Temporal-Logic-Based Reactive Mission and Motion Planning

Hadas Kress-Gazit, *Member, IEEE*, Georgios E. Fainekos, *Member, IEEE*, and George J. Pappas, *Fellow, IEEE*

***Abstract*—This paper provides a framework *to automatically* generate a hybrid controller that *guarantees* that the robot can achieve its task when a robot model, a class of admissible environments, and a high-level task or behavior for the robot are provided. The desired task specifications, which are expressed in a fragment of linear temporal logic (LTL), can capture complex robot behaviors such as search and rescue, coverage, and collision avoidance. In addition, our framework explicitly captures sensor specifications that depend on the environment with which the robot is interacting, which results in a novel paradigm for sensor-based temporal-logic-motion planning. As one robot is part of the environment of another robot, our sensor-based framework very naturally captures multirobot specifications in a decentralized manner. Our computational approach is based on first creating discrete controllers satisfying specific LTL formulas. If feasible, the discrete controller is then used to guide the sensor-based composition of continuous controllers, which results in a hybrid controller satisfying the high-level specification but only if the environment is admissible.**

***Index Terms*—Controller synthesis, hybrid control, motion planning, sensor-based planning, temporal logic.**

I. Introduction

MOTION planning and task planning are two fundamental problems in robotics that have been addressed from different perspectives. Bottom-up motion-planning techniques concentrate on creating control inputs or closed-loop controllers that steer a robot from one configuration to another [1], [2] while taking into account different dynamics and motion constraints. On the other hand, top-down task-planning approaches are usually focused on finding coarse, which are typically discrete, robot actions in order to achieve more complex tasks [2], [3].

The traditional hierarchical decomposition of planning problems into task-planning layers that reside higher in the hierarchy than motion-planning layers has resulted in a lack of approaches that address the integrated system, until very recently. The modern paradigm of hybrid systems, which couples continuous and discrete systems, has enabled the formal integration of high-level discrete actions with low-level controllers in a unified framework [4]. This has inspired a variety of approaches that translate high-level, discrete tasks to low-level, continuous controllers in a verifiable and computationally efficient manner [5]–[7] or compose local controllers in order to construct global plans [8]–[10].

This paper, which expands on the work presented in [11], describes a framework that automatically translates high-level tasks given as linear temporal-logic (LTL) formulas [12] of specific structure into *correct-by-construction* hybrid controllers. One of the strengths of this framework is that it allows for reactive tasks, i.e., tasks in which the behavior of the robot depends on the information it gathers at runtime. Thus, the trajectories and actions of a robot in one environment may be totally different in another environment, while both satisfy the same task.

Another strength is that the generated hybrid controllers drive a robot or a group of robots such that they are *guaranteed* to achieve the desired task if it is feasible. If the task cannot be guaranteed, because of various reasons discussed in Section VI, no controller will be generated, which indicates that there is a problem in the task description.

To translate a task to a controller, we first lift the problem into the discrete world by partitioning the workspace of the robot and writing its desired behavior as a formula belonging to a fragment of LTL (see Section III). The basic propositions of this formula include propositions whose truth value depends on the robot's sensor readings; hence, the robot's behavior can be influenced by the environment. In order to create a discrete plan, a synthesis algorithm [13] generates an automaton that satisfies the given formula (see Section IV). Then, the discrete automaton is integrated with the controllers in [8] and results in an overall hybrid controller that orchestrates the composition of low-level controllers based on the information gathered about the environment at runtime (see Section V). The overall closed-loop system is guaranteed (see Section VI) by construction to satisfy the desired specification, but only if the robot operates in an environment that satisfies the assumptions that were explicitly modeled, as another formula, in the synthesis process. This leads to a natural assume-guarantee decomposition between the robot and its environment.

In a multirobot task (see Section VIII), as long as there are no timing constraints or a need for joint-decision making, each robot can be seen as a part of the environment of all other robots. Hence, one can consider a variety of multirobot missions, such as search and rescue and surveillance, that can be addressed in a decentralized manner.

Manuscript received November 6, 2008; revised April 3, 2009. First published September 15, 2009; current version published December 8, 2009. This paper was recommended for publication by Associate Editor O. Brock and Editor L. Parker upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant EHS 0311123, in part by the National Science Foundation under Grant ITR 0324977, and in part by the Army Research Office under Grant MURI DAAD 19-02-01-0383.

H. Kress-Gazit is with the Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853 USA (e-mail: hadaskg@cornell.edu).

G. E. Fainekos was with the General Robotics, Automation, Sensing, and Perception Laboratory, University of Pennsylvania, Philadelphia, PA 19104 USA. He is now with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85287-0112 USA (e-mail: fainekos@grasp.upenn.edu).

G. J. Pappas is with the General Robotics, Automation, Sensing, and Perception Laboratory, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: pappasg@grasp.upenn.edu).

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

Digital Object Identifier 10.1109/TRO.2009.2030225

This paper expands on the work outlined in [11] in several directions. First, here, we allow the task to include specification that relate to different robot actions in addition to the motion, thus accommodating a larger set of tasks. Second, the continuous execution of the discrete automaton has been modified in order to allow immediate reaction to changes in the state of the environment. Finally, this paper includes a discussion (see Section VI) regarding the strengths, weaknesses, and extensions of the framework, as well as the examples that demonstrate complex tasks.

A. Related Work

The work presented in this paper draws on results from automata theory, control, and hybrid systems. Combining these disciplines is a recurring theme in the area of symbolic control [4].

Motion description languages (MDLs and MDLe) [14]–[17] provide a formal basis for the control of continuous systems (robots) using sets of behaviors (atoms), timers, and events. This formalism captures naturally reactive behaviors in which the robot reacts to environmental events, as well as composition of behaviors. The work presented here is similar in spirit in that we compose basic controllers in order to achieve a task; however, the main differences are the scope of allowable tasks (temporal behaviors as opposed to final goals) and the automation and guarantees provided by the proposed framework.

Maneuver automata [18], which can be seen as a subset of MDLs, are an example for the use of a regular language to solve the motion task of driving a complex system from an initial state to a final state. Here, each symbol is a motion primitive that belongs to a finite library of basic motions, and each string in the language corresponds to a dynamically feasible motion behavior of the system. Our paper, while sharing the idea of an automaton that composes basic motions, is geared toward specifying and guaranteeing higher level and reactive behaviors (sequencing of goals, reaction to environmental events, and infinite behaviors). Ideas, such as the ones presented in [18], could be incorporated in the future into the framework proposed in this paper to allow for complex nonlinear robot dynamics.

The work in [19] describes a symbolic approach to the task of navigating under sensor errors and noise in a partially known environment. In this paper, we allow a richer set of specifications, but perfect sensors and a fully known environment are assumed. Exploring the ideas regarding the use of Markov decision processes (MDPs) and languages to deal with uncertainty is a topic for future research.

This paper assumes that a discrete abstraction of the robot behavior (motion and actions) can be generated. For simple dynamics, such as the kinematic model considered in this paper, there is considerable work supporting this assumption ([8]–[21], etc.); however, such an assumption is harder to satisfy when complex nonlinear dynamics are considered. Results, such as the work reported in [22] and [23], where nonlinear dynamical systems are abstracted into symbolic models, could be used in the future to enhance the work presented in this paper.

The use of temporal logic for the specification and verification of robot controllers was advocated way back in 1995 [24], where computation tree logic (CTL) [12] was used to generate and verify a supervisory controller for a walking robot. In the hybrid systems community, several researchers have explored the use of temporal and modal logic for the design of controllers. Moor and Davoren [25] and Davoren and Moor [26] use modal logic to design switching control that is robust to uncertainty in the differential equations. There, the system is controlled such that it achieves several requirements, such as safety, event sequencing, and liveness, and is assumed to be closed, i.e., the controller does not need to take into account external events from the environment, whereas, here, we assume an open system in which the robot reacts to its environment.

From the discrete systems point of view, [27] describes fixpoint iteration schemes to solve LTL games. The decidability of the synthesis problem for metric temporal logic (MTL), which is a linear time logic that includes timing constraints, is discussed in [28].

This paper is based on ideas presented in [7], [29], and our previous work [5], [6], and [30], such as the use of bisimulations [31], [32] to lift a continuous problem into a discrete domain while preserving important properties, as well as the use of LTL as the formalism to capture high-level tasks. These approaches provide a correct-by-construction controller, whenever one can be found, which is similar to the work presented in this paper. The main difference between these papers and the work presented here is that in [5]–[7] the behavior is nonreactive in the sense that the behavior is required to be the same, no matter what happens in the environment at runtime, for example, visiting different rooms in some complex order. In these papers, other than localization, the robot does not need to sense its surroundings, and does not react to its environment. Here, in contrast, the robot is operating in a possibly adversarial environment to which it is reacting, and the task specification can capture this reactivity. Such behaviors could include, for example, searching rooms for an object and, once an object is found, returning to a base station. The behavior of the robot will be different in each execution since the environment may be different (the object might be in different locations in each execution). As such, this framework provides a plan of action for the robot so that it achieves its task under any allowable circumstance. Furthermore, this framework can handle nonlinear dynamics and motion constraints as long as a suitable discrete abstraction of the motion and a set of low-level controllers can be defined (see Section VI).

