2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Where's Waldo? Sensor-Based Temporal Logic Motion Planning *

Hadas Kress-Gazit, Georgios E. Fainekos and George J. Pappas
GRASP Laboratory, University of Pennsylvania
Philadelphia, PA 19104, USA
{hadaskg,fainekos,pappasg}@grasp.upenn.edu

Abstract— **Given a robot model and a class of admissible environments, this paper provides a framework for automatically and verifiably composing controllers that satisfy high level task specifications expressed in suitable temporal logics. The desired task specifications can express 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, resulting 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 multi-robot specifications. Our computational approach is based on first creating discrete controllers satisfying so-called General Reactivity(1) formulas. If feasible, the discrete controller is then used in order to guide the sensor-based composition of continuous controllers resulting in a hybrid controller satisfying the high level specification, but only if the environment is admissible.**

Index Terms— **Motion planning, temporal logics, sensor-based planning, controller synthesis, hybrid control.**

I. Introduction

Motion planning and task planning are two fundamental problems in robotics that have been addressed from differen perspectives. Bottom-up motion planning techniques concentrate on creating control inputs or closed loop controllers for detailed robot models that steer it from one configuration to another [1], [2]. Such controllers can either assume perfect knowledge of the environment [3] or receive information about the environment through the use of sensors [4]. On the other hand, top-down task planning approaches are usually focused on finding coarse, typically discrete, robot actions in order to achieve more complex tasks [5]. Such goals may include final goals for multiple robots [6] or temporal ordering or sequencing of goals [7].

The natural hierarchical decomposition of task planning layers residing higher 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, coupling continuous and discrete systems, has enabled the formal integration of high level discrete actions with low level controllers in a unified framework. 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 [8]–[10] or compose local controllers in order to construct global plans [11]–[13].

This paper follows the spirit of our previous work [8], [9] where complex task specifications are expressed as linear temporal logic formulas [14]. However, this paper contributes in two very significant and novel directions. The first novelty is that the temporal logic we consider explicitly models sensor inputs. This enables our task descriptions to depend on possibly dynamic environment, capturing multi-robot search and rescue style missions. In a multi-robot setting, one robot is part of the environment of another robot, hence it is very natural to consider a variety of other multi-robot missions as well. The interpretation or execution of such tasks has a very natural game-theoretic flavor between the robot and the environment (or other robots). Depending on the the environment, the execution of the task may be different, but it will satisfy the task if the environment is admissible.

The second novelty in this paper is the use of a very recent fragment of linear temporal logic which is called General Reactivity (1) (GR(1)) [15]. By restricting to GR(1) formulas, the complexity of translating a formula to an automaton becomes polynomial (from double exponential in the size of the formula). This dramatic acceleration in computation does not come at a major expense of expressivity, as a large number of (but not all) tasks specified in practice is naturally captured in the fragment of interest.

As in [8], [9], the solution of the discrete synthesis algorithm is integrated with the controllers in [11] resulting in an overall hybrid controller that is orchestrating the composition of low level controllers based on the sensorial interaction with the environment. The overall closed loop system is guaranteed, by construction, to satisfy the desired specification but only if the robot operates in an environment that satisfies whatever assumptions that were explicitly modeled, as another temporal logic formula, in the synthesis process. This leads to a very natural assume-guarantee decomposition between the robot and its environment.

II. Problem Formulation

The goal of this paper is to construct controllers for mobile robots that generate continuous trajectories satisfying given specifications. Furthermore, we would like to achieve such specifications while interacting, using sensors, with a variety of environments. To achieve this, we need to specify

*This work is partially supported by National Science Foundation EHS 0311123, National Science Foundation ITR 0324977, and Army Research Office MURI DAAD 19-02-01-0383.

a *robot model*, assumptions on *admissible environments*, and the desired *system specification*.

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

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

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 creates boolean propositions $\mathcal{Y} = \{r_1, r_2 \ldots, r_n\}$ which are true if the robot is located in P_i, for example r_1 is true iff $p \in P_1$. Since $\{P_i\}$ is a partition of P, exactly one r_i can be true at any time.

