2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

# Analyzing and Revising High-Level Robot Behaviors Under Actuator Error

Benjamin Johnson and Hadas Kress-Gazit

*Abstract*— **One increasingly popular approach for creating robot controllers for complex tasks is to automatically synthesize a hybrid controller from a high-level task specification. Such an approach, in addition to reducing the time and expertise required for creating a controller, guarantees that the robot will satisfy all of the underlying specifications, given perfect sensing and actuation. This paper investigates the probabilistic guarantees that can be made about the behavior of the robot when the actuation of the robot is no longer assumed to be perfect, as well as the possible specification revisions that can be made to improve the behavior of the robot. The approach described in this paper composes probabilistic models of the environment behavior and the robot actuation error with the synthesized controller, and uses probabilistic model checking techniques to find the probability that the robot satisfies a set of high level specifications. This paper also presents a preliminary approach for analyzing the composed model and automatically generating revisions to improve the robot's high-level behavior.**

## I. Introduction

Creating robotic controllers for complex tasks is a difficult process, often requiring a high level of technical expertise and a great deal of effort to program and debug the controller. Additionally, extensive testing is required to validate the behavior of the controller; such testing can be a time consuming and expensive process, and cannot guarantee the performance of the controller for all possible situations. In some cases, experimental validation of the controller may even be infeasible due to complexity, time requirements, cost, or the availability of data.

One potential method for lessening this burden is to use formal techniques to automatically synthesize correct-by-construction hybrid controllers from high-level task specifications [1]–[13]. Such methods take advantage of formal synthesis techniques to provide guarantees about the behavior of the controller. The correct-by-construction controllers synthesized using the approach in [7], for example, offer the guarantee that, when the robot's sensing and actuation are perfect, the robot will behave in a manner that satisfies all of the specifications (if a controller is generated).

Unfortunately, this assumption of perfect sensing and actuation is often inappropriate for practical systems. Consider, for example, the modified iRobot Create, pictured in Figure 1. This robot is outfitted with a vacuum-driven universal gripper [14], and is tasked with finding a number of small objects in its workspace, picking them up, and delivering them to a particular location. During operation, however, it becomes apparent that, due to errors in sensing (e.g. mis-detecting the location of an object) and actuation (e.g. prematurely dropping an object), the robot is not able to perfectly execute its task.

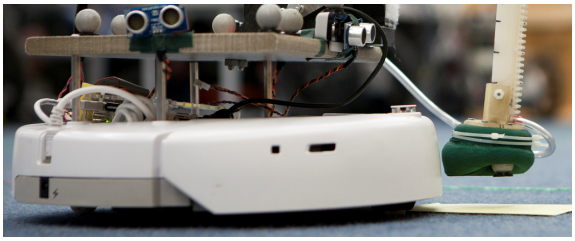


Fig. 1. Photograph of an iRobot Create, modified with additional sensors and a universal gripper to pick up objects. Photo source: Lindsay France.

This paper considers formally synthesized controllers, and relaxes the assumption of perfect actuation, analyzing the controller to determine what probabilistic guarantees may then be made about the behavior of the robot. Additionally, this paper presents a method for automatically providing feedback to the user, in the form of suggested revisions to the original specification, with the goal of improving the behavior of the synthesized controller.

Related work includes [3], in which the authors present an approach for generating a feedback control strategy that satisfies a temporal logic specification, despite disturbances in the continuous dynamics of the system. Building on that work, the authors of [8] and [11] present an approach in which they model the uncertainty in the outcomes of the low-level continuous controllers by a set of probabilities on the discrete transitions of the system. Leveraging temporal logic model checking techniques, the authors then find the control strategy that maximizes the probability that the controller satisfies the specification. In [15] the authors improve the computational feasibility of this technique by using dynamic programming techniques to obtain an approximately optimal solution. In [16], the authors present a method for synthesizing control strategies such that the synthesized controller maximizes the probability that the robot satisfies a temporal logic specification within a given time-bound.

In [17], the authors address a different type of environmental uncertainty: that of changes in the topology of the robot's workspace, which may render execution of the synthesized automaton infeasible. They propose a method for locally resynthesizing "patches" of the automaton to navigate around the blocked topology, while maintaining the guarantees on the behavior of the robot and avoiding the computational burden of resynthesizing a global solution.