II. Problem Formulation

The goal of this paper is to construct controllers for mobile robots that generate continuous trajectories that satisfy the given high-level specifications. Furthermore, we would like to achieve such specifications while interacting, by using sensors, with a variety of environments. The problem that we want to solve is graphically illustrated in Fig. 1.

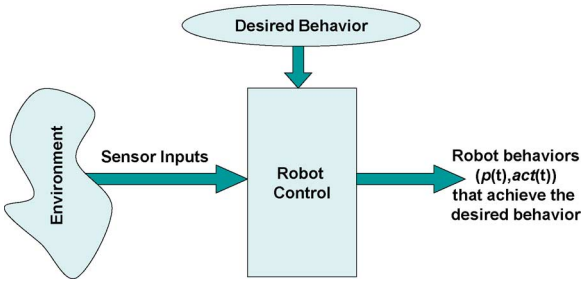

Fig. 1. Problem description.

To achieve this, we need to specify a *robot model*, assumptions on *admissible environments*, and the desired *system specification*.

1) Robot Model: We will assume that a mobile robot (or, possibly, several mobile robots) is operating in a polygonal workspace P. The motion of the robot is expressed as

$$\dot{p}(t) = u(t), \qquad p(t) \in P \subseteq \mathbb{R}^2, \qquad u(t) \in U \subseteq \mathbb{R}^2 \quad (1)$$

where $p(t)$ is the position of the robot at time t, and $u(t)$ is the control input. We will also assume that the workspace P is partitioned using a finite number of cells $P_1, \ldots, P_n$, where $P = \cup_{i=1}^{n} P_i$, and $P_i \cap P_j = \emptyset$, if $i \neq j$. Furthermore, we will also assume that each cell is a convex polygon. The partition naturally induces Boolean propositions $\{r_1, r_2, \ldots, r_n\}$, which are true if the robot is located in P_i

$$r_i = \begin{cases} \text{True}, & \text{if } p \in P_i \\ \text{False}, & \text{if } p \notin P_i. \end{cases}$$

Since $\{P_i\}$ is a partition of P, exactly one r_i can be true at any time.

The robot may have actions that it can perform, such as making sounds, operating a camera, transmitting messages to a base station, etc. In this paper, we allow actions that can be turned on and off at any time, and we encode these actions as propositions $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$

$$a_i = \begin{cases} \text{True}, & \text{if action } i \text{ is being executed} \\ \text{False}, & \text{if action } i \text{ is not being executed.} \end{cases}$$

We define $\text{act}(t) \subseteq \mathcal{A}$ as the set of actions that are active (true) at time t as

$$a_i \in \text{act}(t), \qquad \text{iff } a_i \text{ is true at time } t.$$

Combining these propositions together with the location propositions, we can define the set of robot propositions as $\mathcal{Y} = \{r_1, r_2, \ldots, r_n, a_1, a_2, \ldots, a_k\}$. Note that if two actions cannot be active at the same time, then such a requirement must be added to the system specification either by the user (if it is task-dependent) or automatically from the robot model (when the actions for a specific robot are defined).

2) Admissible Environments: The robot interacts with its environment using sensors, which, in this paper, are assumed to be binary. This is a natural assumption when considering simple sensors, such as temperature sensors or noise-level detectors, which can indicate whether the sensed quantity is above or below a certain threshold, but this is also a reasonable abstraction when considering more complex sensors, such as vision systems, where one can utilize decision-making frameworks or computer-vision techniques. Furthermore, in this paper, we assume that the sensors are perfect, i.e., they always provide the correct and accurate state of the world. Relaxing this assumption is a topic for future research.

The m binary sensor variables $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ have their own (discrete) dynamics that we do not model explicitly. Instead, we place high-level assumptions on the possible behavior of the sensor variables, thus defining a class of admissible environments. These environmental assumptions will be captured (in Section III) by a temporal-logic formula φ_e. Our goal is to construct controllers that achieve their desired specification not for any arbitrary environment, but rather for all possible admissible environments satisfying φ_e.

3) System Specification: The desired system specification for the robot will be expressed as a suitable formula φ_s in a fragment of LTL [12]. Informally, LTL will be used (see Section III) to specify a variety of robot tasks that are linguistically expressed as

1) coverage: "go to rooms P_1, P_2, P_3, P_4 in any order";
2) sequencing: "first go to room P_5, then to room P_2";
3) conditions: "If you see Mika, go to room P_3, otherwise stay where you are";
4) avoidance: "Do not go to corridor P_7."

Furthermore, LTL is compositional, which enables the construction of complicated robot-task specifications from simple ones.

Putting everything together, we can describe the problem that will be addressed in this paper.

Problem 1 [Sensor-based temporal-logic-motion planning]: Given a robot model (1) and a suitable temporal-logic formula φ_e modeling our assumptions on admissible environments, construct (if possible) a controller so that the robot's resulting trajectories $p(t)$ and actions $\text{act}(t)$ satisfy the system specification φ_s in any admissible environment, from any possible initial state.

In order to make Problem 1 formal, we need to precisely define the syntax, semantics, and class of temporal-logic formulas that are considered in this paper.

III. Temporal Logics

Generally speaking, temporal logic [12] consists of propositions, the standard Boolean operators, and some temporal operators. They have been used in several communities to represent properties and requirements of systems, which range from computer programs to robot-motion control. There are several different temporal logics, such as CTL, CTL*, real-time logics, etc. In this paper, we use a fragment of LTL for two reasons. First, it can capture many interesting and complex robot behaviors, and second, such formulas can be synthesized into controllers in a tractable way (see Section IV).

We first give the syntax and semantics of the full LTL and then describe, by following [13], the specific structure of the LTL formulas that will be used in this paper.

A. LTL Syntax and Semantics

Syntax: Let AP be a set of atomic propositions. In our setting, $AP = \mathcal{X} \cup \mathcal{Y}$, which includes both sensor and robot propositions. LTL formulas are constructed from atomic propositions $\pi \in AP$ according to the following grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi.$$

As usual, the Boolean constants True and False are defined as True $= \varphi \vee \neg\varphi$ and False $= \neg$True, respectively. Given negation ($\neg$) and disjunction ($\vee$), we can define conjunction ($\wedge$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$). Furthermore, given the temporal operators "next" ($\bigcirc$) and "until" ($\mathcal{U}$), we can also derive additional temporal operators such as "Eventually" $\Diamond\varphi =$ True$\,\mathcal{U}\,\varphi$ and "Always" $\Box\varphi = \neg\Diamond\neg\varphi$.

Semantics: The semantics of an LTL formula φ is defined over an infinite sequence σ of truth assignments to the atomic propositions $\pi \in AP$. We denote the set of atomic propositions that are true at position i by $\sigma(i)$. We recursively define whether sequence σ satisfies LTL formula φ at position i (denoted $\sigma, i \models \varphi$) by

$$\begin{array}{ll}
\sigma, i \models \pi, & \text{iff } \pi \in \sigma(i) \\
\sigma, i \models \neg\varphi, & \text{iff } \sigma, i \not\models \varphi \\
\sigma, i \models \varphi_1 \vee \varphi_2, & \text{iff } \sigma, i \models \varphi_1 \text{ or } \sigma, i \models \varphi_2 \\
\sigma, i \models \bigcirc\varphi, & \text{iff } \sigma, i+1 \models \varphi \\
\sigma, i \models \varphi_1 \,\mathcal{U}\, \varphi_2, & \text{there exists } k \geq i \text{ such that } \sigma, k \models \varphi_2 \\
 & \text{and for all } i \leq j < k, \text{we have } \sigma, j \models \varphi_1.
\end{array}$$

Intuitively, the formula $\bigcirc\varphi$ expresses that φ is true in the next "step" (the next position in the sequence), and the formula $\varphi_1 \,\mathcal{U}\, \varphi_2$ expresses the property that φ_1 is true until φ_2 becomes true. The sequence σ satisfies formula φ if $\sigma, 0 \models \varphi$. The sequence σ satisfies formula $\Box\varphi$ if φ is true in every position of the sequence, and satisfies the formula $\Diamond\varphi$ if φ is true at some position of the sequence. Sequence σ satisfies the formula $\Box\Diamond\varphi$ if at any position φ will eventually become true, i.e., φ is true infinitely often.

