Correct-by-Construction Control Synthesis for Multi-Robot Mixing

Yancy Diaz-Mercado[1], Austin Jones[2], Calin Belta[2,3] and Magnus Egerstedt[1]

***Abstract*— This paper considers the problem of controlling a team of heterogeneous agents to conform to high-level interaction (coordination, sensing, and communication) missions. We consider interactions that can be specified via symbolic inputs from the braid group. We define a novel specification language, called Braid Temporal Logic (BTL), that allows us to specify rich, temporally-layered tasks involving agents' locations in an environment, their relative positions to each other, and frequency of location swaps and information exchanges between agents. We use techniques from formal methods to generate symbolic inputs that conform to a given BTL specification and use recently developed hybrid optimal control synthesis techniques to enact the synthesized pattern. The generated trajectories are provably guaranteed to be collision-free, respect physical boundaries of the agents' mission space, and to satisfy the high-level mission. Results are validated via implementation on a team of wheeled robots.**

I. INTRODUCTION

In this paper, we consider the problem of enforcing high-level coordination, sensing, and communication missions for a team of robots with heterogeneous sensing capabilities. Many of the existing works on multi-agent cooperation use control theory, optimization, and graph theory to enforce team properties such as connectivity of the communication graph [1], optimal coverage [2], or optimal routing [3]. Here, we propose a framework for specifying and enforcing a general class of high-level mission specifications that subsumes many common tasks and can be used to address combinations of these common tasks, e.g., "ensure that the environment remains covered and that every agent shares its data with at least two other agents."

Temporal logic (TL) [4] has been used in robotics to generate control policies for single agents that are guaranteed to satisfy a given high-level mission (TL formula) [5], [6]. Less research exists on using TL to coordinate teams of agents [7], [8]. This paper represents one of the first examples (besides [9], [10]) in which temporal logic has been used to coordinate teams of agents and low-level controls have been synthesized that are guaranteed to satisfy the given mission. In contrast to the cited works, this paper explicitly considers interactions between agents rather than requirements over the absolute positions of the agents.

These interactions are formally encoded as members of the algebraic braid group [11] and used as symbolic inputs to multi-robot controllers for achieving rich interaction patterns. Hybrid controllers called *braid controllers* [12] can be synthesized to safely execute these braids. In this paper, we address the question of how to generate symbolic inputs (sequences of braid generators) that can be enacted by braid controllers to satisfy mission specifications. We define a new model, called the braid transition system (BTS), that encapsulates how enacting these inputs affects the state of the multi-robot system. We define a new specification language over BTSs, called braid temporal logic (BTL), that can describe properties involving agents' locations and communication. These properties can be interleaved via Boolean and temporal operators to form high-level missions, e.g., "the distance between agents 2 and 3 is never greater than δ. At least two different agents survey location 4. If agent 1 communicates with agent 2, then agent 2 passes the message to agent 4 or 5."

We present provably correct techniques for generating a braid string that is guaranteed to enforce a given BTL specification, number of agents, and maximum number of allowed symbolic inputs. In particular, we present a novel, computationally-efficient technique, in which the BTS, whose size grows combinatorially with the number of agents, is never constructed explicitly. We demonstrate our method with an end-to-end case study. We define a BTL specification, use our synthesis algorithm to generate a satisfying braid string, generate a set of minimum tracking error braid controllers, and implement them on a team of wheeled robots.

II. ROBOTIC MIXING USING BRAIDS

We presented a framework in [12] that used generators of the *N-strand Braid Group* [11] as symbolic

This work was supported by a grant from the U.S. Air Force Office for Scientific Research, NSF NRI-1426907 and NSF CMMI-1400167.

[1]Authors are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA yancy.diaz@gatech.edu, magnus@gatech.edu

[2]Authors are with the Division of Systems Engineering, Boston University, Boston, Massachussets, USA austinmj@bu.edu, cbelta@bu.edu

[3]Authors are with the Department of Mechanical Engineering, Boston University

inputs to an N-agent robot team to encode desired interaction patterns. In general, there are N generators in the N-strand braid group, with σ_0 being the trivial generator (i.e., no interactions) and σ_k describing the interaction between the "strands" (or connecting lines) k and $k+1$, with $k = 1,\dots,N-1$. Fig. 1a illustrates the generators of the 4-strand braid group. Complex, temporally sequenced interactions can be constructed by concatenating generator symbols to form *braid strings* (Fig. 1b), which are themselves members of the braid group. We denote the length of a braid string M as the number of concatenated symbols. This length provides a notion of the amount of *mixing*, or interactions, that the team of agents achieves. We denote the set of all braid strings of length M that can be generated from the N-strand braid group as Σ_N^M. For example, a braid string of length four $\sigma \in \Sigma_N^4$ has the form $\sigma = \sigma_a \cdot \sigma_b \cdot \sigma_c \cdot \sigma_d$, $a,b,c,d \in \{0,\dots,N-1\}$.

