Correct High-level Robot Behavior in Environments with Unexpected Events

Kai Weng Wong*, Rüdiger Ehlers† and Hadas Kress-Gazit*
*Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, New York, USA
†University of Bremen and DFKI GmbH, Bremen, Germany
Emails: kw358@cornell.edu, ruediger.ehlers@uni-bremen.de, hadaskg@cornell.edu

***Abstract*—Synthesis of correct-by-construction robot controllers from high-level specifications has the advantage of providing guaranteed robot behavior under different environments. Typically, when such controllers are synthesized, assumptions that the user makes about the behavior of the environment, if any, are incorporated into the resulting controller. In practice, however, the environment assumptions may be unknown to the user, thus preventing the application of synthesis. Even if environment assumptions are available, they may not hold during the robot's execution due to modeling errors or unforeseen anomalous operating conditions.**

In this paper, we address both of these problems. We present an approach for synthesizing controllers that include built-in recovery transitions, enabling the robot to make progress towards its goals in the event of environment assumption violation, whenever possible. Furthermore, we present a process for automatically augmenting a specification with environment assumptions that are computed from the robot's observations at runtime. We start with a set of candidate assumptions that is updated whenever violated at runtime.

I. Introduction

Recent work on synthesizing robot controllers from high-level specifications (e.g. [2], [3], [9], [12], [13], [15], [25]) advocates for applying the approach to create correct-by-construction controllers. In this context, the specification is usually expressed using temporal logic, and the synthesized controllers, which are finite-state machines, are guaranteed to cause the robot to fulfill its tasks. When considering *reactive* tasks in which the robot's behavior depends on what the environment does during runtime, the synthesized controller is able to react to every possible environment behavior in a way that is compliant with the specification.

When users write specifications, they may make assumptions about the environment behavior. For example:

Example 1: Consider Specification 1 with the workspace in Fig. 1 in which the robot is asked to patrol *mailroom* and *office* (lines 3,4) while making sure that it is not in *door* when the door is closed (line 2). In addition, the user made an assumption that the door is never closed (line 1). The dark gray regions in Fig. 1 are obstacles.

Specification 1 Specification for Example 1

	Components of φ_e^t		Components of φ_s^g
1	$\square(\neg\pi_{doorClosed})$	3	$\square\diamondsuit(\pi_{mailroom})$
	Components of φ_s^t	4	$\square\diamondsuit(\pi_{office})$
2	$\square(\pi_{doorClosed} \rightarrow \neg\pi_{door})$		

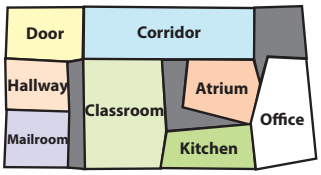

Fig. 1: Workspace for Example 1 and 2

Without the assumption on the behavior of the door as in line 1 of Specification 1, the desired robot behavior cannot be guaranteed, since the environment controlling the door could keep the door closed indefinitely. In this case here with the assumption in line 1, Specification 1 is realizable and a correct-by-construction controller can be synthesized.

In previous work [15], [25], the synthesized controllers were allowed to crucially rely on the environment assumptions to hold. Thus, these controllers typically do not have a suitable course of action if the environment assumptions are violated. Yet, assumption violations can occur in practice due to improper modeling or lack of prior information. In Example 1, if at runtime the door closes, the user's assumption about the environment is violated and the robot will not know what to do, i.e. the controller will not have a next state.

In this paper, we approach the problem of ensuring correct robot behavior in environments with unexpected events or unknown environments using the following *recovery* and *environment characterization* schemes. Our proposed *recovery* scheme modifies the strategy extraction process of the synthesis procedure to synthesize an automaton that contains (i) the correct robot actions when the environmental assumptions are satisfied (as in [4], [15], [25]) and (ii) robot actions, if any exist, that preserve the robot's safety requirements and make sure that the robot can make progress towards its goals when the environment resumes the expected behavior. In particular, we add transitions during the synthesis process that are used as fall-back transitions when environment safety assumptions, i.e., assumptions about the environment's admissible transitions, are violated.

Our *environment characterization* scheme is inspired by previous work [16], [17] in the runtime verification community. We monitor the behavior of the environment and check for violations of the environment safety assumptions during execution. Every time a violation is detected, we note the violation and continue to make progress towards the robot's goals with the recovery scheme if possible; otherwise,

we reconstruct the environment safety assumption based on the observed environment behavior and synthesize a new controller. This way, we tailor the environment assumptions to the actual environment behavior but avoid an expensive resynthesis step whenever it is not crucially necessary. If the resulting assumptions render the specification unsynthesizable, this implies that some environment liveness assumption is still missing. We then employ specification analysis tools (e.g. [14], [22], [23]) and provide feedback to the user. Based on the feedback that pinpoints the group of formulae responsible for the specification being unsynthesizable, the user can add an additional environment liveness assumption. An advantage of the environment characterization scheme is that it can be applied to scenarios in which a reasonable set of environment assumptions is completely unknown by simply starting with an unsatisfiable safety assumption (i.e., $\varphi_e^t = \square(\mathbf{false})$. See Def. 2.3.).

This paper proposes an approach to creating provably correct task executions for robots that operate in environments that display unexpected/unmodeled behaviors. Instead of silently failing, the robots attempt to recover from sporadic environment safety assumption violations when possible, and relax these assumptions based on the observed behavior, when not possible. If relaxing the assumptions leads to unrealizable specifications, the robot alerts the user and allows them to modify the assumptions. The work in this paper is novel in that it (i) allows the robot to make progress towards its goals, when it can, even if the assumptions are violated and (ii) it automatically redefines the assumed environment behavior that the robot must react to, based on the robot observation.

The outline of this paper is as follows: After discussing related work, we define the necessary preliminaries in Section II. In Section III, we motivate and explain the problem being solved in this paper and in Sections IV, V and VI we describe our approach. Section VII illustrates and discusses the performance of our approach using an example.

Related work: *Topology change and motion planning:* [19], [20] consider a similar problem of having the robot's workspace topology further constrained during execution. To address this change, they propose online syntheses of local strategies at the affected workspace area and patch these local controllers onto the original robot controller. [11] revises and verifies a motion plan based on real-time information while keeping the original specification unchanged in partially known workspaces. In this paper, we address the occurrence of unexpected environment behavior at runtime; this is not related to the workspace topology, which is part of the system guarantees and not the environment assumptions.

Unrealizable specification: Work by Li et al. [18] generates realizable specifications from unrealizable ones through the analysis of counter-strategies returned by the synthesis procedure and mines environment assumptions with LTL formula templates. Their approach, which is an offline process, returns realizable specifications but those may not capture the actual behavior of the environment as observed at runtime.