Additionally, the work presented here is closely related to

This work supported by NSF CNS-0931686, DARPA N66001-12-1-4250.
Both authors are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA {blj39,hadaskg}@cornell.edu

the authors' previous work in [18] and [19], which presents a method for analyzing the probabilistic guarantees that can be made when actuators are assumed to be perfect, but sensors exhibit false positives and false negatives. In these papers, the authors create a probabilistic model of the system, modeling the sensor inaccuracies with a set of abstract propositions that imperfectly mimic the state of the environment. They then apply model-checking techniques to find the probability that the model satisfies a set of temporal logic specifications. In [13], the authors applied this approach to an autonomous car, where the behavior of other vehicles is probabilistically anticipated, and analyzed the effect of the uncertainty in that anticipation on the safety of the vehicle's behavior.

The work described in this paper presents a method that compliments that proposed in [18] and [19], and allows for the probabilistic analysis of controllers under actuation uncertainty. Specifically, the discrete model of the controller, which is obtained from the synthesis algorithm, is expanded to include new transitions and states caused by inaccuracies in the system actuation. The new probabilistic system is then evaluated using a probabilistic model-checker.

This paper also presents a method for using the evaluation of the probabilistic system to automatically generate specification revisions for the consideration of the user. The use of probabilistic analysis to automatically inform revisions of the specification is a novel approach and is not, to the knowledge of the authors, presented elsewhere in the literature. The most closely related work is that of [20] and [21], in which the authors present a method for revising a temporal logic specification that is found to be unsynthesizable, by relaxing the initial specification with the minimal changes that result in a synthesizable specification. This is in contrast to the proposed approach, which suggests revisions to the already synthesizable specification to improve the probabilistic behavior of the synthesized controller.

The paper is organized as follows. Section II defines preliminary concepts. Section III provides a detailed description of the problem addressed in this paper. Section IV describes the approach, which Section V demonstrates with several examples. The paper concludes in Section VI.

## II. Preliminaries

### A. Linear Temporal Logic

Linear Temporal Logic (LTL) formulas are defined over a set of Boolean propositions $\Pi$. A formula $\phi$ is defined recursively, as shown in Equation 1.

$$\phi ::= true \mid \pi \in \Pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \qquad (1)$$

The Boolean operators $\neg$ ("not") and $\wedge$ ("and") can be used to define the additional Boolean operators $\vee$ ("or"), $\rightarrow$ ("implies"), and $\leftrightarrow$ ("if and only if"). Similarly, the temporal operators $\bigcirc$ ("next") and $\mathcal{U}$ ("until") can be used to define the additional temporal operators $\Diamond$ ("eventually") and $\Box$ ("always").

The semantics of LTL are defined over an infinite sequence of states, where each state is labeled with a truth assignment to the set of propositions $\Pi$. Intuitively, the formula $\bigcirc\phi$ is true if and only if the formula $\phi$ is true in the next time step. The formula $\Diamond\phi$ is true if and only if $\phi$ is true at some point in the future, and $\Box\phi$ is true if and only if $\phi$ holds for the current state and all future states. A more complete description of the semantics can be found in [22].

### B. Synthesized Controller

The robot controllers used in this paper are synthesized using the approach described in [7], and take the form of an automaton $\mathcal{A} = \{S, S_0, \mathcal{X}, \delta, \mathcal{Y}, L, \Gamma\}$, where $S$ is a set of states and $S_0 \subseteq S$ is the set of initial states. The transition relation $\delta : S \times 2^{\mathcal{X}} \rightarrow S$ defines the possible next states for a given state and subset of the input alphabet $\mathcal{X}$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is the set of Boolean propositions describing the abstract characteristics of the environment state (e.g. $trash$ may represent the presence of a piece of trash in front of the robot). The labeling function $L : S \rightarrow 2^{\mathcal{Y}}$ labels each state with a subset of the robot propositions in $\mathcal{Y}$, where $\mathcal{Y} = \{y_1, \ldots, y_m\}$ is the set of propositions representing the location of the robot in a partitioned workspace and the abstract actions for the robot (e.g. $pickup$ may refer to using a manipulator to pick up an object). Finally, since the robot may have multiple goals it must eventually satisfy (e.g. take the $trash$ to the $TrashCan$), $\Gamma : S \rightarrow \mathbb{N}$ is a function which maps each state to a positive integer, representing the index of the robot's high-level goal, at that state.