Geometrically speaking, elements of the braid group will represent bijections between sets of agent positions. The intermediary points in a braid to and from which agents are bijectively mapped will be referred to as *braid points*. These points correspond to sets of spatio-temporal constraints that are decided by the underlying application, or introduced as intermediary waypoints for the sake of enforcing interactions. We will denote $\mathcal{P}_i \in \mathbb{R}^{N\times 2}$ to be a matrix containing the set braid points at step i, i.e., the set of positions the agents must occupy at some time t_i. If the mapping encoded by the symbol σ_1 is applied to transition from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$, then the agent occupying position $[\mathcal{P}_i]_1$ (resp. $[\mathcal{P}_i]_2$) at step i will occupy position $[\mathcal{P}_{i+1}]_2$ (resp. $[\mathcal{P}_{i+1}]_1$) at step $i+1$, where $[\cdot]_j$ is used to denote the j^{th} row of the matrix.

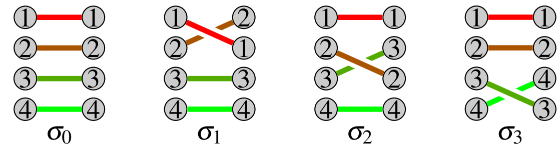
The elements of the braid group themselves do not have a fixed geometric interpretation. For the sake of robotic navigation, we interpret the "strands" of the braid as the path agents should follow to achieve their interactions. The positions of the agents at the end of the path are given by the previously described bijection.

We now recall some definitions originally stated in [12], but included here for completeness. Consider a collection of N planar robots attempting to achieve the mixing strategy given by a symbolic input.

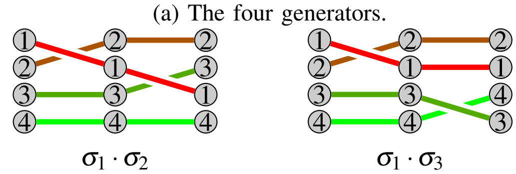
Definition 1 (Braid Controller [12]): A multi-robot controller is a *braid controller* if the resulting trajectories satisfy the spatio-temporal constraints imposed by the braid points, and are also collision-free for all collision-free initial conditions. □

Definition 2 (Mixing Limit [12]): The *mixing limit* $M^\star$ is the largest integer M such that there exists a braid controller for every string in Σ_N^M. □

Whenever two strands of the braid associated with a given braid string cross, the two associated agents will have to interact. The mixing limit therefore serves as an input-independent bound on how much mixing is achievable for a given team of agents operating in a given environment. In [12], we make some assumptions that allow us to compute bounds on the mixing limit.



(a) The four generators.



(b) Generator concatenations.

Fig. 1: 4-Strand Braid Group Example.

Theorem 1 (Mixing Limit Theorem [12]): Given the safety separation δ and bounds on the linear velocity along the braid such that $v(t) \in [0, v_{max}]\ \forall t$, the mixing limit $M^\star$ for N-agent braids that can be performed in a space of height h and length ℓ in time T is bounded above by

$$M^\star \leq \min\left\{\frac{\ell\sqrt{4h^2-\delta^2(N-1)^2}}{\delta h}, \frac{2(N-1)(v_{\max}T-(\ell+\delta))-1}{2h}\right\}.$$

Proof: See [12]. ■

Theorem 1 provides a compact expression with which to obtain an upper bound on the mixing limit that abstracts away strand geometry. It provides a notion of whether or not desirable mixing levels are achievable in the space, regardless of what the actual movement patterns to achieve these mixing levels are used (encoded in the braid string of length $M \leq M^\star$).

