Decision and Control*, Shanghai, China, 2009, pp. 5997–6004.

[10] E. M. Clarke, D. Peled, and O. Grumberg, *Model checking*. MIT Press, 1999.

[11] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, pp. 971–984, 2000.

[12] R. Burridge, A. Rizzi, and D. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, p. 534, 1999.

[13] D. Conner, H. Choset, and A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies," *Proceedings of robotics: Science and systems II*, 2006.

[14] L. Habets, P. Collins, and J. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Transactions on Automatic Control*, vol. 51, pp. 938–948, 2006.

[15] R. Milner, *Communication and concurrency*. Prentice-Hall, 1989.

[16] G. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 279–295, 1997.

[17] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[18] M. Mukund, "From global specifications to distributed implementations," in *Synthesis and control of discrete event systems*. Kluwer, 2002, pp. 19–34.

[19] Y. Chen, X. Ding, A. Stefanescu, and C. Belta, "A formal approach to deployment of robotic teams in an urban-like environment," in *10th International Symposium on Distributed Autonomous Robotics Systems (DARS 2010)*, 2010 (to appear).

[20] J. Shewchuk, "Triangle," `http://www.cs.cmu.edu/~quake/triangle.html`.

[21] K. Fukuda, "cdd/cdd+ package," `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/`.

[22] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Conf. on Computer Aided Verification*, ser. Lecture Notes in Computer Science, no. 2102. Springer, 2001, pp. 53–65.

[23] J. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, Leuven, Belgium, 1998.

[24] P. Wolper, M. Vardi, and A. Sistla, "Reasoning about infinite computation paths," in *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, E. N. et al., Ed., Tucson, AZ, 1983.

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

[26] M. Kloetzer, X. C. Ding, and C. Belta, "Multi-robot deployment from LTL specifications with reduced communication," technical report, 2011, available at `http://arxiv.org/abs/1108.3240`.