B. Special Class of LTL Formulas

Due to computational considerations mentioned in Section IV, we consider a special class of temporal-logic formulas [13]. We first recall that we have divided our atomic propositions into sensor propositions $\mathcal{X} = \{x_1, \ldots, x_m\}$ and robot propositions $\mathcal{Y} = \{r_1, \ldots, r_n, a_1, \ldots, a_k\}$.

These special formulas are LTL formulas of the form $\varphi = (\varphi_e \Rightarrow \varphi_s)$, where φ_e is an assumption about the sensor propositions, and, thus, about the behavior of the environment, and φ_s represents the desired behavior of the robot. The formula φ is true if φ_s is true, i.e., the desired robot behavior is satisfied, *or* φ_e is false, i.e., the environment did not behave as expected. This means that when the environment does not satisfy φ_e, and is thus not admissible, there is no guarantee about the behavior of the system. Both φ_e and φ_s have the following structure:

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \;;\; \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$$

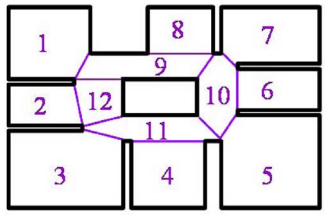

Fig. 2. Workspace of Example 1.

where φ_i^e and φ_i^s are nontemporal Boolean formulas constraining (if at all) the initial value(s) for the sensor propositions $\mathcal{X}$ and robot propositions $\mathcal{Y}$, respectively, φ_t^e represents the possible evolution of the state of the environment and consists of a conjunction of formulas of the form $\Box B_i$, where each B_i is a Boolean formula constructed from subformulas in $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X}$ and where $\bigcirc\mathcal{X} = \{\bigcirc x_1, \ldots, \bigcirc x_n\}$. Intuitively, formula φ_t^e constrains the next sensor values $\bigcirc\mathcal{X}$ based on the current sensor $\mathcal{X}$ and robot $\mathcal{Y}$ values, φ_t^s represents the possible evolution of the state of the robot and consists of a conjunction of formulas of the form $\Box B_i$, where each B_i is a Boolean formula in $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X} \cup \bigcirc\mathcal{Y}$. Intuitively, this formula constrains the next robot values $\bigcirc\mathcal{Y}$ based on the current robot $\mathcal{Y}$ values and on the current and next sensor $\mathcal{X} \cup \bigcirc\mathcal{X}$ values. The next robot values can depend on the next sensor values because we assume that the robot first senses, and then acts, as explained in Section IV, and φ_g^e and φ_g^s represent goal assumptions for the environment and desired goal specifications for the robot. Both formulas consist of a conjunction of formulas of the form $\Box\Diamond B_i$, where each B_i is a Boolean formula in $\mathcal{X} \cup \mathcal{Y}$.

The formula $\phi = (\varphi_g^e \Rightarrow \varphi_g^s)$, which will be discussed in Section IV, is a *generalized reactivity(1)* [GR(1)] formula.

This class of LTL imposes additional restrictions on the structure of allowable formulas. Therefore, some LTL formulas cannot be expressed, for example, $\Diamond\Box\phi$, which is true if at some unknown point in the future, ϕ becomes true and stays true forever.[1] However, according to our experience, there does not seem to be a significant loss in expressivity as most specifications that we encountered can be either directly expressed or translated to this format. More formally, any behavior that can be captured by an implication between conjunctions of deterministic Buchi automata can be specified in this framework [13]. Furthermore, the structure of the formula very naturally reflects the structure of most sensor-based robotic tasks. We illustrate this with a relatively simple example.

Example 1: Consider a robot that is moving in the workspace shown in Fig. 2 consisting of 12 regions labeled $1, \ldots, 12$ (which relate to the robot propositions $\{r_1, \ldots, r_{12}\}$). Initially, the robot is placed either in region 1, 2, or 3. In natural language, the desired specification for the robot is as follows: *Look for Nemo in regions 1, 3, 5, and 8. If you find him, turn your video camera* ON, *and stay where you are. If he disappears again, turn the camera* OFF, *and resume the search.*

[1]The restricted class of LTL can capture a formula that states that *right after something happens*, ϕ becomes true and stays true forever, but it cannot capture that ϕ must become true at some arbitrary point in time.

Since Nemo is part of the environment, the set of sensor propositions contains only one proposition $\mathcal{X} = \{s^{\text{Nemo}}\}$, which becomes True if our sensor has detected Nemo. Our assumptions about Nemo are captured by $\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$. The robot initially does not see Nemo, and thus, $\varphi_i^e = (\neg s^{\text{Nemo}})$. Since we can only sense Nemo in regions 1, 3, 5, and 8, we encode the requirement that in other regions the value of s^{Nemo} cannot change. This requirement is captured by the formula

$$\varphi_t^e = \Box((\neg r_1 \wedge \neg r_3 \wedge \neg r_5 \wedge \neg r_8) \Rightarrow (\bigcirc s^{\text{Nemo}} \Leftrightarrow s^{\text{Nemo}})).$$

We place no further assumptions on the environment propositions, which means that $\varphi_g^e = \Box\Diamond(\text{True})$, completing the modeling of our environment assumptions. Note that the environment is admissible whether Nemo is there or not.

We now turn to modeling the robot and the desired specification, which are captured by $\varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$. We define 13 robot propositions $\mathcal{Y} = \{r_1, \ldots, r_{12}, a^{\text{CameraON}}\}$.

Initially, the robot starts somewhere in region 1, 2, or 3 with the camera; hence

$$\varphi_i^s = \begin{cases} (r_1 \wedge_{i\in\{2,\cdots,12\}} \neg r_i \wedge \neg a^{\text{CameraON}}) \\ \bigvee(r_2 \wedge_{i\in\{1,3,\cdots,12\}} \neg r_i \wedge \neg a^{\text{CameraON}}) \\ \bigvee(r_3 \wedge_{i\in\{1,2,4,\cdots,12\}} \neg r_i \wedge \neg a^{\text{CameraON}}). \end{cases}$$

The formula φ_t^s models the possible changes in the robot state. The first block of subformulas represent the possible *transitions* between regions, for example, from region 1, the robot can move to adjacent region 9 or stay in 1. The next subformula captures the *mutual exclusion* constraint, i.e., at any step, exactly one of $r_1, \ldots, r_{12}$ is true. For a given decomposition of workspace P, the generation of these formulas is easily automated. The final block of subformulas is a part of the desired specification and states that if the robot sees Nemo, it should remain in the same region in the next step, and the camera should be ON. It also states that if the robot does not see Nemo, the camera should be OFF

$$\varphi_t^s = \begin{cases} \begin{cases} \bigwedge \Box(r_1 \Rightarrow (\bigcirc r_1 \vee \bigcirc r_9)) \\ \bigwedge \Box(r_2 \Rightarrow (\bigcirc r_2 \vee \bigcirc r_{12})) \\ \vdots \\ \bigwedge \Box(r_{12} \Rightarrow (\bigcirc r_2 \vee \bigcirc r_9 \vee \bigcirc r_{11} \vee \bigcirc r_{12})) \end{cases} \\ \bigwedge \Box((\bigcirc r_1 \wedge_{i\neq 1} \neg \bigcirc r_i) \\ \quad \vee(\bigcirc r_2 \wedge_{i\neq 2} \neg \bigcirc r_i) \\ \quad \vdots \\ \quad \vee(\bigcirc r_{12} \wedge_{i\neq 12} \neg \bigcirc r_i) \\ \begin{cases} \bigwedge \Box(\bigcirc s^{\text{Nemo}} \Rightarrow \\ \quad (\wedge_{i\in\{1,\cdots,12\}} \bigcirc r_i \Leftrightarrow r_i) \wedge \bigcirc a^{\text{CameraON}}) \\ \quad \bigwedge \Box(\neg \bigcirc s^{\text{Nemo}} \Rightarrow \neg \bigcirc a^{\text{CameraON}}). \end{cases} \end{cases}$$

Finally, the requirement that the robot keeps looking in regions 1, 3, 5, and 8, unless it has found Nemo, is captured by

$$\varphi_g^s = \begin{cases} \Box\Diamond(r_1 \vee s^{\text{Nemo}}) \bigwedge \Box\Diamond(r_3 \vee s^{\text{Nemo}}) \\ \bigwedge \Box\Diamond(r_5 \vee s^{\text{Nemo}}) \bigwedge \Box\Diamond(r_8 \vee s^{\text{Nemo}}). \end{cases}$$

This completes our modeling of the robot specification as well. Combining everything together, we get the required formula $\varphi = (\varphi_e \Rightarrow \varphi_s)$.

Having modeled a scenario using φ, our goal is now to synthesize a controller-generating trajectories that will satisfy the formula if the scenario is possible (if the formula is realizable). Section IV describes the automatic generation of a discrete automaton satisfying the formula, while Section V explains how to continuously execute the synthesized automaton.