Here, we use this bound on the mixing limit to generate policies over the braid strings that conform with a user provided specification through a flavor of temporal logic we call *braid temporal logic* (defined in Section IV). First, we present some preliminaries on temporal logic and formal methods in Section III.

III. Temporal Logic and Automata

The set of all finite and set of all infinite words over alphabet Ω are denoted by Ω^* and Ω^∞, respectively.

A *deterministic transition system* [4] (DTS) is a tuple $TS = (Q, q_0, Act, Trans)$, where Q is a set of states, $q_0 \in Q$ is the initial state, Act is a set of actions, and $Trans \subseteq Q \times Act \times Q$ is a transition relation. A *labeled DTS* is a tuple $TS = (Q, q_0, Act, Trans, AP, L)$ where AP is a set of atomic propositions, and the labeling function $L : Q \to 2^{AP}$ maps states to propositions. An input sequence $a =$

$a^0a^1\ldots \in Act^*$ induces a *run* $r = q^0q^1q^2\ldots \in Q^*$ such that $q^0 = q_0$ and $(q^i, a^i, q^{i+1}) \in Trans$. The *trace* of a run of a labeled transition system is a word $w = w^0w^1\ldots \in (2^{AP})^*$ such that $w^i = L(q^i)$.

A *syntactically co-safe linear temporal logic* (scLTL) formula over a set AP is inductively defined as [13]:

$$\phi := p|\neg p|\phi\vee\phi|\phi\wedge\phi|\phi\ \mathscr{U}\ \phi|\ \bigcirc\ \phi|\ \diamondsuit\ \phi, \quad (1)$$

where $p \in AP$ and ϕ is an scLTL formula. The logical operators $\vee, \wedge$, and $\neg$ are disjunction, conjunction, and negation, respectively, and the temporal operators $\mathscr{U}$, $\bigcirc$, and $\diamondsuit$ are until, next, and eventually, respectively. We also use Boolean implication $\Rightarrow$, where $(\phi_1 \Rightarrow \phi_2) = (\neg\phi_1 \vee \phi_2)$. The logic scLTL is defined over words $w = w^0w^1\ldots \in (2^{AP})^*$. The notation $w \models \phi$ is used to mean that w *satisfies* an scLTL formula ϕ. The *language* of ϕ is $\mathscr{L}(\phi) = \{w|w \models \phi\}$.

A *(deterministic) finite state automaton* (FSA) is a tuple $FSA = (\Omega, \Pi, \Omega_0, F, \Delta_{FSA})$ where Ω is a finite set of states, Π is an input alphabet, $\Omega_0 \subseteq \Omega$ is a set of initial states, $F \subseteq \Omega$ is a set of final (accepting) states, and $\Delta_{FSA} \subseteq \Omega \times \Pi \times \Omega$ is a deterministic transition relation. An *accepting run* r_{FSA} of an automaton FSA is a sequence of states $\omega^0\omega^1\ldots\omega^{j+1}$ such that $\omega^{j+1} \in F$ and $(\omega^i, \pi^i, \omega^{i+1}) \in \Delta_{FSA}\ \forall i \in [0, j]$. The *language* of FSA, denoted $\mathscr{L}(FSA)$, is the set of words $w \in \Pi^*$ that lead to an accepting run. Given an scLTL formula ϕ over AP, there exist off-the-shelf algorithms [14] for creating an FSA FSA_ϕ with input alphabet 2^{AP} such that $\mathscr{L}(FSA_\phi) = \mathscr{L}(\phi)$.

The *product automaton* between a labeled deterministic transition system TS and an FSA FSA_ϕ is an FSA $\mathscr{P}_\phi = TS \times FSA_\phi = (\Omega_\mathscr{P}, \chi_0, Act, F_\mathscr{P}, \Delta_\mathscr{P})$ [4], where $\Omega_\mathscr{P} \subseteq Q \times \Omega$, $\chi_0 = (q_0, \omega_0)$, $F_\mathscr{P} \subseteq Q \times F$, and $\Delta_\mathscr{P} = \{(q,\omega), p, (q',\omega')|(q,p,q') \in Trans, (\omega, L(q), \omega') \in \Delta_{FSA}\}$. The state of the automaton at time k is denoted as $\chi^k = (q^k, \omega^k)$. Any accepting word $a = a^0a^1\ldots \in Act^*$ over $\mathscr{P}$ induces a trace w over TS such that $w \models \phi$. Thus, finding a path on TS that satisfies ϕ corresponds to a reachability problem on $\mathscr{P}_\phi$.