Admissible environments: The robot interacts with its environment using sensors, which in this paper are assumed to be binary. The m binary sensor variables $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ have their own (discrete) dynamics which we do not model explicitly. Instead, we place high level assumptions on the possible behavior of the sensor variables, defining a class of admissible environments. These environmental assumptions will be captured (in Section III) by a suitable 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.

System Specification: The desired system specification for the robot will be expressed as a suitable formula φ_s in the so-called linear temporal logic (LTL) [14]. Informally, LTL will be used (in Section III) to specify a variety of robot tasks that are linguistically expressed as:

- Coverage: "Go to rooms P_1,P_2,P_3,P_4 in any order".
- Sequencing: "First go to room P_5, then to room P_2".
- Conditions: "If you see Mika, go to room P_3, otherwise stay where you are".
- Avoidance: "Don't go to corridor P_7 ".

Furthermore, LTL is compositional, enabling 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 robot model (1), initial position $p(0)$, and suitable temporal logic formula φ_e modeling our assumptions on admissible environments, construct (if possible) a controller so that the robot's resulting trajectories $p(t)$ satisfy the system specification φ_s in any admissible environment.
The approach presented in the paper can be easily generalized to the case where the initial position of the robot is not specified, but may belong in a number of cells.

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

Loosely speaking, Linear Temporal Logic (LTL) [14] consists of the standard propositional logic with some temporal operators that allow us to express requirements on sequences of propositions.

A. LTL Syntax and Semantics

Syntax: Let AP be a set of atomic propositions. In our setting $AP = \mathcal{X} \cup \mathcal{Y}$, including both sensor and system 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, 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 on an infinite sequence σ of truth assignments to the atomic propositions $\pi \in AP$. For a formal definition of the semantics we refer the reader to [14]. Informally, 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 $\Box\varphi$ 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 φ is true infinitely often.

B. Special class of LTL formulas

Following [15], we consider a special class of temporal logic formulas. We first recall that we have divided our atomic propositions into sensor propositions $\mathcal{X} = \{x_1, \ldots, x_m\}$, and system propositions $\mathcal{Y} = \{r_1, \ldots, r_n\}$.

These special formulas are LTL formulas of the form $\varphi = (\varphi_e \Rightarrow \varphi_s)$. 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 system. Both φ_e and φ_s have the following structure

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

where

- φ_i^e, φ_i^s - Non-temporal boolean formulas constraining (if at all) the initial value(s) for the sensor propositions $\mathcal{X}$ and system propositions $\mathcal{Y}$ respectively.
- φ_t^e - represents the possible evolution of the state of the environment. It consists of a conjunction of formulas of the form $\Box B_i$ where each B_i is a boolean formula constructed from subformulas in $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X}$, 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 system $\mathcal{Y}$ values.
- φ_t^s - represents the possible evolution of the state of the system. It consists of a conjunction of formulas of

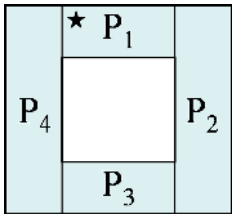


Fig. 1: The workspace of Example 1. The initial position of the robot is marked with a star.

the form $\square B_i$ where each B_i is a boolean formula in $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X} \bigcirc \mathcal{Y}$.

- φ_g^e, φ_g^s - represent goal assumptions for the environment and desired goal specifications for the system. Both formulas consist of a conjunction of formulas of the form $\square\diamond B_i$ where each B_i is a boolean formula.

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.