Learning: [10] uses grammatical inference to iteratively refine symbolic controllers during execution in an unknown adversarial environment. [5] uses automata learning methods to develop a provably correct control strategy for a robot moving in an environment with unknown dynamics. In our work, we do not learn an exact/probabilistic model of the environment but rather we relax assumptions based on observations. This way, we only resynthesize and change the assumptions when needed and we do not need to collect data for quantitative learning of transition probabilities.

Response to disturbances: [21] defines metric automata and strategies on which it guarantees that the system behavior remains close to the nominal-case behavior under the influence of unmodeled disturbances. In our work we either have guaranteed correct behavior or we resynthesize a new "nominal" behavior if one exists. We do not consider closeness metrics. [24] generates a robust control strategy for an open finite transition system with modeling uncertainties. Users are required to provide uncertainty models, in the form of transitions in addition to those in the original finite transition system, in order to compute control strategies. Our approach does not require user input to deal with the violations other than of liveness assumptions when the specification becomes unsynthesizable. In [1] the authors synthesize controllers from a Continuous Stochastic Logic (CSL) specification that aim to complete as many tasks as possible during a given time constraint with a-priori knowledge about the probabilistic nature of the environment and its changing topology. Our work considers a different problem where we require (non probabilistic) guarantees for robots operating in environments with unmodeled behaviors.

Tolerating assumption violations: [7] proposes a qualitative error-resilient system synthesis approach in which the system is allowed to temporarily violate its guarantees in case of environment assumption violations. In our work, we ensure that no system guarantees is ever violated. [8] describes an approach to synthesize a controller that can tolerate short sequences of up to k environment assumption violations for some k. The approach comes at a high computational cost, and does not include an additional best-effort approach to tolerate more assumption violations whenever possible, as our recovery approach does. However, the technique is orthogonal to ours in the sense that they can be combined to yield both benefits.

II. Preliminaries

***Definition** 2.1:* **(Linear Temporal Logic (LTL))** Linear Temporal Logic formulas are defined recursively with a set of atomic proposition AP as:

$$\varphi ::= \pi \in AP \mid \neg\varphi \mid \varphi \vee \varphi' \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi'$$

The conjunction ($\wedge$), implication ($\rightarrow$) and biimplication ($\leftrightarrow$) operators can be derived from the disjunction ($\vee$) and negation ($\neg$) operators. The "always " ($\square$) and "finally" ($\diamondsuit$) operators can be derived from the "until" ($\mathcal{U}$) and "next" ($\bigcirc$) operators.

The truth of an LTL formula is evaluated over infinite sequences of states σ of a finite state machine $\mathcal{A}$ (Definition 2.5), with each state defined as a truth assignment to the set of

propositions AP. The statement $\sigma \models \bigcirc \varphi$ holds if φ is true at the next state in the sequence; $\sigma \models \square \varphi$ holds if φ is true at all states in the sequence; $\sigma \models \diamondsuit \varphi$ holds if φ is true at some state in the sequence. $\mathcal{A}$ is said to satisfy a formula φ if for every execution σ of $\mathcal{A}$, $\sigma \models \varphi$. A complete and formal definition of the semantics of LTL can be found in [6].

***Definition** 2.2:* **(Atomic Propositions)** The continuous behavior of the robot and the environment are abstracted by a finite set of atomic propositions, $AP = \mathcal{X} \cup \mathcal{Y}$.

- $\mathcal{X}$ is the set of environment inputs result from abstracting the behavior of the environment sensed by the robot's physical sensors into a set of boolean propositions.
- $\mathcal{Y} = Reg \cup Act$ is the set of robot outputs. The set of region propositions Reg is obtained by partitioning the workspace of the robot. There is always exactly one r_i with the value $true$, indicating the robot is inside a region r_i. The set of robot action propositions Act is abstracted from robot actions, e.g., grasping a letter. a_i is true if the robot is performing the action and false otherwise.

***Definition** 2.3:* **(Mission Specification)** We consider an LTL specification φ of the form:

$$\varphi = \varphi_e \rightarrow \varphi_s \tag{1}$$

We denote $\mathcal{X} \cup \mathcal{Y}$ and $\mathcal{X}' \cup \mathcal{Y}'$ as the atomic propositions in the current state and the next state respectively. With $\alpha \in [e, s]$, each φ_α is a conjunction of the following formulas:

- φ_α^i specify the initial conditions of the environment and the system.
- φ_α^t consist of the system safety guarantees (φ_s^t) and the environment safety assumptions (φ_e^t). φ_s^t are of the form $\bigwedge_{k \in K} \square A_k$ with A_k being boolean formulae defined over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}' \cup \mathcal{Y}'$. φ_e^t are of the form $\bigwedge_{j \in J} \square B_j$ with B_j being boolean formulae defined over $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{X}'$. $\mathcal{X}'$ and $\mathcal{Y}'$ are equivalent to their corresponding variables in $\mathcal{X}$ and $\mathcal{Y}$ when the latter are prefixed by $\bigcirc$ operators.
- φ_α^g consist of the system liveness guarantees (φ_s^g) and the environment liveness assumptions (φ_e^g). They are of the form $\bigwedge_{i \in I} \square \diamondsuit \phi_i^\alpha$ with ϕ_i^α being boolean formulae defined over $\mathcal{X} \cup \mathcal{Y}$. The ϕ_i^g components are also called system goals.

The topology of the workspace is encoded in the specification φ as part of φ_s^t. Here we omit the details of that encoding.

A specification is said to be unsatisfiable if φ_s cannot be satisfied no matter what the environment behavior is. A specification is said to be unrealizable if the environment can enforce a violation of φ. No controller $\mathcal{A}$ (see Def 2.5) is generated if the specification is unsatisfiable or unrealizable. A specification is unsynthesizable if it is either unsatisfiable or unrealizable; otherwise, it is synthesizable, or realizable.

***Definition** 2.4:* **(Game Structures)** In order to determine if there exists a controller (Def. 2.5) that satisfies some mission specification φ (Def. 2.3), the generalized reactivity(1), or GR(1) synthesis approach from [4] is used, which reduces the problem to finding a winning strategy in a two-player game between a system (robot) and an environment. The game structure is defined as a tuple $\mathcal{G} = (\mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \phi)$ with the following components:

- $\mathcal{X}$ and $\mathcal{Y}$ are environment input variables and system (robot) output variables, as described in Def. 2.2. For notational convenience, we also declare $V = \mathcal{X} \cup \mathcal{Y}$. We say that some assignment to the variables in V is a *position* of the game.
- $\Theta \subseteq 2^V$ is the set of initial positions of $\mathcal{G}$.
- $\rho_e \subseteq 2^V \times 2^{\mathcal{X}}$ is the set of input constraints that restricts the behavior of the environment at some positions.
- $\rho_s \subseteq 2^V \times 2^V$ is the transition relation of $\mathcal{G}$ that describes the allowed transitions by the system.
- ϕ is a winning condition for the system in $\mathcal{G}$, given as an LTL formula of the form $(\square \diamondsuit \phi_1^e \wedge \ldots \wedge \square \diamondsuit \phi_m^e) \rightarrow (\square \diamondsuit \phi_1^s \wedge \ldots \wedge \square \diamondsuit \phi_n^s)$, where every element of $\phi_1^e, \ldots, \phi_n^s$ is a boolean formula over V that is free of temporal operators.