IV. Discrete Synthesis

Given an LTL formula, the realization or synthesis problem consists of constructing an automaton whose behaviors satisfy the formula, if such an automaton exists. In general, creating such an automaton is proven to be doubly exponential in the size of the formula [33]. However, by restricting ourselves to the special class of LTL formulas, we can use the efficient algorithm that was recently introduced in [13], which is polynomial O(n^3) time, where n is the size of the state space. Each state in this framework corresponds to an allowable truth assignment for the set of sensor and robot propositions. In Example 1, an allowable state is one in which, for example, the proposition r_1 is true, and all of the other propositions are false. However, a state in which both r_1 and r_2 are true is not allowed (violates the mutual exclusion formula) as is a state in which s^{Nemo} and r_2 are true (the environment assumptions state that Nemo cannot be seen in region 2). Section IX discusses further problem sizes and computability.

In the following, the synthesis algorithm is informally introduced, see [13] for a full description.

The synthesis process is viewed as a game played between the robot and the environment (as the adversary). Starting from some initial state, both the robot and the environment make decisions that determine the next state of the system. The winning condition for the game is given as a GR(1) formula ϕ. The way in which this game is played is that at each step, first, the environment makes a transition according to its transition relation, and then, the robot makes its own transition. If the robot can satisfy ϕ, no matter what the environment does, we say that the robot is winning, and therefore, we can extract an automaton. However, if the environment can falsify ϕ, we say that the environment is winning and the desired behavior is unrealizable.

Relating the formulas of Section III-B to the game mentioned previously, the initial states of the players are given by φ_i^e and φ_i^s. The possible transitions that the players can make are given by φ_t^e and φ_t^s, and the winning condition is given by the GR(1) formula $\phi = (\varphi_g^e \Rightarrow \varphi_g^s)$. Note that the system is winning, i.e., ϕ is satisfied if φ_g^s is true, which means that the desired robot behavior is satisfied, *or* φ_g^e is false, which means that the environment did not reach its goals (either because the environment was faulty or the robot prevented it from reaching its goals). This implies that when the environment does not satisfy φ_g^e, there is no guarantee about the behavior of the robot. Furthermore, if the environment does not "play fair," i.e., violates its assumed behavior $\varphi_i^e \wedge \varphi_t^e$, the automaton is no longer valid.

The synthesis algorithm [13] takes the formula φ and first checks whether it is realizable. If it is, then the algorithm extracts a possible (but not necessarily unique) automaton, which implements a strategy that the robot should follow in order to satisfy the desired task. The automaton that is generated by the algorithm is modeled as a tuple $A = (\mathcal{X}, \mathcal{Y}, Q, Q_0, \delta, \gamma)$.

1) $\mathcal{X}$ is the set of input (environment) propositions.
2) $\mathcal{Y}$ is the set of output (robot) propositions.
3) $Q \subset \mathbb{N}$ is the set of states.
4) $Q_0 \in Q$ is the set of initial states.
5) $\delta : Q \times 2^{\mathcal{X}} \to 2^Q$ is the transition relation, i.e., $\delta(q, X) = Q' \subseteq Q$, where $q \in Q$ is a state and $X \subseteq \mathcal{X}$ is the subset of sensor propositions that are true.
6) $\gamma : Q \to 2^{\mathcal{Y}}$ is the state labeling function, where $\gamma(q) = y$, and $y \subseteq \mathcal{Y}$ is the set of robot propositions that are true in state q.

An admissible input sequence is a sequence $X_1, X_2, \ldots, X_j \in 2^{\mathcal{X}}$ that satisfies φ_e. A run of this automaton under an admissible input sequence is a sequence of states $\sigma = q_0, q_1, \ldots$. This sequence starts at some initial state $q_0 \in Q_0$ and follows the transition relation δ under the truth values of the input propositions, i.e., for all $j \geq 0$, we have $q_{j+1} \in \delta(q_j, X_j)$. An interpretation of a run σ is a sequence $y_0, y_1, \ldots$, where $y_i = \gamma(q_i)$ is the label of the ith state in the run. We use this sequence of labels to construct the discrete path that the robot must follow and to activate/deactivate the different robot actions. As mentioned before, when given a nonadmissible input sequence, i.e., an input sequence that violates any part of φ_e, the automaton is no longer relevant, and we will not be able to construct a correct path for the robot.

Using this synthesis algorithm, there are several ways to extract an automaton that satisfies the LTL formula. It can either be made to be deterministic, i.e., for every input, there will be a unique next state, or nondeterministic, as is the case in this paper. Furthermore, the automaton can be one that always takes the "fastest route" toward the goals, i.e., reaches the goals in the minimum number of transitions, as in this paper, or one that allows all possible routes, even longer ones, as long as they eventually satisfy the task.

Example 2: Revisiting Example 1, Fig. 3 represents the synthesized automaton that realizes the desired behavior. The circles represent the states of the automaton, and the propositions that are written inside each circle are the state's label, i.e., the output propositions that are true in that state. The possible initial states are shown as filled circles. The edges are labeled with the sensor propositions that must be true in order for the transition to be made. Edges with no labels are thus labeled with $\neg s^{\mathrm{Nemo}}$.

As can be seen, the automaton causes the robot to stay where it is and turn the camera ON if it senses Nemo; otherwise, the robot will keep searching for Nemo, with the camera OFF, forever. Note that this automaton is not unique and is nondeterministic. Furthermore, this automaton can only be executed if the environment (Nemo) is behaving according to the assumptions. Thus, if the robot suddenly senses Nemo in region 9, the automaton will not have a suitable transition.

From the interpretation of a run of the automaton, we extract a discrete path for the robot based on the location propositions.

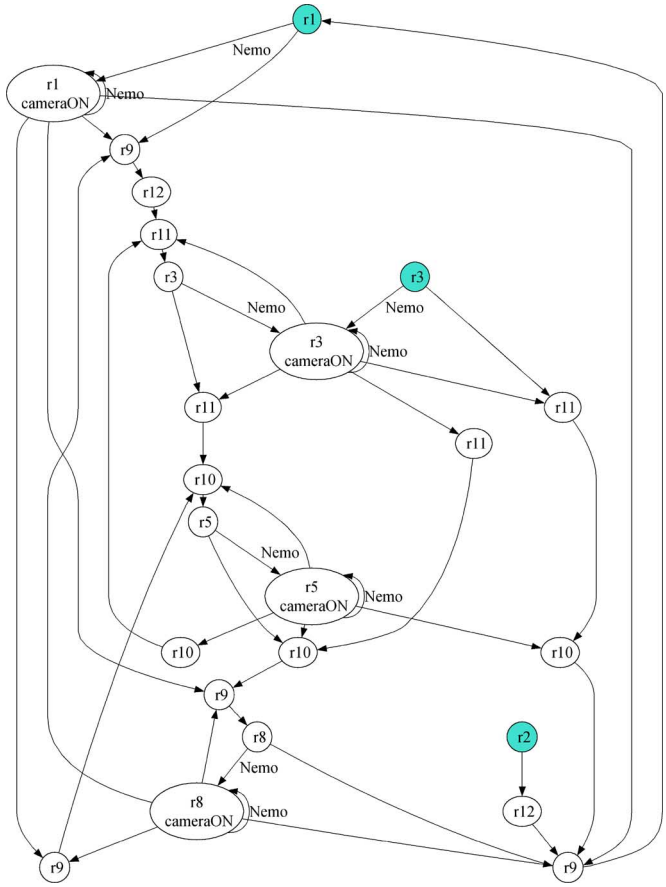

Fig. 3. Synthesized automaton of Example 2.

What is left to do is to transform this discrete path into a continuous trajectory, as is explained in the next section.

V. Continuous Execution

In order to continuously implement the discrete solution of the previous section, we construct a hybrid controller that takes a set of simple controllers and composes them sequentially according to the discrete execution of the automaton.

Following the work in [5], we utilize atomic controllers that satisfy the so-called bisimulation property [31], [32], [34]. Such controllers are guaranteed to drive the robot from one region to another, regardless of the initial state in the region. There are several recent approaches for generating such simple controllers, such as [8], [10], [20], [35], and [36]. We use the framework developed in [8] due to its computational properties and the variety of regions it can be applied to. In this approach, a convex polygon is mapped to the unit disk, then Laplace's equation is solved (in closed form) on the disk, obtaining the potential function, and finally, the solution is mapped back to the polygon. This approach resembles the navigation functions introduced in [37].