Despite the structural restrictions of this class of LTL formulas, there does not seem to be a significant loss in expressivity. 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. 1 consisting of four areas labelled $P_1, ..., P_4$ (which define the system propositions $\mathcal{Y} = \{r_1, \ldots, r_4\}$). Initially, the robot is placed somewhere in region P_1. In natural language, the desired specification for the robot is: *Look for Waldo in regions P_2 and P_4, if you find him, stay where you are, and if not, keep looking.*

Since Waldo is part of the environment, we consider one sensor proposition $\mathcal{X} = \{s^{\text{Waldo}}\}$ which becomes true if our sensor has detected Waldo. Our assumptions about Waldo are captured by $\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$. The robot initially does not see Waldo, thus $\varphi_i^e = (\neg s^{\text{Waldo}})$. Since we can only sense Waldo in regions P_2 and P_4, we encode the requirement that in other regions the value of s^{Waldo} cannot change. Furthermore, we assume (for simplicity) that once the robot detects Waldo, Waldo doesn't move. These requirements are captured by the formula

$$\varphi_t^e = \begin{cases} \square((\neg r_2 \wedge \neg r_4) \Rightarrow (\bigcirc s^{\text{Waldo}} \Leftrightarrow s^{\text{Waldo}})) \\ \bigwedge \square(s^{\text{Waldo}} \Rightarrow \bigcirc s^{\text{Waldo}}) \end{cases}$$

We place no further assumptions on the environment propositions which means that $\varphi_g^e = \square\diamond True$, completing the modeling of our environment assumptions. Notice that the environment is admissible whether Waldo is there or not.

We now turn to modeling the robot and the desired specification, captured by $\varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$. Initially, the robot starts somewhere in region r_1, hence $\varphi_i^s = (r_1 \wedge \neg r_2 \wedge \neg r_3 \wedge \neg r_4)$. φ_t^s models the possible changes in in the robot state. The first four subformulas represent the possible *transitions* between regions, for example, from region P_1 the robot can move to adjacent regions P_2, P_4, or stay in P_1. The next four subformulas capture the *mutual exclusion* constraint, that is at any step, exactly one of r_1, r_2, r_3, and r_4 is true. For a given decomposition of workspace P, the generation of these formulas is easily automated. The final subformula is part of the desired specification and states that if the robot is in region P_2 (or P_4) and it sees Waldo when he senses[1] it should remain in region P_2 (respectively P_4) in the next step as well.

$$\varphi_t^s = \begin{cases} \bigwedge \square(r_1 \Rightarrow (\bigcirc r_1 \vee \bigcirc r_2 \vee \bigcirc r_4)) \\ \bigwedge \square(r_2 \Rightarrow (\bigcirc r_1 \vee \bigcirc r_2 \vee \bigcirc r_3)) \\ \bigwedge \square(r_3 \Rightarrow (\bigcirc r_2 \vee \bigcirc r_3 \vee \bigcirc r_4)) \\ \bigwedge \square(r_4 \Rightarrow (\bigcirc r_1 \vee \bigcirc r_3 \vee \bigcirc r_4)) \\ \bigwedge \square((\bigcirc r_1 \wedge \neg \bigcirc r_2 \wedge \neg \bigcirc r_3 \wedge \neg \bigcirc r_4) \\ \quad \vee (\neg \bigcirc r_1 \wedge \bigcirc r_2 \wedge \neg \bigcirc r_3 \wedge \neg \bigcirc r_4) \\ \quad \vee (\neg \bigcirc r_1 \wedge \neg \bigcirc r_2 \wedge \bigcirc r_3 \wedge \neg \bigcirc r_4) \\ \quad \vee (\neg \bigcirc r_1 \wedge \neg \bigcirc r_2 \wedge \neg \bigcirc r_3 \wedge \bigcirc r_4) \ ) \\ \bigwedge_{i \in \{2,4\}} \square((r_i \wedge \bigcirc s^{\text{Waldo}}) \Rightarrow \bigcirc r_i \ ) \end{cases}$$

Finally, the requirement that the robot keeps looking in regions P_2, P_4 unless it has found Waldo is captured by

$$\varphi_g^s = \square\diamond(r_2 \vee s^{\text{Waldo}}) \bigwedge \square\diamond(r_4 \vee s^{\text{Waldo}})$$

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 modelled 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). This is the goal of the next two sections.

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 [16]. However, by restricting ourselves to the special class of LTL formulas, we can use the efficient algorithm recently introduced in [15] which is polynomial O(n^3) time, where n is the number of valuations of the sensor and state variables. We present the algorithm informally, and refer the reader to [15] for a full description.