By solving the game, we can obtain a winning strategy for the system (robot) player. This is elaborated in Section V.

***Definition** 2.5:* **(Synthesis and Execution of the Controller)** Our synthesized controller is a deterministic finite state automaton $\mathcal{A} = (\mathcal{X}, \mathcal{Y}, S, S_0, \delta_s, \gamma_\mathcal{X}, \gamma_\mathcal{Y})$ constructed based on the framework presented in [15], where:

- $\mathcal{X}$ and $\mathcal{Y}$ are defined in Definition 2.2,
- S is the set of states with $S_0 \subseteq S$ being the set of initial states,
- $\delta_s : S \times 2^\mathcal{X} \rightharpoonup S$ is the (partial) transition function that defines the next state for a current state and a subset of the environment inputs $\mathcal{X}$ that are true.
- $\gamma_\mathcal{X} : S \rightarrow 2^\mathcal{X}$ labels a state s_i by input valuations, which correspond to all transitions $x_i \subseteq \mathcal{X}$ leading to the state.
- $\gamma_\mathcal{Y} : S \rightarrow 2^\mathcal{Y}$ is the state labeling function that labels each state s_i with a subset of the propositions in $y_i \subseteq \mathcal{Y}$ that are true in that state.

The infinite execution of automaton $\mathcal{A}$ is a sequence of states $\sigma = s_0, s_1, \ldots$ such that $s_0 \in S_0$ and $s_{i+1} = \delta_s(s_i, x_{i+1})$ for all $j \in \mathbb{N}$. We define $(\gamma_\mathcal{X}(s_0), \gamma_\mathcal{Y}(s_0))\,(\gamma_\mathcal{X}(s_1), \gamma_\mathcal{Y}(s_1)) \ldots$ to be the trace induced by σ.

III. Main Problem Formulation

This paper addresses the challenge of dealing with specifications that are synthesizable only with the addition of environment assumptions, such as the necessity of line 1 in Example 1. In many situations, a user defining a high-level robot task may have partial or no information about the expected environment behavior. Nonetheless, assumptions about the environment are necessary in order to synthesize a controller. Not knowing what to assume, the user may make unrealistic assumptions that are violated at runtime. In previous work, such violations caused the robot to fail its mission.

***Problem** 1:* Let $\varphi = \varphi_e \rightarrow \varphi_s$ be a realizable mission specification, and φ_e' be an unknown LTL formula describing the actual environment constraints such that φ_e' is of the form described in Definition 2.3 and $\varphi_e \rightarrow \varphi_e'$, i.e. the actual

environment can exhibit a richer set of behaviors. We want to control a robot in an environment satisfying φ_e' to enforce φ_s whenever possible, and obtain an *observed* approximation φ_e'' with $\varphi_e \rightarrow \varphi_e''$. If $\varphi_e' \rightarrow \varphi_s$ is unrealizable, we want to eventually detect that. If φ_e' is only concerned with variables in $\mathcal{X}$, we additionally want the approximation to be conservative, i.e., such that $\varphi_e'' \rightarrow \varphi_e'$ holds.

Note that in Problem 1, we can always set $\varphi_e = \square(\mathbf{false})$ in order to guarantee that we start with a realizable mission specification in the case that no conservative starting point for the actual environment assumptions is available.

IV. Approach

To address Problem 1, we consider a combined approach; first, we create a controller that is less sensitive to unexpected behavior of the environment. We modify the controller synthesis algorithm of [4] so that it creates a controller with extra "recovery" transitions (Section V).

Second, we capture the actual behavior of the environment during execution and rewrite the assumptions portion of the specification, in what we call "Environment Characterization" (Section VI). Since our synthesized controller has recovery transitions, it can continue to work in many cases where rewriting is found to be necessary during runtime. This allows the controller to continue its operation and collect more data for the environment characterization. Note that in the characterization process, we are not modeling the environment explicitly, but rather we are relaxing the assumptions about its behavior and thus synthesizing a plan for a richer set of environment behavior.

Fig. 2 illustrates how our approach affects the flow from high-level mission specification to mission execution of low-level robot behavior.

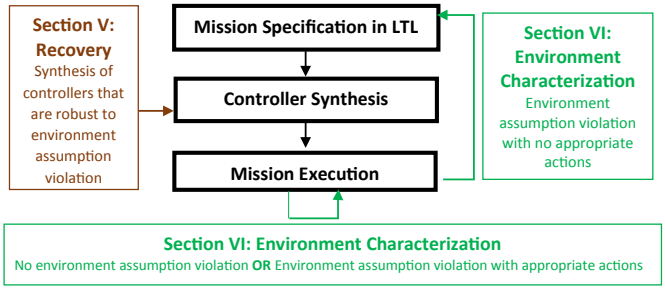

Fig. 2: Approach

V. Recovery

For the first part of our approach, we propose creating robot controllers that (i) preserve the correctness guarantee of the GR(1) synthesis algorithm in [4] when no environment assumption violations occur, and (ii) always choose some next action for the robot, if it exists, that does not violate the system safety guarantees while ensuring that after recovery, the system can still satisfy its mission specification when ignoring the last environment assumption violation.

The algorithm in [4] performs synthesis in two steps: it first solves the game that we build from the specification. Then, if the system is found to be winning, the algorithm uses the artifacts computed in the game solving process to extract a winning strategy for the system player. Our proposed approach for incorporating fault recovery modifies the strategy extraction step of the construction, and in fact *extends* the strategy computed by the approach in [4].