The algorithm for executing the discrete automaton is shown in Algorithm 1. Initially, the robot is placed at position p_0 in region i such that $r_i \in \gamma(q_0)$, where q_0 is the initial automaton state satisfying $q_0 \in Q_0$. Furthermore, based on all other

Algorithm 1 Continuous Execution of the discrete automaton

procedure EXECUTE(A, q_0, p_0)
  $CurrState \leftarrow q_0$
  $CurrReg \leftarrow r_i \in \gamma(q_0)$
  $CurrOutput \leftarrow \{a_1, a_2, \cdots\} \in \gamma(q_0)$
  $ActivateDeactivateOutputs(CurrOutput, \mathcal{Y})$
  $p \leftarrow p_0$
  while 1 **do** ▷ Infinite execution
    $InputVal \leftarrow Sense()$
    $NxtState \leftarrow$
      $GetNxtState(A, CurrState, InputVal)$
    $NxtReg \leftarrow r_i \in \gamma(NxtState)$
    $p \leftarrow ApplyController(CurrReg, NxtReg, p)$
    if $p \in P_{NxtReg}$ **then** ▷ Automaton transition
      $CurrState \leftarrow NxtState$
      $CurrReg \leftarrow NxtReg$
      $CurrOutput \leftarrow \{a_1, a_2, \cdots\} \in \gamma(CurrState)$
      $ActivateDeactivateOutputs(CurrOutput, \mathcal{Y})$
    else if $p \in P_{CurrReg}$ **then** ▷ No transition
      Continue
    else
      ERROR - No legal automaton state
    end if
  end while
end procedure

propositions $\gamma(q_0)$, the appropriate robot actions are activated. At each step, the robot senses its environment and determines the values of the binary inputs $\mathcal{X}$. Based on these inputs and its current state, it chooses a successor state NxtState and extracts the next region NxtReg, where it should go from $\gamma(Nxt\text{State})$. Then, it invokes an atomic controller that drives it toward the next region. If the robot enters the next region in this step, then the execution changes the current automaton state, extracts the appropriate actions, and activates/deactivates them. If the robot is neither in the current region nor in the next region, which could happen only if the environment violated its assumptions, then the execution is stopped with an error.

This continuous execution is bisimilar to the discrete execution of the automaton that resembles the continuous execution of a sequence of discrete states that was presented in [5] and [7]. Note that in the type of automaton that is extracted in this paper, i.e., in which the next state is the one that advances the robot the most toward its goals, an action might not be turned on/off simultaneously with the sensor input change. In other words, the change in the action might occur only when the robot enters a new region. This may be avoided by extracting an automaton that does not have to make progress at each step, but this is not within the scope of this paper.

This kind of execution allows the robot to react in real time to changing environment conditions as opposed to the approach taken in [11], where it is assumed that the robot senses only when it makes a transition in the automaton (enters a new region).

Example 3: Fig. 4 depicts a possible execution of the automaton synthesized in Example 2. Here, the robot starts in region 3 and searches for Nemo. In Fig. 4(a), the robot finds Nemo in region 5, and therefore, it stays there and turns on its camera, which is indicated by the magenta squares. Then, as depicted in Fig. 4(b), Nemo disappears, and the robot resumes the search. It then continues to search all the regions of interest, as seen in Fig. 4(c).

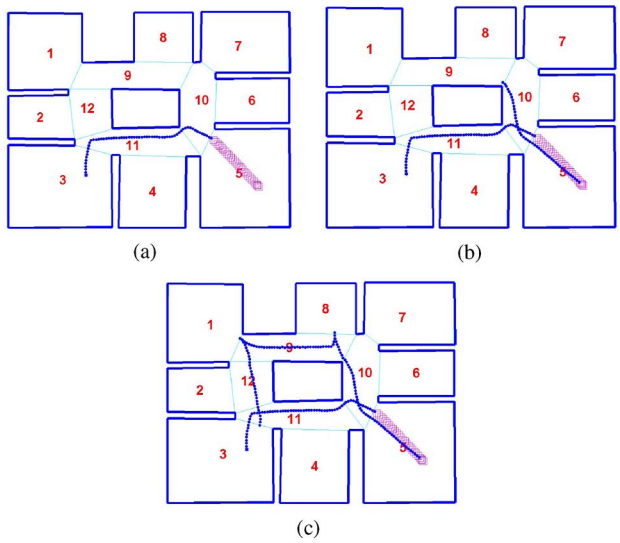

Fig. 4. Possible run of the automaton of Example 2. (a) Nemo is sensed in region 5. (b) Nemo disappeared; therefore, the robot continues the search. (c) Robot continues to search regions 1, 3, 5, and 8.

VI. Guarantees and Generalizations

The method presented in this paper is *guaranteed*, under some conditions, to generate *correct* robot behavior. These conditions can be divided into two groups. The first group refers to tasks, or formulas, that are unrealizable. These formulas cannot be synthesized into an automaton either because they are logically inconsistent, for example, "Go to region 1 and never leave region 4," or they are topologically impossible, for example, a task that requires the robot to move between two unconnected regions in the workspace, or the environment, has a strategy that prevents the robot from achieving its goals.[2] If the formula is unrealizable, the synthesis algorithm will inform the user that it is impossible to satisfy such a task.

The second group refers to tasks that are realizable, i.e., there exists an automaton that realizes them; however, the execution of this automaton is not correct. This can happen only if the environment behaves "badly" or if it either violates the assumptions encoded in φ_e, for example, a sensor proposition becomes true when the formula states it cannot, or it causes the robot to violate its possible transitions, for example, if someone picks the robot up and moves it to a different region. In these cases, it is most likely that the automaton will not have any valid transitions left, and the execution will halt. Note that while the satisfaction of the assumptions on the environment is crucial for correct behavior, these assumptions can be very general and nonrestrictive.

The introduced framework can be extended very easily to handle different robot models. By choosing controllers that are

[2]The synthesis of the controller is done in a worst-case approach.

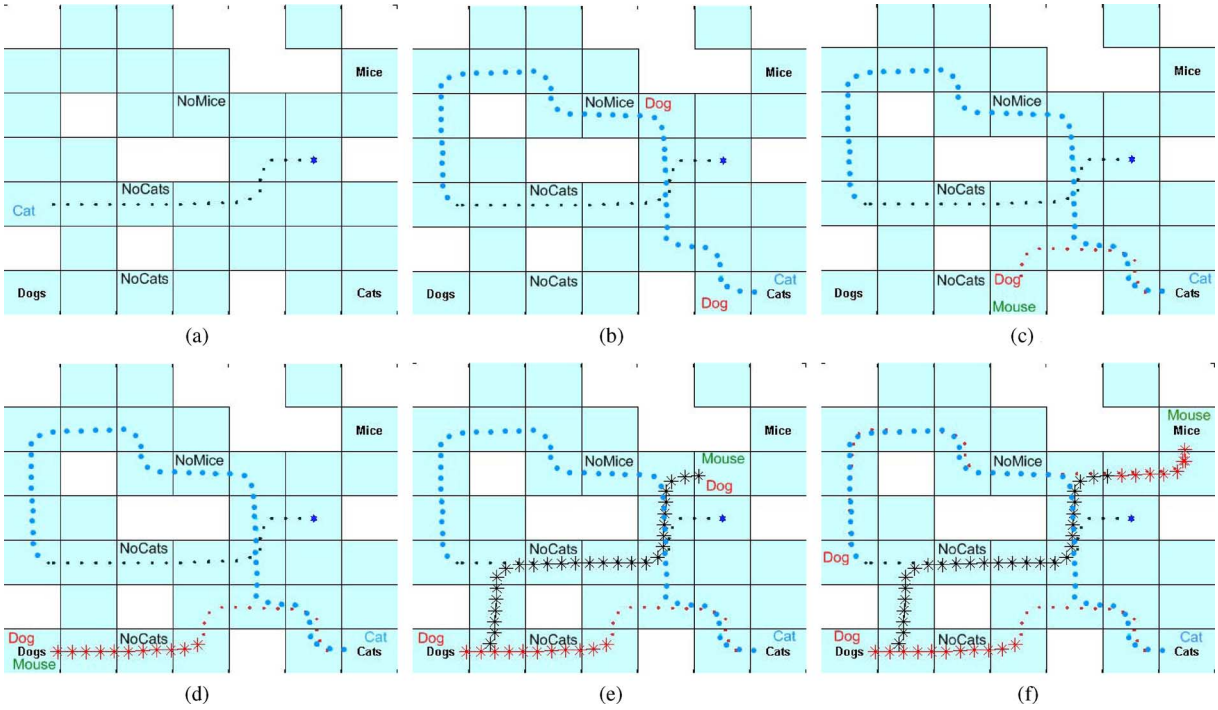

Fig. 5. Animal herding. (a) Cat is found. (b) Robot herds the cat to its meeting region, without going through the NoCats regions, by trailing yarn. It ignores dogs sensed along the way. (c) Dog is sensed immediately after leaving the cats meeting region. It is then herded to the dogs region. A mouse is sensed along the way. (d) Robot herds both the dog and the mouse to the dogs meeting region. (e) Mouse is herded to its meeting region. A dog is sensed. (f) Mouse stays at its meeting region and the dog is being herded.

guaranteed to drive a robot with a more complex model from one region to an adjacent one without going through any other region, we obtain the same guarantees of correctness, for example, we can handle robots with car-like nonholonomic constraints by employing the controllers in [36]. Note that we can only guarantee correctness and completeness if there exists a discrete abstraction of the workspace and a set of bisimilar controllers corresponding to the robot dynamics and the abstraction.