The synthesis process is viewed as a game played between the system (robot) and the environment (as the adversary). Starting from some initial state, both the robot and the environment make transition to the state of the system. The winning condition for the game is given as a GR(1) formula ϕ. The way the game is played is that at each step, first the environment makes a transition according to its transition relation and then the system makes its own transition. If the system can satisfy ϕ no matter what the environment does, we say that the system is winning and we can extract an automaton for our robot. 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 above, the initial states of the players are given by φ_i^e

[1]As explained in Section IV, at each step the robot first senses the environment and then moves, therefore we need to refer to the truth value of $\bigcirc s^{\text{Waldo}}$

and φ_i^s. The possible transitions 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 system 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 system. 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 [15] takes the GR(1) formula φ and first checks whether it is realizable. If it is, 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 behaviour. The automaton that is generated by the algorithm can be modeled as a tuple $A = (\mathcal{X}, \mathcal{Y}, Q, q_0, \delta, \gamma)$ where:

- $\mathcal{X}$ is the set of input (environment) propositions
- $\mathcal{Y}$ is the set of output (system) propositions
- $Q \subset \mathbb{N}$ is the set of states
- $q_0 \in Q$ is the initial state
- $\delta : \ Q \times 2^{\mathcal{X}} \rightarrow 2^Q$ 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.
- $\gamma : \ Q \rightarrow 2^{\mathcal{Y}}$ is the state labeling function where $\gamma(q) = y$ and $y \in 2^{\mathcal{Y}}$ is the set of state propositions that are true in state q. Note that in our case, since the only outputs are the regions, and there is only one output proposition that is true at every state, $\gamma(q) = y \in \mathcal{Y}$.

Note that this automaton can be nondeterministic[2]. An admissible input sequence is a sequence $X_1, X_2, ...\ , X_j \in 2^{\mathcal{X}}$ that satisfies φ_e. A run of this automaton under an admissible input sequence is a sequence of states $\sigma = q_0, q_1, ...$ This sequence starts at the initial state and follows the transition relation δ and the truth values of the input propositions, i.e. for all $j \geq 0$, $q_{j+1} \in \delta(q_j, X_j)$. An interpretation of a run σ is a sequence $y_0, y_1, ...$ where $y_i = \gamma(q_i)$ is the label of the i^{th} state in the run. We use this sequence of labels to construct the discrete path the robot must follow. As mentioned before, when given a non-admissible 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.

Example 2: Revisiting Example 1, Fig. 2 represents the synthesized automaton that realizes the desired behavior. The number at the top of each circle is the state and the proposition that is written inside each circle is the state's label, i.e. the output proposition that is true in that state. We can see that the robot will first search P_2 and then, if it doesn't find Waldo, continue to search P_4. If Waldo is nowhere to be found, the robot will continue to look for him forever. Note that this plan is not unique, since the robot could have started searching in P_4. Furthermore, it is also nondeterministic since the robot can go from state 2 to state 6 through either state 3 or 4.

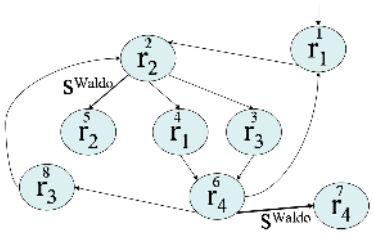


Fig. 2: The synthesized automaton of Example 2

From the interpretation of a run of the automaton, we extract a discrete path for the robot. What is left to do, is to transform this discrete path to a continuous trajectory, as is explained in the next section.

V. Controller Composition

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.

Initially, the robot is placed in region i_0 such that $\gamma(q_0) = r_{i_0}$. During the execution, at step $j \geq 1$ the robot first senses its environment[3] and determines X_j. Then the next automaton state is selected $q_j \in \delta(q_{j-1}, X_j)$ and the next region i_j the robot must go to is extracted by $r_{i_j} = \gamma(q_j)$. When the robot reaches region i_j, step $j + 1$ is performed. By continuing this procedure, the discrete path $r_{i_0}, r_{i_1}, ...$ is extracted, and by combining the simple controllers, the continuous path is achieved.