By the results of [4], we can compute the set of winning positions for the system player in $\mathcal{G}$ by the following formula:

$$W = \nu L. \bigwedge_{j=1}^{n} \mu M. \bigvee_{i=1}^{m} \nu N.(\phi_j^s \wedge \textcircled{\scriptsize$\exists$}(L) \vee \textcircled{\scriptsize$\exists$}(M) \vee \neg\phi_i^e \wedge \textcircled{\scriptsize$\exists$}(N))$$

In this formula, ν is a greatest fixed point operator, μ is a least fixed point operator, and $\textcircled{\scriptsize$\exists$}$ is the enforceable predecessor operator. Intuitively, the formula represents that we search for the largest set of positions W from which the system can guarantee to satisfy the winning condition ϕ without leaving W. Inside the outer-most fixed point operator, we iterate over all goals $\phi_1^s, \ldots, \phi_n^s$ and check that the system can reach every goal if the environment liveness assumptions ϕ_i^e hold. The system might not be able to reach the next goal in one step. To find multi-step strategies towards the next goal, we start with the goal positions; we use the least fixed point operator μ to successively compute larger and larger sets of positions in the game from which the system can reach the goal. As the environment may prevent the system from reaching a goal by violating its assumptions, we must account for this fact. For this, we build a greatest fixed point νN over the positions from which we either get closer to the goal, or stay in a set of positions that witnesses the non-satisfaction of the environment liveness assumptions.

When the initial positions are contained in the set of winning positions, the specification is synthesizable. In such cases, we extract a strategy in the form of a finite-state automaton $\mathcal{A}$ from the intermediate computations. In particular, during the last iteration of the outermost greatest fixed point, where we have already computed $W = \nu L. \ldots$, we store for every $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$, $c \in \mathbb{N}$ the following series of intermediate results:

$$P_{j,c,i} = \nu N.(\phi_j^s \wedge \textcircled{\scriptsize$\exists$}(L) \vee \textcircled{\scriptsize$\exists$}(M_{j,c}) \vee \neg\phi_i^e \wedge \textcircled{\scriptsize$\exists$}(N)),$$

where we define:

$$M_{j,c} = \mu^c M. \bigvee_{i=1}^{m} \nu N.(\phi_j^s \wedge \textcircled{\scriptsize$\exists$}(L) \vee \textcircled{\scriptsize$\exists$}(M) \vee \neg\phi_i^e \wedge \textcircled{\scriptsize$\exists$}(N))$$

In this context, the variable j represent the robot goal under concern, c denotes the *reactive distance* to the goal, and i represents some environment liveness condition number. The expression μ^c denotes stopping the least fixed point computation of the formula right of the operator after c steps and returning the result.

We construct a finite-state machine that has a set of states for every goal. When the implementation is in one of the states for a goal j, the finite state machine chooses the next outputs such that the system moves closer to goal ϕ_j^s. We also label every state in the finite state machine with the current input $X \subseteq \mathcal{X}$ and current output $Y \subseteq \mathcal{Y}$ of the system. When the environment provides some next input X' that fulfills the assumptions, i.e., we have $(X, Y, X') \models \rho_e$, the system chooses some next output Y' such that $(X, Y, X', Y') \models \rho_s$

and $(X', Y') \in P_{j,c,i}$. For the lexicographically minimal values of c and i, we give priority to minimizing c. In this way, we enforce that the robot controller moves closer to the goal. Whenever $(X', Y') \models \phi_j^s$, we switch to the next goal.

To incorporate the idea of recovery into this setting, we remove the check whether $(X, Y, X') \models \rho_e$ holds before computing a successor position for some input X'. Instead of looking at which next input is *allowed*, we consider *all* next input possibilities and compute transitions in the finite-state automaton that are *non-failure-causing*. From a state that is labeled by (X, Y), we call some transition for input X' and output Y' non-failure-causing if we have $(X, Y, X', Y') \models \rho_s$ and $(X', Y') \in W$, i.e., starting from position (X', Y') in the game, the system can still satisfy its winning condition. The idea here is that we check if the violation of a safety assumption requires the system to violate the specification in the long run. Whenever this is not the case, there is some recovery output for the system that we can use as Y'. Again, we choose a successor position in $P_{j,c,i}$ with the lexicographically minimal index (c, i). This ensures that environment safety assumption violations that are not of relevance for the robot's current goal are simply ignored by the robot controller that we synthesize. The overall algorithm to obtain an error-resilient implementation after computing $\{P_{j,c,i}\}_{1 \leq j \leq n, c < cmax_j, i < imax_{j,c}}$ is described in Algorithm 1.

Algorithm 1 Strategy extraction algorithm. The game structure is given as a tuple $\mathcal{G} = (\mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \phi)$, the family of results $\{P_{j,c,i}\}_{1 \leq j \leq n, c < cmax_j, i < imax_{j,c}}$ as defined on page 4 is assumed to be computed already. We assume, w.l.o.g., that $n \geq 1$. The variable *ToDo* stores the states whose successors still have to be computed as a set, so duplicates are removed.

procedure EXTRACTIMPLEMENTATION
  $S_0 \leftarrow \Theta \times \{1\}$, $ToDo \leftarrow S_0$, $S \leftarrow \emptyset$, $\delta_s = \emptyset$
  while $ToDo \neq \emptyset$ **do**
    $(s, j) \leftarrow \mathsf{popElement}(ToDo)$
5: $S \leftarrow S \cup \{(s, j)\}$
    $\gamma_{\mathcal{Y}} \leftarrow \gamma_{\mathcal{Y}} \cup \{(s, j) \mapsto s|_{\mathcal{Y}}\}$
    for $X' \subseteq \mathcal{X}$ **do**
      for $c < cmax_j, i < imax_{j,c}$ **do**
        if $\exists Y' \subseteq \mathcal{Y} : (X', Y') \in P_{j,c,i}$
                $\wedge\, (s, X', Y') \models \rho_s$ **then**
10: $Y' \leftarrow$ some $Y' \subseteq \mathcal{Y}$ s.t. $(X', Y') \in P_{j,c,i}$
                $\wedge\, (s, X', Y') \models \rho_s$
          if $(X', Y') \models \phi_j^s$ **then**
            $j' \leftarrow (j \mod n) + 1$
          else
            $j' \leftarrow j$
15: $\delta_s \leftarrow \delta_s \cup \{((s, j), X') \mapsto ((X', Y'), j')\}$
          if $((X', Y'), j') \notin S$ **then**
            $ToDo \leftarrow ToDo \cup \{((X', Y'), j')\}$
          break to line 7
  return $(\mathcal{X}, \mathcal{Y}, S, S_0, \delta_s, \gamma_{\mathcal{Y}})$

The following theorem captures the error-recovery guarantees in the robot controllers synthesized with our approach.