The *distance to acceptance* [15], [16] $W : \Omega_\mathscr{P} \to \mathbb{Z}^+$ is defined such that $W(\chi)$ is the minimum number of actions required to drive $\mathscr{P}$ from χ to a state $\chi_f \in F_\mathscr{P}$.

IV. Temporal Logic and the Braid Group

In this section we introduce a formal framework to specify rich, temporally layered multi-robot mixing requirements. We define a special class of deterministic transition systems, called braid transition systems (BTSs), to encapsulate how braid strings affect the mapping between braid points. In Section IV-B we define a new logic, called Braid Temporal Logic, that is interpreted over runs of BTSs.

A. Braid Transition System

In the BTS, we model the set of braid points abstractly as configurations.

Definition 3 (Configuration space): Let $v_N = [1\ \ldots\ N]^T$. The *(mixing) configuration space* for a team of N agents is $Perm(v_N)$ where $Perm(\cdot)$ denotes the set of permutations of the elements of the vectors. □

The configuration associated with $\mathcal{P}_0$ is by definition v_N. A column vector $c \in Perm(v_N)$ corresponds to a configuration of the braid points such that $c(k) = j$ if and only if agent j is mapped to the braid point $[\mathcal{P}_i]_k$ at step i.

Definition 4 (Braid Transition System (BTS)): The *braid transition system* of size N is a deterministic transition system described by the tuple $BTS_N = \big(v_N \cup Perm(v_N)^2, v_N, \Sigma_N, Trans_N\big)$, where $Trans_N \subseteq C_N \times \Sigma_N \times C_N$ is the smallest transition relation that satisfies

$$(v_N, \sigma_0, (v_N, v_N)) \in Trans_N \quad (2a)$$

$$\left.\begin{array}{c}(v_N, \sigma_i, (v_N, c_{22})) \in Trans_N \Leftrightarrow \\ c_{22}(i) = i+1 \wedge c_{22}(i+1) = i \\ \forall i \in 1,\ldots,N-1\end{array}\right\} \quad (2b)$$

$$((c_{11}, c_{12}), \sigma_0, (c_{12}, c_{12})) \in Trans_N \quad (2c)$$

$$\left.\begin{array}{c}((c_{11}, c_{12}), \sigma_i, (c_{21}, c_{21})) \in Trans_N \Leftrightarrow \\ c_{21} = c_{12} \wedge c_{21}(i) = c_{22}(i+1) \\ \wedge c_{21}(i+1) = c_{22}(i) \\ \forall i \in 1,\ldots,N-1\end{array}\right\} \quad (2d)$$

□

Example 1: The braid transition system for two agents, BTS_2, is illustrated in Fig. 2(a). □

A BTS stores one time unit of history, i.e., if the BTS is in state (c_1, c_2), $c_i \in Perm(v_N)$, at time k, then the robots were in configuration c_1 at time $k-1$ and are in configuration c_2 at time k. This allows us to check properties that explicitly involve interactions between agents. Given a run of the braid transition system $r = v_N, (v_N, c^1), (c^1, c^2)\ldots \in (Perm(v_N) \cup Perm(v_N)^2)^*$ we define its *configuration trace* as $r_C = v_N, c^1, c^2, \ldots$. For a finite N, the number of states in BTS_N is $(N!)N+1$ and the number of transitions in $Trans_N$ is $N^2(N!)+N$.

B. Braid Temporal Logic

In order to describe rich, temporally layered requirements on the agents' mixing, we define a new predicate temporal logic, called Braid Temporal Logic (BTL).

Definition 5 (BTL Syntax): The *syntax* of BTL is defined inductively as

$$\begin{aligned}\phi :=\ & A_m p_g | d(A_m, A_\ell) \sim x | A_m A_\ell | \neg Aipj | \neg AiAj \\ & |\phi\vee\phi|\phi\wedge\phi|\phi\ \mathscr{U}\ \phi|\ \bigcirc\ \phi|\ \diamondsuit\ \phi,\end{aligned} \quad (3)$$

where ϕ is a BTL formula, $\sim \in \{<,>\}$, $x \in \mathbb{N}$, and the Boolean and temporal operators are as defined for scLTL in Section III. The predicate $A_m p_g$ means agent m is in position g; $d(A_a, A_\ell) \sim x$ means that the distance between agents m and ℓ is less than (or greater than) x; $A_m A_\ell$ means that agents m and ℓ interact. □