Following the work in [8], we utilize atomic controllers that satisfy the so-called bisimulation property [17]. 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 [11], [18].We use the framework developed in [11] due to its computational properties and the variety of regions it can be applied to. In this approach, the control input is the gradient of a harmonic potential function.

We would like to emphasize that this method can employ different and more realistic types of controllers, dealing with convex bodied robots and nonholonomic constraints [19], as long as they satisfy the bisimulation property.

VI. Case Studies

In this section we give several examples of desired behaviors, the automata that implement them and the trajectories which they induce. The polygonal environment we use for the examples is shown in Fig. 3. In the following we refer to region P_i as region i.

[2]By making a small change in the algorithm, the automaton may become deterministic, i.e. for every input there will be a unique next state

[3]An implicit assumption is that the sensing is performed only when entering a region. Another approach would be to check the sensor values every computation cycle and allow the controller to change before exiting the current region.

A. Single robot - Nursery scenario

The desired behaviour is: "Starting in region 1, keep checking whether a baby is crying in regions 2 or 4. If you find a crying baby, go look for an adult in regions 6, 7 and 8. Keep looking until you find him. After finding the adult, go back to monitoring the babies and so on..."

We can define two environment propositions here, one indicating a crying baby was sensed and another indicating an adult was found. In order to reduce the number of variables, the computation time and the size of the automaton, we use one environment proposition, $CkBby$, indicating whether the robot should check on the babies (when the proposition is true) or go look for an adult (when the proposition is false). Initially $CkBby$ is true. We assume that the proposition becomes false in regions 2 and 4 if the robot senses a baby crying and once it becomes false it stays false as long as it is in 2 or 4 (a baby does not stop crying on her own and she cannot be ignored). Furthermore, we assume that $CkBby$ becomes true in regions 6, 7 and 8 only if the robot sensed an adult. Once it becomes true it stays true in these regions (once the adult was found, the robot must return to check on the babies). In all other regions, the truth value of the proposition may not change.

Following these assumptions, we can construct φ_e:

$$\varphi_e = \begin{cases} CkBby \\ \bigwedge \square(((r_2 \vee r_4) \wedge \neg CkBby) \rightarrow (\neg \bigcirc CkBby))) \\ \bigwedge \square(((r_6 \vee r_7 \vee r_8) \wedge CkBby) \rightarrow (\bigcirc CkBby))) \\ \bigwedge \square(\neg(r_2 \vee r_4 \vee r_6 \vee r_7 \vee r_8) \\ \qquad \rightarrow (\bigcirc CkBby \leftrightarrow CkBby)) \\ \bigwedge \square\diamond True \end{cases}$$

As for the robot, we have ten system propositions $r_1, ..., r_{10}$, one for each region. Constructing φ_s:

$$\varphi_s = \begin{cases} r_1 \wedge_{i=2,...,10} \neg r_i \\ \bigwedge Transitions \bigwedge Mutual\ Exclusion \\ \bigwedge_{i\in\{2,4\}} \square\diamond(r_i \vee \neg CkBby) \\ \bigwedge_{i\in\{6,7,8\}} \square\diamond(r_i \vee CkBby) \end{cases}$$

The first and second lines encode the initial condition, possible transitions and mutual exclusion requirement as in Example 1. The rest of the formula describes the desired behaviour, for example, the third line requires the robot to infinitely often either visit region $i,\ i \in \{2,4\}$ or look for an adult.

Running this example through the synthesis algorithm, the computation time was 2 seconds and we got an automaton with 41 states that realizes this specification. Sample simulations are shown in Fig. 4.

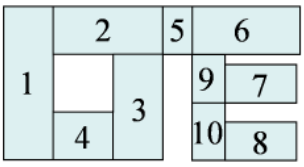
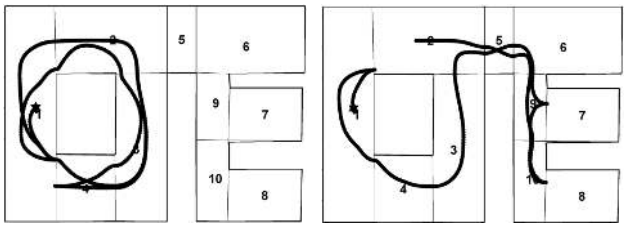


Fig. 3: The environment used in section VI.