Theorem 1: Let $\varphi = (\varphi_e^i \wedge \varphi_e^t \wedge \varphi_e^g) \rightarrow (\varphi_s^i \wedge \varphi_s^t \wedge \varphi_s^g)$ be a mission specification with $\varphi_e^t = \square \varphi_e^{t,1} \wedge \ldots \wedge \square \varphi_e^{t,k}$. We obtain a finite-state automaton $\mathcal{A}$ that satisfies φ using Algorithm 1 and claim the following quality guarantees of $\mathcal{A}$:

1) Let $w = (w_0^e, w_0^s)(w_1^e, w_1^s)(w_2^e, w_2^s) \ldots \in (2^{\mathcal{X}} \times 2^{\mathcal{Y}})^\omega$ be an infinite trace induced by $\mathcal{A}$ and only for finitely many $j \in \mathbb{N}$, we have $(w_j^e, w_j^s)(w_{j+1}^e, w_{j+1}^s) \not\models \varphi_e^{t,i}$ for some $1 \leq i \leq k$. Then $w \models (\varphi_s^i \wedge \varphi_s^t \wedge \varphi_s^g)$ if $w \models \varphi_e^g$.
2) Let $w = (w_0^e, w_0^s)(w_1^e, w_1^s)(w_2^e, w_2^s) \ldots (w_l^e, w_l^s) \in (2^{\mathcal{X}} \times 2^{\mathcal{Y}})^*$ be a finite trace induced by $\mathcal{A}$; $\sigma = s_0 s_1 \ldots s_l$ be its corresponding run, and $X' \subseteq \mathcal{X}'$ such that $\delta_s(s_l, X')$ is undefined (i.e., the run ends after l transitions). Then we have $(w_l^e, w_l^s)X' \not\models \varphi_e^{t,i}$ for some $1 \leq i \leq k$. Also, there does not exist another automaton $\mathcal{A}'$ that (a) satisfies φ along all of its runs; (b) has a trace of the form $(w_0^e, w_0^s)(w_1^e, w_1^s)(w_2^e, w_2^s) \ldots (w_l^e, w_l^s)(X, w_{l+1}^s) \ldots$ for some w_{l+1}^s, and (c) also has quality guarantee number one from this list.

Proof: For the first claim, note that $w \models \varphi_s^i$ by the fact that φ_s^i is only concerned with the first element of w. Also, by the construction of Θ in $\mathcal{G}$ (see [4]), the automaton can only pick a first output such that it satisfies φ_s^i. We have that $w \models \varphi_s^t$ as Algorithm 1 never picks outputs that violate φ_s^t (which is encoded into ρ_s when building the game from φ). The satisfaction of $\varphi_e^g \rightarrow \varphi_s^g$ follows from the fact that the transitions taken after the last environment assumption violation are also possible transitions in an automaton computed using the classical strategy extraction algorithm for GR(1) specifications, which ensures the satisfaction of the robot goals.

For the second claim, note that the automaton $\mathcal{A}$ computed by Algorithm 1 never transitions to a state whose corresponding position in $\mathcal{G}$ is not winning for the system player. This can be seen from the fact that for all j, $P_{j,c,i}$ converges to the set of winning positions with increasing c and i, and no transition to a state labeled by a position that is not in $P_{j,c,i}$ for some c is possible in $\mathcal{A}$. As for winning $\mathcal{G}$, the system player always has admissible next moves for all environment player moves allowed by ρ_e. The only way for the strategy extraction algorithm not to find a transition is that ρ_e has just been violated. As ρ_e is computed from φ_e^t, this means that some $\varphi_e^{t,i}$ has been violated. However, whenever there is some next move that leads to a winning position and such a move is allowed by φ_s^t, our algorithm finds the position, as it appears in some set $P_{j,c,i}$. On the other hand, transitioning to a non-winning position in the game would prevent the system from satisfying $(\varphi_s^i \wedge \varphi_s^t \wedge \varphi_s^g)$ even if $w \models \varphi_e^t \wedge \varphi_e^g$ in the future; the first property in the claim would be violated. ■

Intuitively, the theorem guarantees two properties of the synthesized robot controller:

- Whenever the trace of the controller is infinite and there are only finitely many temporary violations of the environment safety assumptions, the robot controller will eventually meet its goals while performing only safe actions. Thus, the violations are tolerated.

- If the robot controller gets into a deadlock at some point, then this can only be the case where an environment safety assumption violation has just been witnessed. There does not exist a way to tolerate the violation without preventing the robot from achieving its mission in the future, even in the optimum case that no further environment safety assumption violations occur.

The complexity of GR(1) synthesis with recovery is the same as for the GR(1) synthesis without recovery [4]. The recovery algorithm gives a list of next possible actions of the robot, which is of size exponential in the number of atomic propositions. This is the same as for the original GR(1) synthesis algorithm. Admittedly, the list length often grows a bit when recovery transitions are added, but the overall complexity stays the same. In our experiments, we showed that this growth is not prohibitive in practice.

If environment assumption violations keep occurring, they can prevent the robot from fulfilling its mission. In that case, the environment assumptions should be rewritten as described in Section VI.

VI. Environment Characterization and Resynthesis

In this section, we present an approach for online construction of the observed environment assumption approximation φ_e'' (described in Problem 1), followed by controller resynthesis. Recall that we are concerned with GR(1) specifications, so φ_e'' will be of the form $\varphi_e''^i \wedge \varphi_e''^t \wedge \varphi_e''^g$. Using runtime verification of the environment safety assumptions $\varphi_e''^t$ (Section VI-A), we detect violations of $\varphi_e''^t$ when they occur. We then update $\varphi_e''^t$ based on the information gathered during execution and resynthesize the controller if needed, as described in Section VI-B. Whenever for some approximation $\varphi_e''^t$ the specification $\varphi_e'' \rightarrow \varphi_s$ is unsynthesizable, our algorithm provides feedback to the user with the analysis tool developed by [22] and asks for constraints to be added to $\varphi_e''^g$. We use the environment assumptions $\varphi_e = \varphi_e^i \wedge \varphi_e^t \wedge \varphi_e^g$ already provided by the user as an initial approximation for $\varphi_e'' = \varphi_e''^i \wedge \varphi_e''^t \wedge \varphi_e''^g$. Without loss of generality, we assume that all environment transition constraints have been merged into one, i.e., we have $\varphi_e^t = \square\,\psi_e^t$ for some subformula ψ_e^t. We can always perform such a merge by taking $\psi_e^t = \bigwedge_{(\square\,\psi) \text{ is a conjunct of } \varphi_e^t} \psi$.

A. Runtime Verification

To monitor the environment safety assumptions during execution, we parse each safety assumption into a tree structure. The valuation of the robot and environment propositions $\mathcal{Y}$ and $\mathcal{X}$ are used together with the latest sensor values for $\mathcal{X}'$ to evaluate the parsed tree at each timestep of the mission execution, thus detecting environment assumption violations.

B. Environment Characterization and Resynthesis

In essence, the environment characterization process iteratively relaxes the assumptions on the environment, based on actual observed environment behavior, until either the robot completes its task or a specification becomes unsynthesizable at which point the user is asked to provide liveness assumptions. This section describes when the formula $\varphi_e''^t$ is updated and what it is updated to.