Definition 6 (BTL semantics): The *semantics* of BTL is defined recursively as

$$\begin{array}{lcl} c^i \models A_m p_g & \Leftrightarrow & c^i(g) = m \\ c^i \models \neg A_m p_g & \Leftrightarrow & c^i \not\models A_m p_g \\ c^i \models d(A_m, A_\ell) \sim x & \Leftrightarrow & |f-g| \sim x \text{ where} \\ & & c^i(f) = m \text{ and } c^i(f) = \ell \\ c^i \models A_m A_\ell & \Leftrightarrow & m \text{ and } \ell \text{ swap between} \\ & & c^{i-1} \text{ and } c^i. \\ c^i \models \neg A_m A_\ell & \Leftrightarrow & c^i \not\models A_m A_\ell \\ c^i \models \phi_1 \vee \phi_2 & \Leftrightarrow & c^i \models \phi_1 \text{ or } c^i \models \phi_2 \\ c^i \models \phi_1 \wedge \phi_2 & \Leftrightarrow & c^i \models \phi_1 \text{ and } c^i \models \phi_2 \\ c^i \models \phi_1 \ \mathcal{U} \ \phi_2 & \Leftrightarrow & \exists j \geq i \text{ s.t. } c^j \models \phi_2 \\ & & \text{and } c^k \models \phi_1 \ \forall i \leq k < j \\ c^i \models \bigcirc \phi & \Leftrightarrow & c^{i+1} \models \phi \\ c^i \models \Diamond \phi & \Leftrightarrow & \exists j \geq i \text{ s.t. } c^j \models \phi. \end{array} \tag{4}$$

□

Example 1 (cont'd): For the case of two robots interacting, we can use the BTL formula

$$\phi_{n=2} = \Diamond (A_1 p_2 \wedge \bigcirc A_1 A_2) \tag{5}$$

to describe the property "eventually, Agent 1 is in position 2 and in the next step, Agent 1 and Agent 2 interact."

Example 2: Consider the specification

$$\begin{aligned} \phi_c = & \Diamond (A_3A_4 \vee A_2A_4) \\ & \wedge (\neg(A_3A_4 \vee A_2A_4) \ \mathcal{U} \ A_1A_4) \\ & \wedge ((A_3A_4 \Rightarrow \ \Diamond A_3p_5) \wedge (\neg A_3p_5 \ \mathcal{U} \ A_3A_4) \\ & \vee (A_2A_4 \Rightarrow \ \Diamond A_2p_5) \wedge (\neg A_2p_5 \ \mathcal{U} \ A_2A_4)) \end{aligned} \tag{6}$$

In plain English, this is "agent 4 communicates with agent 2 or 3 after it has communicated with Agent 1. Whichever agent 4 communicates with reports to position 5."

V. Formal Synthesis of Braid Strings

A. Synthesis of Braid Strings from BTL Formulae

In this work, we are both interested verifying rich temporally layered behaviors of interacting robots, and in developing braid controllers that enforce a given BTL specification. We encode this in the following problem.

Problem 1 (Braid String Synthesis): Given a set of N agents and a BTL formula ϕ, find a word $\sigma \in \Sigma_N^M$ such that applying σ to BTS_N will lead to a configuration trace that satisfies ϕ and σ has fewer generators than the mixing limit M^*. □

There are potentially many words σ that satisfy Problem 1. Here, we synthesize the shortest satisfying word. The problem of generating a braid controller from a given word is addressed in [12], and these controllers are used in the case study in Section VI.

The standard approach to solving Problem 1 is to convert it to the problem of scLTL-based synthesis for labeled transition systems. Briefly, we convert a given BTL formula ϕ to an scLTL formula ϕ' by applying a mapping π that maps every predicate in ϕ to a unique atomic proposition. The product automaton $\mathscr{P} = BTS_N \times FSA_{\phi'}$ is constructed and then Djikstra's algorithm is used to produce the shortest accepting word. We ensure that the length of the resulting word is less than the mixing limit. Applying this word to BTS_N will result in a configuration trace r_C that satisfies ϕ.