(a) Babies are not crying (b) A baby in region 4 cries and the adult is in region 8

Fig. 4: Nursery Example

B. Multi robot - Search and Rescue

Our framework captures very naturally multi-robot scenarios where one robot becomes part of the environment of another robot. In a natural *decentralized* model, each robot is tasked by it's own formula φ_i resulting in it's own synthesized automaton. The coordination between robots can be done using the input (sensor) propositions, as shown in the following scenario.

In this search and rescue scenario, we employ two UAV's that continuously search regions 1, 3, 7 and 8 for injured people. Once an injured person was found, a ground vehicle (ambulance) goes to the person's location and helps out. If there are no reports of people needing help, the ground vehicle does not move. If the ground vehicle is in any of the search regions, the UAV's may skip it. We assume, for simplicity, that the two UAV's fly at different altitudes so there can be no collisions between the agents.

The two UAV's will be named robot 1 and 2 and initially they are in regions 4 and 6 respectively. Other than the initial region, the two formulas φ_1, φ_2 will be the same therefore we describe φ_1 only. Since the behavior of these robots depend only on the location of the ground vehicle (denoted as robot 3), we define four environment propositions $r_i^3,\ i \in \{1,3,7,8\}$ indicating whether robot 3 is in either of these regions.

$$\varphi_1^e = \begin{cases} \bigwedge \neg r_1^3 \bigwedge \neg r_3^3 \bigwedge \neg r_7^3 \bigwedge \neg r_8^3 \\ \bigwedge Mutual\ Exclusion\ between\ r_i^3,\ i \in \{1,3,7,8\} \\ \bigwedge \square\diamond True \end{cases}$$

Robot 3 does not start in regions 1, 3, 7 or 8, and it cannot be in two regions at the same time.

$$\varphi_1^s = \begin{cases} r_4^1 \wedge_{i=1,2,3,5,...,10} \neg r_i^1 \\ \bigwedge Transitions \bigwedge Mutual\ Exclusion \\ \bigwedge_{i\in\{1,3,7,8\}} \square\diamond(r_i^1 \vee r_i^3) \end{cases}$$

The robot has to infinitely often visit region i, unless robot 3 is there. This formula took 11 seconds to realize and the automaton has 129 states.

Robot 3 (the ground vehicle) is initially in region 10. The behavior of robot 3 depends on the sensing done by robots 1 and 2 that is transmitted to it. For φ_3 we define four input propositions: $help_i,\ i \in \{1,3,7,8\}$ indicating people needing help in the respective regions. To make the automaton smaller, we assume that once the robot reaches region i, the proposition $help_i$ becomes false, and if $help_j$

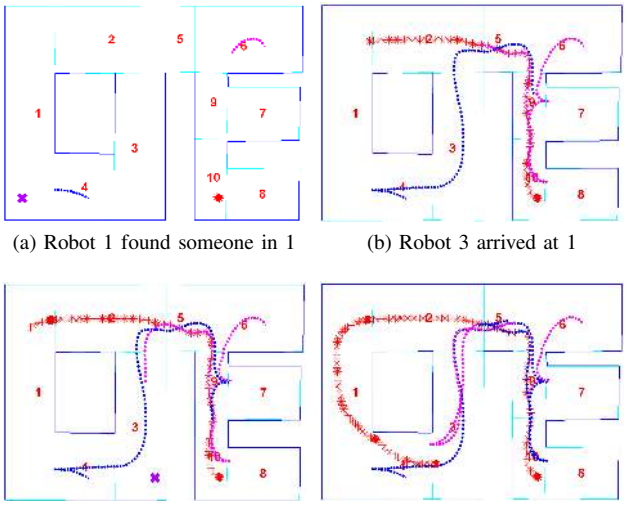
is true, it stays true until the robot reaches region j.