Whenever environment assumptions are violated, $\varphi_e''^t$ is modified. Given the recovery transitions (Section V), the robot may be able to complete its task without updating the assumptions. In that case, we do not perform resynthesis. If the robot cannot guarantee progress towards its goals, a new controller is synthesized with the updated $\varphi_e''^t$. This occurs in the following situations: (i) the recovery transition is a self transition ($s_i \equiv s_{i+1}$), meaning that the current state satisfies system safety φ_s^t but the recovery behavior is to wait until the assumption violation is over, or (ii) the recovery transitions cause the robot to change state ($\delta_s(s_{i-1}, x_i) \neq \delta_s(s_i, x_{i+1})$) but there could be a livelock situation. We treat possible livelock situations conservatively; if the environment assumption is continuously violated, i.e. the robot is only taking recovery transitions, and the robot has made N state transitions (N being user defined), we rewrite φ_e'' and resynthesize the robot controller. While this may result in unnecessary synthesis calls, it avoids environment induced livelocks.

The environment characterization and resynthesis algorithm, as described in Algorithm 2, relaxes the environment safety assumption by adding disjuncts to $\varphi_e''^t$ based on observed behavior. We consider two types of updates: Formula 2 where current environment states are added as allowed and Formula 3 where environment transitions (pairs of current and next states) are added. The former allows for more environment behaviors and is attempted first (line 7). However, if the specification remains unsynthesizable, the latter is attempted.

$$\square(\psi_e^t \vee \bigvee_{i=0}^{k}(\bigwedge_{x \in x_i} x \wedge \bigwedge_{x \in \mathcal{X} \setminus x_i} \neg x)) \tag{2}$$

$$\square(\psi_e^t \vee \bigvee_{i=0}^{k}(\bigwedge_{x \in x_i} x \wedge \bigwedge_{x \in \mathcal{X} \setminus x_i} \neg x \wedge \bigcirc(\bigwedge_{x \in x_{i+1}} x \wedge \bigwedge_{x \in \mathcal{X} \setminus x_{i+1}} \neg x))) \tag{3}$$

If the specification is still unsynthesizable with the more restrictive assumption (Formula 3), we employ the analysis tool of [22] to identify the environment liveness condition φ_e^{*g} which, when appended to the specification as $\varphi_e''^g = \varphi_e^g \wedge \varphi_e^{*g}$, makes the specification realizable. If this creates a synthesizable specification, the algorithm then returns to using the less restrictive environment behavior of Formula 2.

The algorithm allows the user to start with a specification in which the environment safety assumptions are $\square(\mathbf{false})$, i.e. the environment has no possible behaviors and the system is winning. The approach will then automatically characterize the environment during execution, generating a realizable specification if one exists. If the environment cannot be described by a formula in GR(1) over the environment propositions, $\varphi_e''^t$ can overapproximate the real environment transition constraints, up to the point of becoming equivalent to $\square(\mathbf{true})$. Otherwise, our approach guarantees that we always have $\varphi_e''^t \rightarrow \varphi_e'^t$ for the actual (unknown) transition constraint $\varphi_e'^t$ that the physical environment adheres to, provided that the algorithm

Algorithm 2 Runtime Environment Characterization. The inputs are the current inputs x_i, the incoming environment inputs x_{i+1}, the current system outputs y_i and the specification $\varphi = \varphi_e^i \wedge \varphi_e^t \wedge \varphi_e^g \rightarrow \varphi_s^i \wedge \varphi_s^t \wedge \varphi_s^g$.

procedure ENVCHARACTERIZATION
  $\varphi_e^{''t} \leftarrow \varphi_e^t$, $\varphi_e^{*g} = \emptyset$, $i \leftarrow 0$
  while not abortExecution **do**
    Update Eq. 2 and Eq. 3
    if $(x_i, y_i)x_{i+1} \not\models \psi_e^{''t}$ and ($s_i \equiv s_{i+1}$ or $\delta_s(s_{i-1}, x_i) \neq \delta_s(s_i, x_{i+1})$ for N times) **then**
      $\varphi^i \leftarrow (\varphi_e^i \wedge \varphi_s^i) \vee$ mapToLTL($x_i \cup y_i$)
      $\varphi_e^{''t} \leftarrow$ Eq. 2
      $Aut \leftarrow$ SYNTHESIS($\varphi = \varphi_e'' \rightarrow \varphi_s$)
      if $\neg Aut$ (φ is unrealizable) **then**
        $\varphi_e^{''t} \leftarrow$ Eq. 3
        $Aut \leftarrow$ SYNTHESIS($\varphi = \varphi_e'' \rightarrow \varphi_s$)
    if $\neg Aut$ (φ is still unrealizable) **then**
      $\varphi_e^{*g} \leftarrow \varphi_e^{*g} \wedge$ addEnvLivenessByUser()
      $\varphi_e^{''g} = \varphi_e^{*g} \wedge \varphi_e^g$
      $Aut \leftarrow$ SYNTHESIS($\varphi = \varphi_e'' \rightarrow \varphi_s$)
      if $\neg Aut$ (φ is unrealizable with user input) **then**
        abortExecution
    $i \leftarrow i + 1$

switched to using Formula 3. If this approximation quality is of importance, we can alternatively start with that formula.

Our approach allows the robot to make as much progress towards its goals as possible while at the same time reducing the number of resynthesis occurrences, as they are relatively expensive. If the unexpected environment behavior is sporadic, recovery alone typically suffices as long as the robot is safe. Whenever the robot cannot make progress towards its goal, we employ environment characterization to synthesize a controller that is correct for the observed environment behavior.

VII. EXAMPLE

Example 2: Consider a robot performing a package delivery task as defined in Specification 2, in the workspace shown in Fig 1. The robot starts in *hallway* with all sensors false (line 1-2). It then visits *mailroom* to pick up a package if it is notified that a package is ready for pickup by the *packageReady* sensor (line 6, 9-10). With the package in hand, the robot must deliver it to *office* (line 7-8, 11). In addition, the robot cannot enter the *atrium* when the sensor *betweenClasses* is true, and it cannot enter the *kitchen* if *cooking* is true (line 4-5). The *door* region is blocked if the $doorClosed$ sensor is on (line 3).

Note that Specification 2 does not make assumptions about the behavior of the environment; as such, it is unrealizable since the environment could have the *doorClosed* sensor on forever. This prevents the robot from reaching *office* and delivering its package.