Example 1 (Cont'd): Fig. 2(b) shows the FSA constructed from (5). Fig. 2(c) shows the product automaton between BTS_2 and the FSA from (5). In Fig. 2(d), we see the path that results from finding the shortest accepting word on $\mathscr{P}$.

B. Language-Guided Synthesis

The number of states in BTS_N scales exponentially with N. We present a procedure, outlined in Algorithm 1, that constructs the part of the product automaton between BTS_N and $FSA_{\phi'}$ necessary to solve Problem 1, denoted $\mathscr{P}_{LG}$, that does not require explicitly constructing BTS_N.

After constructing $FSA_{\phi'}$, we use its accepting states and the set of predicates that enable transitions to these states to enumerate the accepting states F_P of $\mathscr{P}$. Next, we construct $\mathscr{P}_{LG}$ backwards. At each iteration j, the procedure BackStep constructs the set of states K_j such that $W(\chi) = j \ \forall \chi \in K_j$ and connects K_j to $\mathscr{P}_{LG}$. Since $Trans_N$ can be represented by (2), we can enumerate all transitions in BTS_N that would result in a state $(c_1, c_2) \in K_j$. The inputs of the transitions in $FSA_{\phi'}$ can be used to determine whether paths originating from these enumerated states can reach an accepting state in j steps. Finally, after executing BackStep $M^* - 1$ times, we connect the initial state (v_N, ω_0) to $\mathscr{P}_{LG}$ and then trim any states in the graph that are not reachable from (v_N, ω_0).

Proposition 1 (Exactness): The language of $\mathscr{P}_{LG}$ is the set of all paths that will induce BTS_N to satisfy ϕ and respect the mixing limit.

Proof: (Sketch) The result is guaranteed by enforcing the loop invariant $W(\chi) = j \ \forall \chi \in K_j$. ■

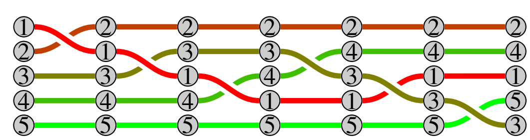
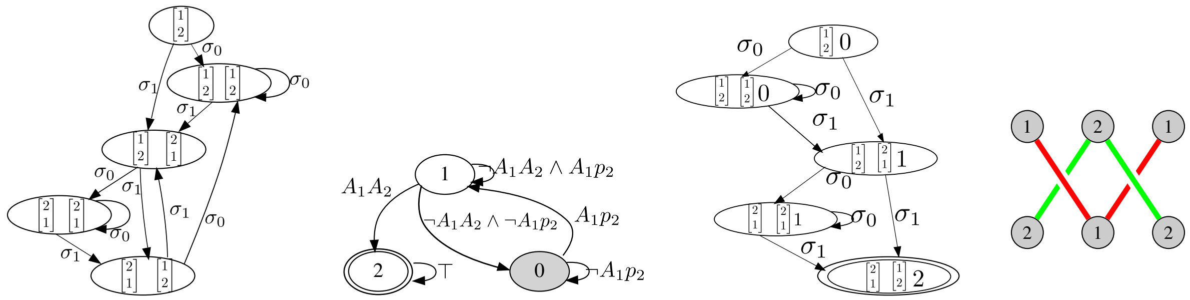


Fig. 3: Shortest braid that satisfies (6).



(a) (b) (c) (d)

Fig. 2: (a) The braid transition system constructed when two agents interact. (b) FSA constructed from (5). The initial state ω_0 is indicated in grey and the accepting state is indicated by the double outline. The edges are annotated with the BTL subformulae whose language is the set of inputs that enable the indicated transition. (c) Product automaton between (a) and (b). Again, the accepting state is indicated with double circles. (d) The braid resulting from finding the shortest accepting path on (c).

Example 2 (Cont'd.): The braid in Fig. 3 satisfies (6) and respects the mixing limit of 8. This braid was generated by Algorithm 1 in 3.7s from an automaton with 845 states. Applying the standard approach (Section V-A) calculated the solution in 939s from an automaton with 5724 states. All calculations were performed on a PC with a 2.6 GHz processor with 7.8 GB RAM.