Furthermore, some of the assumptions made in this paper, e.g., the partition of the workspace, can be relaxed. For this approach, we need to encode possible motion of the robot using binary propositions that correspond to the activation of different controllers. Whether such controllers correspond to moving between two regions in a partition of the workspace or moving between overlapping regions does not change the synthesis and execution of the automaton. The only difference would be that one would have to be careful when specifying that areas that may now belong to the domain of several controllers should not be reached. Relaxing the partition requirement allows us to use controllers for convex-bodied nonholonomic robots, such as in [9], as we have done in [38].

VII. Single-Robot Scenario—Animal Herding

One of the strengths of this approach is the ability to generate a complex behavior from a large set of simple instructions or rules. The automatically generated controller is *guaranteed* to obey all given rules, as long as they are consistent, as opposed to the handwritten code that may contain functional, structural, and design-software bugs. Furthermore, if the set of rules is inconsistent, the algorithm will stop without creating the controller, which indicates that the desired rule set cannot be implemented.

This scenario, which is presented as an example for a complex task, includes a robot that is moving in a workspace that has 35 connected regions, as shown in Fig. 5. The robot's goal is to herd dogs, cats, and mice to their respective meeting points, which are denoted in the figure by *Dogs, Cats, and Mice*. In order to do that, the robot can show a bone to the dogs, trail a piece of yarn for the cats, and play a flute for the mice. The animals can appear anywhere in the workspace other than the meeting places, and while an animal is being herded, it might choose not to follow the robot and, thus, may no longer be sensed by the robot. We impose further restrictions on the behavior of the robot, which are as follows.

1) The robot cannot herd a dog and a cat simultaneously.
2) The robot cannot herd a cat and a mouse simultaneously.
3) Unless there are no animals, the robot must herd at least one animal at any given time.
4) The robot cannot herd a cat through "NoCats" regions.
5) The robot cannot herd a mouse through "NoMice."
6) The robot must start herding a dog when it senses one, unless it is already herding a cat.
7) When sensing a dog and a cat, the dog is to be herded.
8) When sensing a cat and a mouse, the cat is to be herded.
9) The robot must start herding a cat when it senses one, unless it is already herding a mouse or it senses a dog.
10) The robot should continue herding an animal until it reaches its meeting region or it disappears and should not switch between animals randomly.

In the original set of rules, restriction 9 stated that "the robot must start herding a cat unless it is already herding a mouse" without any mention of a dog that may be spotted at the same time as the cat. This is inconsistent with requirements 1 and 6, and therefore, no automaton was generated.

In order to encode this behavior, the set of sensor propositions was $\mathcal{X} = \{s^{\text{dog}}, s^{\text{cat}}, s^{\text{mouse}}\}$, and the set of robot propositions was $\mathcal{Y} = \{r_1, \ldots, r_{35}, a^{\text{bone}}, a^{\text{yarn}}, a^{\text{flute}}\}$.[3] For lack of space, we omit the full LTL formula, and only demonstrate how some of the requirements were encoded.

All the aforementioned restriction are encoded as part of φ_t^s, and here, we show restrictions 1, 3, 5, and 6 as

$$\begin{cases} \bigwedge \Box(\neg(\bigcirc a^{\text{bone}} \wedge \bigcirc a^{\text{yarn}})) \\ \bigwedge \Box((\bigcirc s^{\text{dog}} \vee \bigcirc s^{\text{cat}} \vee \bigcirc s^{\text{mouse}}) \\ \quad \Rightarrow (\bigcirc a^{\text{bone}} \vee \bigcirc a^{\text{yarn}} \vee \bigcirc a^{\text{flute}})) \\ \bigwedge \Box(\bigcirc a^{\text{flute}} \Rightarrow \neg \bigcirc r_{20})) \\ \bigwedge \Box((\bigcirc s^{\text{dog}} \wedge (\neg(a^{\text{yarn}} \wedge \bigcirc s^{\text{cat}}))) \Rightarrow \bigcirc a^{\text{bone}}) \end{cases}$$

where region 20 is the "NoMice" region.

The robot must search the space for animals, and once it encounters one, it must herd it to its meeting region. This requirement is encoded as part of φ_g^s, where the search of each region is encoded as

$$\Box\Diamond(r_i \vee s^{\text{dog}} \vee s^{\text{cat}} \vee s^{\text{mouse}})$$

which means that region i should be searched unless an animal is sensed, and the herding is encoded as

$$\Box\Diamond(r_1 \vee \neg a^{\text{bone}}) \bigwedge \Box\Diamond(r_{32} \vee \neg a^{\text{yarn}}) \bigwedge \Box\Diamond(r_{36} \vee \neg a^{\text{flute}})$$

which means that the robot should go to the dogs' (cats' or mice's) meeting region if the robot is showing a bone (trailing yarn or playing the flute), i.e., herding a dog (cat or mouse).

Fig. 5 shows a sample run of the generated automaton. The robot starts by searching the space. It encounters a cat (a) and proceeds to trail yarn, indicated by the large (blue) dots. It herds the cat to its meeting region (b), while ignoring dogs that are present along the way. Furthermore, the robot causes the cat to reach its meeting region without going through the "NoCats" regions. Immediately after leaving the cats' meeting region, the robot encounters a dog and proceeds to herd it to its meeting region by showing it a bone, indicated by small (red) dots. Along the way, it finds a mouse (c) and starts playing the flute as well, which is indicated by the light (red) stars. The robot, now herding both a dog and a mouse, first goes to the dogs' meeting region (d), drops the dog there, and continues, with the mouse, to the mice's meeting region. Along the way, it sees another dog (e), and after dropping the mouse off, it takes the dog to its meeting place (f).

VIII. Multirobot Scenarios

This section illustrates that our framework naturally captures multirobot scenarios where one robot becomes part of the environment of another robot. In a natural *decentralized* model, each robot (or subgroup of robots) is tasked by its own formula φ_i, resulting in its own synthesized automaton. In this case, the coordination between robots (groups) can be done using sensor propositions.

We illustrate this approach with the following example. In the workspace shown in Fig. 2, two robots are placed in regions 1, 2, or 3, independently. The desired behavior is "First, Robot 1 goes looking for Nemo in regions 1, 3, 5, and 8, while Robot 2 stays in place. Once Robot 1 finds him, it stays in place and turns on its camera. Robot 2 then comes and joins them, and turns on its light. If Nemo disappears again, Robot 1 resumes the search with the camera OFF, and Robot 2 stays where it is, but turns OFF its light."

Example 4: We write one formula for each robot, thus creating a separate automaton for each robot. Note that the behavior of Robot 1 is identical to the behavior of the robot in Example 1. Following that example, we define one environment proposition s^{Nemo} and 13 robot propositions $\{r_1, \ldots, r_{12}, a^{\text{CameraON}}\}$, which refers to the robot's location and action. In order to allow communication between the robots (allowing Robot 1 to inform Robot 2 whether Nemo was found and where), we add four more robot propositions: $\{a^{\text{SendNemoIn1}}, a^{\text{SendNemoIn3}}, a^{\text{SendNemoIn5}}, a^{\text{SendNemoIn8}}\}$. These propositions encode the action of transmitting a message saying that Nemo was found in a certain region. Since the motion of Robot 1 and activation of the camera is the same as the behavior of the robot in Example 1, formula φ_1 encoding Robot 1's behavior is φ in this example, with additional subformulas that take care of the new robot propositions.