The experiments were conducted with an Aldebaran Nao with pose provided by a Vicon motion capture system. Figure 3 depicts the items used to represent the environment events: $packageReady$ is a user input; $betweenClasses$ and $cooking$

Specification 2 A package delivery task

Component of φ_e^i and φ_s^i
1 $(\pi_{Hallway} \wedge \neg\pi_{deliver} \wedge \neg\pi_{pickup})$
2 $(\neg\pi_{packageReady} \wedge \neg\pi_{doorClosed} \wedge \neg\pi_{cooking} \wedge \neg\pi_{betweenClasses})$
Component of φ_s^t
3 $\square(\pi_{doorClosed} \rightarrow \neg\pi_{door})$
4 $\square(\pi_{cooking} \rightarrow \neg\pi_{kitchen})$
5 $\square(\pi_{betweenClasses} \rightarrow \neg\pi_{atrium})$
6 $\square(\pi_{mailroom} \wedge \pi_{packageReady} \wedge \neg\pi_{obtainedPackage}) \leftrightarrow (\pi_{pickup})$
7 $\square(\pi_{office} \wedge \pi_{obtainedPackage}) \leftrightarrow (\pi_{deliver})$
8 $\square((\pi_{pickup} \wedge \neg\pi_{deliver}) \rightarrow \bigcirc\pi_{obtainedPackage})) \wedge$
$\square(\pi_{deliver} \rightarrow \neg\bigcirc\pi_{obtainedPackage})) \wedge$
$\square((\pi_{obtainedPackage} \wedge \neg\pi_{deliver}) \rightarrow \bigcirc\pi_{obtainedPackage}) \wedge$
$\square((\neg\pi_{obtainedPackage} \wedge \neg\pi_{pickup}) \rightarrow \neg\bigcirc\pi_{obtainedPackage})$
9 $\square(\pi_{obtainedPackage} \rightarrow \neg\bigcirc\pi_{mailroom})$
Component of φ_s^g
10 $\square\diamondsuit((\pi_{packageReady} \wedge \neg\pi_{obtainedPackage}) \rightarrow \pi_{mailroom})$
11 $\square\diamondsuit(\pi_{obtainedPackage} \rightarrow \pi_{office})$

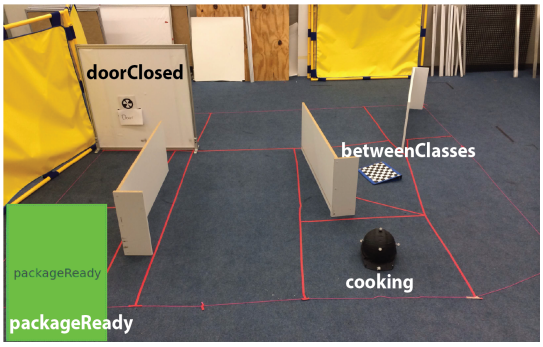

Fig. 3: Sensors used for Example 2: packageReady, doorClosed, cooking and betweenClasses

are detected using the Vicon system and $doorClosed$ is a radial barcode detected by the Nao's built-in vision system. The accompanying video captures all of the following examples.

A. Unrealizable Specification

As described in Section VI-B, when we have no information about the environment, we synthesize a robot controller for a 'False' environment. At the start of the experiment, the environment safety condition $\square(\mathbf{false})$ does not hold and we construct $\varphi_e^{''t}$ in Algorithm 2 by appending the current observations of the environment. As all the sensor values are **false** initially, this results in the new environment safety condition $\varphi_e^{''t} = \square(\mathbf{false} \vee (\neg\pi_{packageReady} \wedge \neg\pi_{doorClosed} \wedge \neg\pi_{cooking} \wedge \neg\pi_{betweenClasses}))$.

With the modified specification being realizable, we resynthesize and continue the execution. At this time, the sensor $packageReady$ is true and the robot should visit $mailroom$ and pick up the package. However, with the previous safety assumption, the generated controller has no transitions going to $mailroom$, since $packageReady$ is assumed to always be false. As the robot is stuck in the current state, the current observed environment ($\pi_{packageReady} \wedge \neg\pi_{doorClosed} \wedge \neg\pi_{cooking} \wedge \neg\pi_{betweenClasses}$) is appended to the environment safety formula $\varphi_e^{''t}$ and a controller is resynthesized. The robot proceeds to *mailroom* and picks up the package.

As the robot is moving towards $door$, it discovers that the door is closed, thus violating the assumptions. Here, the recovery only has a self transition causing the robot to wait in front of the door until it is opened again. The addition of current (Formula 2) and next (Formula 3) inputs to the environment safety formula cannot resolve the unrealizability of the specification. We then use the specification analysis tool and figure out the necessary environment liveness condition for a realizable specification; we find out the robot cannot proceed to *office* if the door is closed forever. With the addition of the environment liveness $\square\diamondsuit(\neg\pi_{doorClosed})$, the specification is realizable again and the characterization of the environment behaviors is reset to Formula 2.

With the door reopened, $atrium$ is now occupied by the students between classes. The robot cannot proceed to its goals with the current controller and thus the clause ($\pi_{packageReady} \wedge \neg\pi_{doorClosed} \wedge \neg\pi_{cooking} \wedge \pi_{betweenClasses}$) is appended to the environment safety formula $\varphi_e^{''t}$ and a new controller is synthesized. The robot then takes the lower path.

As the robot reaches $classroom$, $atrium$ reopens while $kitchen$ is now occupied by the chef. With only a self transition existing for the current state in the controller, the current environment behavior ($\pi_{packageReady} \wedge \neg\pi_{doorClosed} \wedge \pi_{cooking} \wedge \neg\pi_{betweenClasses}$) is appended to φ_e^t. However, the specification is unrealizable as the environment can block $kitchen$ when the robot is in $classroom$, and $atrium$ when the robot is in $corridor$, creating a livelock. Instead, we add the current environment behavior and the next environment behavior from the sensors ($\pi_{packageReady} \wedge \neg\pi_{doorClosed} \wedge \pi_{cooking} \wedge \neg\pi_{betweenClasses} \wedge \bigcirc\pi_{packageReady} \wedge \neg\bigcirc\pi_{doorClosed} \wedge \bigcirc\pi_{cooking} \wedge \neg\bigcirc\pi_{betweenClasses}$) to the environment safety formula $\varphi_e^{''t}$. The robot will now go back to $corridor$ and take the upper path.

The robot will be able to go from $corridor$ to $atrium$ with the highly restrictive environment and finally reaches *office* and delivers the package. This example is included in the supplement video as the first example.

B. Recovery Approach

Consider Specification 2 augmented with the environment assumptions listed in Specification 3, that state: the door is never closed (line 1); if the robot is in $corridor$ then $betweenClasses$ must be false (line 2); the chef is not in $kitchen$ when the robot is in $mailroom$ picking up the package (line 3); and the chef shall leave $kitchen$ at some point (line 4). With the addition, the new specification is now realizable and a controller is generated.