VI. Case study

We consider a team of 6 agents with heterogeneous sensors that is tasked with the high-level mission "Agent 3 visits location 5 and communicates with agent 1. After agent 3's mission is complete, agent 1 goes to location 6 and agent 6 goes to location 1. Agents 1 and 2 are never more than 3 locations apart for the duration of the mission." The corresponding BTL formula contains 30 symbols and is omitted due to length restrictions. This mission represents part of a much longer mission that has been decomposed into sequential BTL specifications over different windows. This would be the case if the team of agents are moving along a path and have variable requirements at different points along this path.

An interpretation of this specification is that over this time-window, agent 3 is carrying an infrared camera that needs to measure an algal bloom in a pond in location 5. Agent 1 will be tasked with updating a base station along the path during the next time window, so it needs an update from Agent 3 about its recent measurements and needs to be in position 6 during the next time window. Agent 6 needs to be in location 1 so it is ready to measure tree density with its LIDAR in the next time window. Agents 1 and 2 use downward-facing cameras to sense cooperatively, so their proximity requirement is permanent.

We used Algorithm 1 to generate a braid string that satisfies the given specification and meets the mixing limit of 15. The braid string was calculated in 43s from an automaton with 5749 states. This resulted in the string

$$\sigma_{spec} = \{\sigma_1 \cdot \sigma_3 \cdot \sigma_5\} \cdot \sigma_2 \cdot \sigma_3 \cdot \sigma_4 \cdot \{\sigma_3 \cdot \sigma_5\} \cdot \{\sigma_2 \cdot \sigma_4\} \cdot \sigma_1 \quad (7)$$

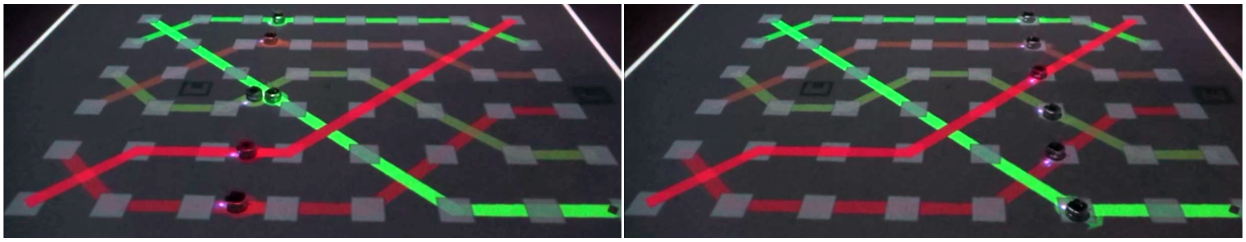
where the grouped substrings may occur simultaneously without violating the specification.

Braid controllers were used to execute the braid string on a team of six Khepera III differential-drive wheeled robots. The robots were controlled over WiFi UDP from an Ubuntu (version 14.04LTS) computer with a 2.8GHz processor and 5.8GB RAM, running ROS (Robot Operating System, Indigo distribution). This computer also received robot state information from ten OptiTrack S250e motion capture cameras. Fig. 4 shows the actual execution of the mixing strategy on the robots at two stages of a 60s total mission time-window. The braid points were uniformly distributed on a rectangular space of length $\ell = 3.4m$ and height $h = 2.5m$. Straight lines were chosen as the braid strands' geometric interpretation. Optimal trajectory tracking controllers are used to minimize the error between the robots actual trajectories and the desired specification-satisfying trajectories, as described in [12].

Fig. 4a shows the agents at a stage of interaction, where an agent exits the safety separation region before the other one enters. Fig. 5a illustrates the instantaneous

Algorithm 1 Language-guided product automaton construction.

function LanguageGuidedConstruction(N, ϕ, M^*)
$FSA_{\phi'}$ =BuildFSA(ϕ)
F_P = ComputeAcceptingState($FSA_{\phi'}$)
$\Omega_P = F_P$; $\Delta_P = \emptyset$; $K_0 = F_P$;
for j = 1 to $M^* - 1$ **do**
  (K_j,Δ_{FSA},Ω_P,Δ_P) = BackStep(K_{j-1},Δ_{FSA},Ω_P,Δ_P)
(Ω_P,Δ_P) = ConnectInitialState(Ω_P,Δ_P)
$\mathscr{P}_{LG} = (\Omega_P, (v_n, c, \omega_0), Act, F_P, \Delta_P)$
return Trim($\mathscr{P}_{LG}$)



(a) The controller is *collision-free* – robots get as close as δ. (b) Braid points are reached simultaneously.