The requirement that proposition $a^{\text{SendNemoIni}}$ is true, i.e., the robot sends a message that Nemo is in region i, if and only if Robot 1 is in region i and it senses Nemo, is encoded as part of φ_1^s as

$$\varphi_1^s = \begin{cases} \varphi_i^s, \text{ (from Example 1)} \\ \varphi_t^s, \text{ (from Example 1)} \\ \bigwedge \Box((r_1 \wedge \bigcirc s^{\text{Nemo}}) \Leftrightarrow \bigcirc a^{\text{SendNemoIn1}}) \\ \bigwedge \Box((r_3 \wedge \bigcirc s^{\text{Nemo}}) \Leftrightarrow \bigcirc a^{\text{SendNemoIn3}}) \\ \bigwedge \Box((r_5 \wedge \bigcirc s^{\text{Nemo}}) \Leftrightarrow \bigcirc a^{\text{SendNemoIn5}}) \\ \bigwedge \Box((r_8 \wedge \bigcirc s^{\text{Nemo}}) \Leftrightarrow \bigcirc a^{\text{SendNemoIn8}}) \\ \varphi_g^s, \text{ (from Example 1).} \end{cases}$$

For the formula φ_2, which describes the desired behavior of Robot 2, we define four environment propositions $\{s^{\text{NemoIn1}}, s^{\text{NemoIn3}}, s^{\text{NemoIn5}}, s^{\text{NemoIn8}}\}$, which indicate that Nemo was found in regions 1, 3, 5, and 8, respectively. Similar to Example 1, we define 13 robot propositions $\{r_1, \ldots, r_{12}, a^{\text{LightON}}\}$ that refer to the robot's location and action. For φ_2^e, we make the following assumptions about the environment: Initially, Nemo is not sensed; hence, initially, all environment propositions are set to False; furthermore, since Nemo can only be in one place at a time, at most, one environment proposition can be True at any time. These assumptions

[3]When coding this example, the regions are encoded as binary vectors instead of separate proposition. Here, we continue with this notation for clarity.

are encoded in φ_2^e as

$$\varphi_2^e = \begin{cases} (\neg s^{\text{NemoIn1}} \wedge \neg s^{\text{NemoIn3}} \wedge \neg s^{\text{NemoIn5}} \wedge \neg s^{\text{NemoIn8}}) \\ \bigwedge \square((\neg \bigcirc s^{\text{NemoIn1}} \wedge \neg \bigcirc s^{\text{NemoIn3}} \\ \quad \wedge \neg \bigcirc s^{\text{NemoIn5}} \wedge \neg \bigcirc s^{\text{NemoIn8}}) \vee \\ \quad (\bigcirc s^{\text{NemoIn1}} \wedge \neg \bigcirc s^{\text{NemoIn3}} \\ \quad \wedge \neg \bigcirc s^{\text{NemoIn5}} \wedge \neg \bigcirc s^{\text{NemoIn8}}) \vee \\ \quad (\neg \bigcirc s^{\text{NemoIn1}} \wedge \bigcirc s^{\text{NemoIn3}} \\ \quad \wedge \neg \bigcirc s^{\text{NemoIn5}} \wedge \neg \bigcirc s^{\text{NemoIn8}}) \vee \\ \quad (\neg \bigcirc s^{\text{NemoIn1}} \wedge \neg \bigcirc s^{\text{NemoIn3}} \\ \quad \wedge \bigcirc s^{\text{NemoIn5}} \wedge \neg \bigcirc s^{\text{NemoIn8}}) \vee \\ \quad (\neg \bigcirc s^{\text{NemoIn1}} \wedge \neg \bigcirc s^{\text{NemoIn3}} \\ \quad \wedge \neg \bigcirc s^{\text{NemoIn5}} \wedge \bigcirc s^{\text{NemoIn8}})) \\ \bigwedge \square\Diamond(\text{True}). \end{cases}$$

The desired robot behavior is encoded in φ_2^s. Initially, Robot 2 starts somewhere in region 1, 2, or 3 with the light OFF, and hence

$$\varphi_{i_2}^s = \begin{cases} (r_1 \wedge_{i\in\{2,\cdots,12\}} \neg r_i \wedge \neg a^{\text{LightON}}) \\ \bigvee (r_2 \wedge_{i\in\{1,3,\cdots,12\}} \neg r_i \wedge \neg a^{\text{LightON}}) \\ \bigvee (r_3 \wedge_{i\in\{1,2,4,\cdots,12\}} \neg r_i \wedge \neg a^{\text{LightON}}). \end{cases}$$

The first two lines of $\varphi_{t_2}^s$ describe the transitions and mutual exclusion, as discussed before. Line 3 states that whenever Robot 2 is in region $i \in \{1, 3, 5, 8\}$ and Nemo is sensed there, it should stay where it is. Line 4 forces Robot 2 to have the light ON whenever it is region i and Nemo is sensed there. Line 5 requires the light to be OFFwhenever Robot 2 is not in the same region as Nemo. The last subformula requires Robot 2 to stay where it is whenever Nemo is not sensed by Robot 1.

$$\varphi_{t_2}^s = \begin{cases} \text{Possible Transitions Between Regions} \\ \bigwedge \text{Mutual Exclusion of Regions} \\ \bigwedge_{i\in\{1,3,5,8\}} \square((r_i \wedge \bigcirc s^{\text{NemoIni}}) \Rightarrow \bigcirc r_i) \\ \bigwedge_{i\in\{1,3,5,8\}} \square((\bigcirc r_i \wedge \bigcirc s^{\text{NemoIni}}) \Rightarrow \bigcirc a^{\text{LightON}}) \\ \bigwedge \square((\neg(\vee_{i\in\{1,3,5,8\}}(\bigcirc r_i \wedge \bigcirc s^{\text{NemoIni}}))) \Rightarrow \\ \quad \neg \bigcirc a^{\text{LightON}}) \\ \bigwedge \square(\ (\wedge_{i\in\{1,3,5,8\}} \neg \bigcirc s^{\text{NemoIni}}) \Rightarrow \\ \quad (\wedge_{i\in\{1,\ldots,12\}}(r_i \Leftrightarrow \bigcirc r_i))). \end{cases}$$

The final part of the formula $\varphi_{g_2}^s$ requires Robot 2 to visit region $i \in \{1, 3, 5, 8\}$ infinitely often if Nemo is sensed in that region

$$\varphi_{g_2}^s = \bigwedge_{i\in\{1,3,5,8\}} \square\Diamond(s^{\text{NemoIni}} \Rightarrow r_i).$$

The synthesized automaton for Robot 1 has the same number of states and the same transitions as the automaton in Example 2. The only difference is that in this example, the robot propositions $a^{\text{SendNemoIn5}}$, $i \in \{1, 3, 5, 8\}$ are added as labels to the relevant states. The synthesized automata for Robot 2 satisfying φ_2 contains 55 states and is omitted here.

A possible execution of these automata is depicted in Fig. 6, where the robots start in regions 3 and 2, respectively. Robot 1 begins by searching for Nemo, while Robot 2 stays in region 2, as seen in Fig. 6(a). In Fig. 6(b), Robot 1 finds Nemo in region 5 and, therefore, stays there and turns on its camera, which is indicated by the magenta squares. Finding Nemo in region 5 causes Robot 2 to move to that region as well. When Robot 2 arrives in region 5, as depicted in Fig. 6(c), it turns on its light, which is indicated by the black triangles. Finally, in Fig. 6(d), Nemo disappears. This causes Robot 1 to turn camera OFF and resume the search, while Robot 2 turns off the light and stays in place.

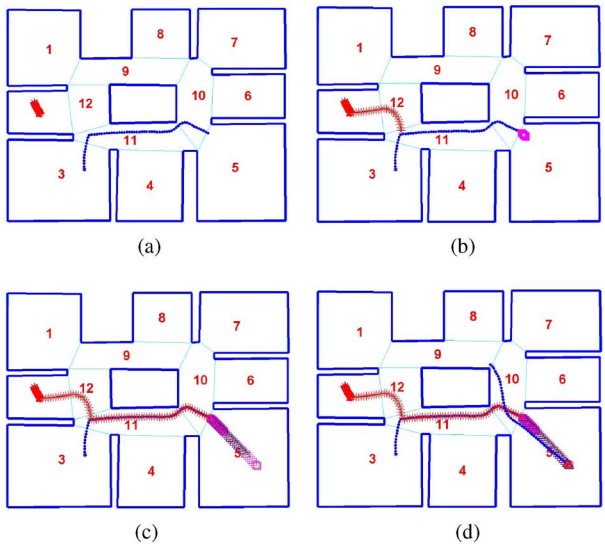

Fig. 6. Possible run of the multirobot scenario of Example 4. (a) Robot 1 is searching for Nemo, and Robot 2 stays in region 2. (b) Robot 1 finds Nemo in region 5, and Robot 2 goes to join him. (c) Robot 2 joins Robot 1 in region 5 and turns the light ON. (d) Nemo disappears, and therefore, Robot 1 resumes the search, and Robot 2 stays where it is.

When considering multirobot behaviors, here, we assume that there are no timing or concurrency constraints, such as two robots that must reach a room at the exact same time. Such constraints are difficult to guarantee in a purely decentralized framework and might require a different approach.