Specification 3 Additional specification to the package delivery mission

1 $\square(\neg\pi_{doorClosed})$
2 $\square(\pi_{corridor} \rightarrow \neg\bigcirc\pi_{betweenClasses})$
3 $\square((\pi_{mailroom} \wedge \pi_{packageReady}) \rightarrow \neg\bigcirc\pi_{cooking})$
4 $\square\diamondsuit(\neg\pi_{cooking})$

In the experiment, the robot starts in $hallway$. With the notification of package is ready for pickup, the robot visits $mailroom$ and picks up the package. When the robot is in $mailroom$, $kitchen$ is occupied by the chef and the $packageReady$ sensor is on. According to Specification 3, the chef should have left the kitchen when package is ready and the robot is in $hallway$. In this case, the environment safety assumption is violated.

Since the robot can leave $mailroom$ with no violations of the system safety guarantees, the robot will move on to deliver the package, using the recovery transitions. On its way to *office*, in $hallway$, the chef left $kitchen$ and $betweenClasses$ becomes true when the robot enters $corridor$, the environment safety assumption $\square(\pi_{corridor} \rightarrow \neg\bigcirc\pi_{betweenClasses})$ in Specification 3 is violated.

Because the shortest path to the office is through the $atrium$, and since the recovery approach only adds extra transitions to states where the robot is safe, the recovery approach causes the robot to wait in $corridor$ until it can go through $atrium$. Once classes start again ($\pi_{betweenClasses}$ is **false**), the robot can go from $atrium$ to *office*. This example is included in the supplement video as the second example.

If the recovery and the environment characterization approaches are used together, when one of the environment safety assumptions is violated and the robot cannot make progress towards its goal, we add in the environment behavior that we have observed so far as in Eq. 2 or Eq. 3 and resynthesize. With this change, the robot moves to $classroom$, $kitchen$ and finally reaches *office* and delivers the package as seen in the third example in the supplement video.

VIII. Conclusions

In this paper, we address the problem of generating provably correct controllers from high-level specifications for robots operating in unexpected or unknown environments. We present an approach that recovers from violations of environment safety assumptions and captures the actual behavior of the environment when unexpected environment behavior is observed. In previous work, such violations would cause the robot to stop since the synthesized controller did not contain any next action for such situations. Furthermore, the robot would not indicate why it has stopped executing its task.

Our approach allows the user to identify and detect the violations of environment safety assumptions during execution. Our modified synthesis algorithm creates controllers that ensure the robot makes progress towards its goals, if possible, in the event of such violations. In contrast to other related approaches, the generated controller satisfies its specification by pursuing its own goals rather than falsifying the environment; the latter of which may create correct controllers exhibiting undesirable behavior. In addition, we modify the specification on the fly and correct the user's false or incomplete assumptions about the environment behavior.

Acknowledgments

This work was supported by NSF ExCAPE. The first and third authors were supported by DARPA N66001-12-1-4250.

REFERENCES

[1] A. I. Medina Ayala, Sean B. Andersson, and Calin Belta. Probabilistic control from time-bounded temporal logic specifications in dynamic environments. In *ICRA*, pages 4705–4710, 2012.

[2] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J. Pappas. Symbolic planning and control of robot motion [Grand Challenges of Robotics]. *IEEE Robot. Automat. Mag.*, 14(1):61–70, 2007.

[3] Amit Bhatia, Lydia E. Kavraki, and Moshe Y. Vardi. Sampling-based motion planning with temporal goals. In *ICRA*, pages 2689–2696, 2010.

[4] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[5] Yushan Chen, Jana Tumova, and Calin Belta. LTL robot motion control based on automata learning of environmental dynamics. In *ICRA*, pages 5177–5182, 2012.

[6] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[7] Rüdiger Ehlers. Generalized Rabin(1) Synthesis with Applications to Robust System Synthesis. In *NASA Formal Methods*, pages 101–115, 2011.

[8] Rüdiger Ehlers and Ufuk Topcu. Resilience to Intermittent Assumption Violations in Reactive Synthesis. In *HSCC*, pages 203–212, 2014.

[9] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. LTLMoP: Experimenting with language, Temporal Logic and robot control. In *IROS*, pages 1988–1993, 2010.

[10] Jie Fu, Herbert G. Tanner, and Jeffrey Heinz. Adaptive planning in unknown environments using grammatical inference. In *CDC*, pages 5357–5363, 2013.

[11] Meng Guo, Karl Henrik Johansson, and Dimos V. Dimarogonas. Revising motion planning under Linear Temporal Logic specifications in partially known workspaces. In *ICRA*, pages 5025–5032, 2013.

[12] Sertac Karaman, Ricardo G. Sanfelice, and Emilio Frazzoli. Optimal control of Mixed Logical Dynamical systems with Linear Temporal Logic specifications. In *CDC*, pages 2117–2122, 2008.

[13] Marius Kloetzer and Calin Belta. A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications. *IEEE Trans. Automat. Contr.*, 53(1):287–297, 2008.

[14] Robert Könighofer, Georg Hofferek, and Roderick Bloem. Debugging formal specifications using simple counterstrategies. In *FMCAD*, pages 152–159, 2009.

[15] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal Logic based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[16] Orna Kupferman and Moshe Y. Vardi. Model Checking of Safety Properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[17] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[18] Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. Mining assumptions for synthesis. In *MEMOCODE*, pages 43–50, 2011.

[19] Scott C. Livingston, Richard M. Murray, and Joel W. Burdick. Backtracking temporal logic synthesis for uncertain environments. In *ICRA*, pages 5163–5170, 2012.

[20] Scott C. Livingston, Pavithra Prabhakar, Alex B. Jose, and Richard M. Murray. Patching task-level robot controllers based on a local mu-calculus formula. In *ICRA*, pages 4588–4595, 2013.

[21] Rupak Majumdar, Elaine Render, and Paulo Tabuada. Robust discrete synthesis against unspecified disturbances. In *HSCC*, pages 211–220, 2011.

[22] Vasumathi Raman and Hadas Kress-Gazit. Automated feedback for unachievable high-level robot behaviors. In *ICRA*, pages 5156–5162, 2012.

[23] Vasumathi Raman and Hadas Kress-Gazit. Towards minimal explanations of unsynthesizability for high-level robot behaviors. In *IROS*, pages 757–762, 2013.

[24] Ufuk Topcu, Necmiye Ozay, Jun Liu, and Richard M. Murray. On synthesizing robust discrete controllers under modeling uncertainty. In *HSCC*, pages 85–94, 2012.

[25] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *HSCC*, pages 101–110, 2010.