Fig. 4: Actual robots executing the mixing strategy given by (7), which is being projected onto the robot workspace.

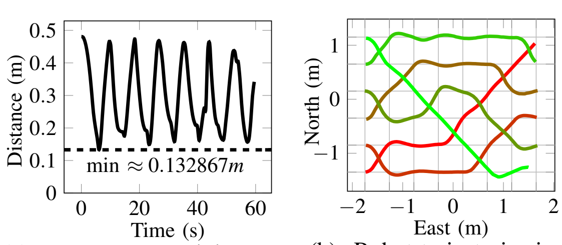
minimum inter-agent distance throughout the execution. It can be seen that the minimum inter-robot distance achieved is approximately $0.132m$ — since the Khepera's have a diameter of $0.13m$, no collisions were observed during execution. Fig. 4b illustrates the robots simultaneously arriving at a set of braid points. Fig. 5b illustrates the robot trajectories in the plane. The optimal tracking controller compensates for deviations due to velocity saturation and the robots' dynamics, thus ensuring the controller remains collision-free and braid points are reached while successfully satisfying the mission specification.

VII. CONCLUSIONS

We approached the problem of controlling a team of agents to conform to high-level interaction patterns and developed a novel specification language, called Braid Temporal Logic (BTL), that describes properties involving agent interactions, relative distance between agents, and the agents' positions in space. We developed a computationally-efficient formal synthesis algorithm that is guaranteed to enforce a given BTL specification. The braid controller generated trajectories are provably guaranteed to be collision-free, respect physical boundaries of the agents' mission space, and to satisfy the high-level mission. The algorithms and controllers were validated on a team of robots in a laboratory setting.

REFERENCES

[1] E. Stump, N. Michael, V. Kumar, and V. Isler, "Visibility-based deployment of robot formations for communication maintenance," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 4498–4505.
[2] C. G. Cassandras and X. Lin, "Optimal control of multi-agent persistent monitoring systems with performance constraints," in *Control of Cyber-Physical Systems*, ser. Lecture Notes in Control and Information Sciences, D. C. Tarraf, Ed. Springer International Publishing, 2013, vol. 449, pp. 281–299.
[3] S. Smith, M. Pavone, M. Schwager, E. Frazzoli, and D. Rus, "Rebalancing the rebalancers: optimally routing vehicles and drivers in mobility-on-demand systems," in *American Control Conference (ACC), 2013*, June 2013, pp. 2362–2367.
[4] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
[5] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *Automatic Control, IEEE Transactions on*, vol. 53, no. 1, pp. 287 –297, feb. 2008.
[6] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, Feb. 2009.
[7] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "Formal approach to the deployment of distributed robotic teams," *Robotics, IEEE Transactions on*, vol. 28, no. 1, pp. 158 –171, feb. 2012.
[8] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, Dec 2008, pp. 3953–3958.
[9] M. Guo, J. Tumova, and D. Dimarogonas, "Cooperative decentralized multi-agent control under local ltl tasks and connectivity constraints," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, Dec 2014, pp. 75–80.
[10] C. Vasile and C. Belta, "An automata-theoretic approach to the vehicle routing problem," in *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.
[11] E. Artin, "Theory of braids," *Annals of Mathematics*, vol. 48, no. 1, pp. 101–126, 1947.
[12] Y. Diaz-Mercado and M. Egerstedt, "Inter-Robot Interactions in Multi-Robot Systems Using Braids," 2015, arXiv:1509.04826v1.
[13] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, 2001, volume 19, pages 291-314.
[14] T. Latvala, "Efficient model checking of safety properties," in *Model Checking Software*, ser. Lecture Notes in Computer Science, T. Ball and S. Rajamani, Eds. Springer Berlin / Heidelberg, 2003, vol. 2648, pp. 624–636.
[15] E. Aydin Gol, M. Lazar, and C. Belta, "Language-guided controller synthesis for discrete-time linear systems," in *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, New York, NY, USA, 2012, pp. 95–104.
[16] A. Jones, M. Schwager, and C. Belta, "Information-guided persistent monitoring under temporal logic constraints," in *IEEE American Control Conference (ACC)*, July 2015, pp. 1911–1916.



(a) Instantaneous minimum inter-robot distance. (b) Robot trajectories in the plane.

Fig. 5: Robotic Implementation Data.