IX. Discussion

In this paper, we have described a method of creating controllers that drive a robot, or a group of robots, such that it satisfies a high-level user-specified behavior. These behaviors are expressed in a subset of LTL, and can capture *reactive* tasks in which the robot's behavior depends on the local information sensed from the environment during runtime. These behaviors are guaranteed to be satisfied by the robots if the environment in which they are operating behaves "well," i.e., it adheres to the assumptions made about it. We have shown that many complex robot behaviors can be expressed and computed, both for a single robot and for multiple robots.

Writing LTL formulas, especially ones that conform to the structure presented in Section III-B, requires some experience and is not always intuitive. Therefore, we plan to explore more user-friendly interfaces, as well as linguistic

formalisms, that can be easily translated to the required logic representation.

Another issue that we wish to address is that of computability. As mentioned in Section IV, the synthesis algorithm is polynomial in the state space, but this space may be exponential in the number of inputs and outputs. Currently, we can generate an explicit representation of an automaton that has 50 000 states in about an hour (an automaton with a few hundred states takes a few seconds to compute) on a regular desktop; however, by moving to a symbolic representation and examining hierarchical approaches, we believe that we can tackle problems of a larger scale.

Finally, we intend to experiment with different controllers and various robots in order to gain a good intuition regarding the variety of tasks and the interplay between the robot dynamics, its behavior, and the appropriate discrete abstractions.

ACKNOWLEDGMENT

The authors would like to thank D. Conner, for sharing with the code for the potential field controllers, and N. Piterman, A. Pnueli, and Y. Sa'ar for fruitful discussions and for sharing their code for the synthesis algorithm.

REFERENCES

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

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

[3] S. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[4] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion: State of the art and grand challenges," *Robot. Autom. Mag.*, vol. 14, no. 1, pp. 61–70, 2007.

[5] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, Barcelona, Spain, 2005, pp. 2020–2025.

[6] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proc. 44th IEEE Conf. Decis. Control*, Dec. 2005, pp. 4885–4890.

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

[8] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Las Vegas, NV, Oct. 2003, pp. 3546–3551.

[9] D. Conner, H. Choset, and A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies," presented at the Robot.: Sci. Syst., Cambridge, MA, Jun. 2006.

[10] S. Lindemann and S. LaValle, "Computing smooth feedback plans over cylindrical algebraic decompositions," presented at the Robot.: Sci. Syst., Cambridge, MA, Jun. 2006.

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

[12] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*, Cambridge, MA: MIT Press, 1990, pp. 995–1072

[13] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Proc. VMCAI*, Jan. 2006, pp. 364–380.

[14] R. W. Brockett, "On the computer control of movement," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 1998, vol. 1, pp. 534–540.

[15] V. Manikonda, P. S. Krishnaprasad, and J. Hendler, *Languages, Behaviors, Hybrid Architectures, and Motion Control*. New York: Springer-Verlag, 1999.

[16] D. Hristu-Varsakelis, M. Egerstedt, and P. S. Krishnaprasad, "On the structural complexity of the motion description language MDLe," in *Proc. 42nd IEEE Conf. Decis. Control*, Dec. 2003, vol. 4, pp. 3360–3365.

[17] F. Delmotte, T. R. Mehta, and M. Egerstedt, "Modebox a software tool for obtaining hybrid control strategies from data," *Robot. Autom. Mag.*, vol. 15, no. 1, pp. 87–95, 2008.

[18] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1077–1091, Dec. 2005.

[19] S. B. Andersson and D. Hristu, "Symbolic feedback control for navigation," *IEEE Trans. Autom. Control*, vol. 51, no. 6, pp. 926–937, Jun. 2006.

[20] C. Belta and L. C. G. J. M. Habets, "Constructing decidable hybrid systems with velocity bounds," in *Proc. IEEE Conf. Decis. Control*, 2004, pp. 467–472.

[21] S. R. Lindemann and S. M. LaValle, "Smoothly blending vector fields for global robot navigation," in *Proc. IEEE Conf. Decis. Control*, Seville, Spain, 2005, pp. 207–214.

[22] S. Ramamoorthy and B. Kuipers. (2004). Controller synthesis using qualitative models and simulation, in *Proc. International Workshop on Qualitative Reasoning (QR-2004)*, J. de Kleer and K. Forbus, Eds. [Online]. Available: http://www.qrg.cs.northwestern.edu/qr04/papers.html.

[23] G. Pola, A. Girard, and P. Tabuada, "Approximately bisimilar symbolic models for nonlinear control systems," *Automatica*, vol. 44, no. 10, pp. 2508–2516, 2008.

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

[25] T. Moor and J. M. Davoren, "Robust controller synthesis for hybrid systems using modal logic," in *Hybrid Systems: Computation and Control*, (Lecture Notes in Computer Science 2034). New York: Springer-Verlag, 2001, pp. 433–446

[26] J. M. Davoren and T. Moor, "Logic-based design and synthesis of controllers for hybrid systems," Dept. Syst. Eng, Aust. Nat. Univ., Canberra, A.C.T., Australia, Tech. Rep., 2000.

[27] L. de Alfaro, T. A. Henzinger, and R. Majumdar, "From verification to control: Dynamic programs for omega-regular objectives," in *Proc. 16th Annu. Symp. Logic Comput. Sci.*, Los Alamitos, CA: IEEE Comput. Soc. Press, Jun. 2001, pp. 279–290.

[28] P. Bouyer, L. Bozzelli, and F. Chevalier, "Controller synthesis for MTL specifications," in *CONCUR* (Lecture Notes in Computer Science 4137), C. Baier and H. Hermanns, Eds. New York: Springer-Verlag, 2006, pp. 450–464.

[29] P. Tabuada and G. J. Pappas, "Linear time logic control of discrete-time linear systems," *IEEE Trans. Autom. Control*, vol. 51, no. 12, pp. 1862–1877, Dec. 2006.

[30] 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, 2009.

[31] R. Milner, *Communication and Concurrency.* Upper Saddle River, NJ: Prentice-Hall, 1989.

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

[33] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. 16th ACM SIGPLAN-SIGACT Symp. Principles Program. Languages*, Toronto, ON, Canada: ACM Press, 1989, pp. 179–190

[34] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 971–984, Jul. 2000.

[35] L. C. G. J. M. Habets and J. H. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, no. 1, pp. 21–35, 2004.

[36] S. R. Lindemann and S. M. LaValle, "Smooth feedback for car-like vehicles in polygonal environments," in *Proc. IEEE Conf. Robot. Autom.*, 2007, pp. 3104–3109.

[37] E. Rimon and D. E. Kodischek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Autom.*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[38] D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas, "Valet parking without a valet," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, San Diego, CA, Oct. 2007, pp. 572–577.

Hadas Kress-Gazit (S'06–M'09) received the Ph.D.degree in electrical and systems engineering from the University of Pennsylvania, Philadelphia, in 2008.

She is currently an Assistant Professor with Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY. Her current research interests include creating verifiable robot controllers for complex high-level tasks using logic, verification methods, synthesis, hybrid systems theory, and computational linguistics.

Dr. Kress-Gazit was a finalist for the Best Student Paper Award at the 2007 IEEE International Conference on Robotics and Automation and a finalist for the 2007 Best Paper Award at the International Conference on Intelligent Robots and Systems.

Georgios E. Fainekos (S'04–M'08) received the Ph.D. degree in computer and information science from the University of Pennsylvania, Philadelphia, in 2008.

He was a Postdoctoral Researcher with the the System Analysis and Verification Group, NEC Laboratories America, Inc., where he was engaged in cyber-physical systems. Since August 2009, he has been an Assistant Professor with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe. His current research interests include formal methods and logic, control theory and hybrid, and embedded and real-time systems with applications to robotics and unmanned aerial vehicles.

Dr. Fainekos was a finalist for the Best Student Paper Award at the 2007 IEEE International Conference on Robotics and Automation and a recipient of the 2008 Frank Anger Memorial Association for Computing Machinery Special Interest Group on Embedded Systems/Special Interest Group on Software Engineering Student Award.

George J. Pappas (S'90–M'91–SM'04–F'09) received the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 1998.

He was the former Director of the General Robotics, Automation, Sensing and Perception (GRASP) Laboratory, University of Pennsylvania, Philadelphia, where he is currently a Member of GRASP, the Deputy Dean with the School of Engineering and Applied Science and is the Joseph Moore Professor with the Department of Electrical and Systems Engineering. He is also with the Department of Computer and Information Sciences and the Department of Mechanical Engineering and Applied Mechanics. His current research interests include hybrid and embedded systems, hierarchical control systems, distributed control systems, nonlinear control systems, and geometric control theory, with applications to robotics, unmanned aerial vehicles, and biomolecular networks.

Prof. Pappas has received numerous awards, including the National Science Foundation (NSF) CAREER Award in 2002, the NSF Presidential Early Career Award for Scientists and Engineers in 2002, and the Eliahu Jury Award for Excellence in Systems Research from the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in 1999.