$$\varphi_3^e = \begin{cases} \wedge_{i\in\{1,3,7,8\}} \neg help_i \\ \bigwedge_{i\in\{1,3,7,8\}} \Box(r_i^3 \rightarrow \neg \bigcirc help_i) \\ \bigwedge_{i\in\{1,3,7,8\}} \Box((\neg r_i^3 \wedge help_i) \rightarrow \bigcirc help_i) \\ \bigwedge \Box\Diamond True \end{cases}$$

Robot 3 stays in place unless there is a need for help.

$$\varphi_3^s = \begin{cases} r_{10}^1 \ \wedge_{i=1,\ldots,9} \ \neg r_i^1 \\ \bigwedge \ Transitions \ \bigwedge \ Mutual\ Exclusion \\ \bigwedge \Box((\wedge_{i\in\{1,3,7,8\}} \neg \bigcirc help_i) \\ \qquad \Rightarrow (\wedge_{j\in\{1,\ldots,10\}} \bigcirc r_j^3 \leftrightarrow r_j^3)) \\ \bigwedge_{i\in\{1,3,7,8\}} \Box\Diamond(r_i^3 \vee \neg help_i) \end{cases}$$

This formula took 60 seconds to realize and the automaton has 282 states. Fig. 5 depicts four snapshots of a sample simulation. In this simulation, robot 1 detects a person (indicated by an X) in region 1, causing robot 3 to move to region 1. Then, later on, robot 2 detects a person in region 3 and subsequently, robot 3 moves to region 3.



(a) Robot 1 found someone in 1 (b) Robot 3 arrived at 1

(c) Robot 2 found someone in 3 (d) Robot 3 arrived at 3

Fig. 5: Search and Rescue

VII. Conclusions - Future work

In this paper, we have described a method of creating controllers which are guaranteed to satisfy a user specified behavior expressed in temporal logic. Furthermore, these controllers behave in a reactive manner, i.e. the behavior of the robot can depend on the local information it senses from the environment in which it is operating. 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 requires some experience, and might lead to unintended behaviors. Therefore, we plan to examine how natural language can be automatically translated into logic, thus enabling "non-expert" users to take advantage of this method. Furthermore, we would like to create some feedback to the user that will help him figure out what went wrong if the specification is unrealizable. Another direction we are working on is experimenting with different controllers and various robots, simulated and real.

Acknowledgements

We would like to thank David Conner for allowing us to use his code for the potential field controllers. We would also like to thank Nir Piterman, Amir Pnueli and Yaniv Sa'ar for allowing us to use 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*. MIT Press, Boston, USA, 2005.

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

[3] Elon Rimon and Daniel E. Kodischek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, October 1992.

[4] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *The International Journal of Robotics Research*, 19:96–125, February 2000.

[5] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, second edition, 2003.

[6] R.M. Jensen and M. M. Veloso. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, 13:189–226, 2000.

[7] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, , and P. Traverso. MBP : A model based planner. In *In Proc. IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[8] Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 2020–2025, 2005.

[9] Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *IEEE Conference on Decision and Control*, Seville, Spain, 2005.

[10] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from ltl specifications. In *9th International Workshop on Hybrid Systems: Computation and Control*, Santa Barbara, California, 2006.

[11] David C. Conner, Alfred A. Rizzi, and Howie Choset. Composition of Local Potential Functions for Global Robot Control and Navigation. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 3546 – 3551, Las Vegas, NV, October 2003.

[12] D. Conner, H. Choset, and A. Rizzi. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.

[13] S. Lindemann and S. LaValle. Computing smooth feedback plans over cylindrical algebraic decompositions. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.

[14] E. Allen Emerson. Temporal and modal logic. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.

[15] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380, Charleston, SC, Jenuary 2006.

[16] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190. ACM Press, 1989.

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

[18] Calin Belta and L.C.G.J.M. Habets. Constructing decidable hybrid systems with velocity bounds. In *IEEE Conference on Decision and Control*, Bahamas, 2004.

[19] David C. Conner, Howie Choset, and Alfred Rizzi. Towards provable navigation and control of nonholonomically constrained convex-bodied systems. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA '06)*, May 2006.