### C. Discrete-Time Markov Chains

A Discrete-Time Markov Chain (DTMC) is defined as $\mathcal{D} = \{S, S_0, \Delta, \Pi, L\}$, where $S$ and $S_0$ are the states and initial states, respectively. The transition function $\Delta : S \times p \rightarrow S$ defines the probability $p \in (0, 1]$ for each possible transition between states. The labeling function $L : S \rightarrow 2^{\Pi}$ labels each state with a subset of the propositions in $\Pi$.

## III. Problem Statement

This paper considers robot controllers that are synthesized from a set of temporal logic specifications defining the desired behavior of the robot, such that when the robot's sensors and actuators are perfectly accurate the controller is guaranteed to satisfy the underlying specifications. This paper investigates the probabilistic guarantees that can be made for the controller, when the robot's actuators are no longer assumed to be perfect (i.e. the outcome of an actuator or low-level controller may not be the intended outcome), and presents a preliminary method for using these guarantees to inform the revision of the original specification.

**Example 1:** Consider the four-region workspace shown in Figure 2. The robot has a single sensor $\mathcal{X} = \{x\}$, and is capable of moving between regions in the workspace $\mathcal{Y} = \{R_1, R_2, R_3, R_4\}$. The robot is tasked with moving between regions $R_2$ and $R_3$ infinitely often, while avoiding $R_1$ when $x$ is true and avoiding $R_4$ when $x$ is false.

The high-level task specification for the robot is written using the General Reactivity (1) fragment of LTL, restricting the specification to the form $\varphi = \varphi_e \rightarrow \varphi_s$ [23]. Such a

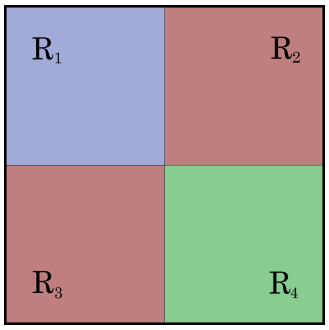



Fig. 2. Environment workspace Example 1.

format prescribes a desired robot behavior (defined by $\varphi_s$) for a set of environment behaviors (defined by $\varphi_e$). A subset of $\varphi_s$ for Example 1 is shown below, where specifications 1 and 2 are safety specifications (things the robot should always do or avoid) and specifications 3 and 4 are liveness specifications (tasks it should repeatedly accomplish).

1) $\square(\bigcirc x \rightarrow \bigcirc\neg R_1)$ "Avoid $R_1$ when $x$ is $true$"
2) $\square(\bigcirc\neg x \rightarrow \bigcirc\neg R_4)$ "Avoid $R_4$ when $x$ is $false$"
3) $\square\lozenge(R_2)$ "Repeatedly visit $R_2$"
4) $\square\lozenge(R_3)$ "Repeatedly visit $R_3$"

The environment behavior is modeled with a set of probabilities $P(X'|X,Y)$ describing the change in the abstract environment propositions. Each new environment state $X'$ is modeled such that it is dependent only on the preceding environment and robot states ($X$ and $Y$, respectively). Additionally, uncertainty in the robot actuation is modeled by a set of probabilities $P(Y'|\overline{Y}',Y)$ describing the probabilistic evolution of the robot propositions. Each new robot state $Y'$ is dependent only on the preceding robot state $Y$, and the desired next robot state $\overline{Y}'$ as given by the automaton.

Previous work [7] has assumed that the robot actuators operate without possibility of failure. That is, all robot propositions change precisely as desired, such that $P(Y'|\overline{Y}',Y) = 1$ when $Y' = \overline{Y}'$, and $P(Y'|\overline{Y}',Y) = 0$ when $Y' \neq \overline{Y}'$. This paper investigates the following problem.

**Problem:** Given a synthesized robot controller $\mathcal{A}$, and probabilistic models of the environment $P(X'|X,Y)$ and imperfect robot actuation $P(Y'|\overline{Y}',Y)$, find the probability that the system satisfies some set of high-level specifications. Additionally, provide the user with possible revisions to the original specification, aimed towards avoiding some undesirable behavior of the system (as defined by the user).

## IV. Approach

The overall approach proposed in this paper is illustrated in Figure 3. The robot controller is synthesized from a set of high-level task specifications, as well as abstract models of the environment and robot. This controller is then used, along with probabilistic characterizations of the environment event probabilities and robot actuation error, to compose a model for the complete system. A probabilistic model checker is then used to find the probability that the composed model satisfies some set of temporal logic specifications. Additionally, the composed model is analyzed to provide the user with potential revisions to the original task specification.

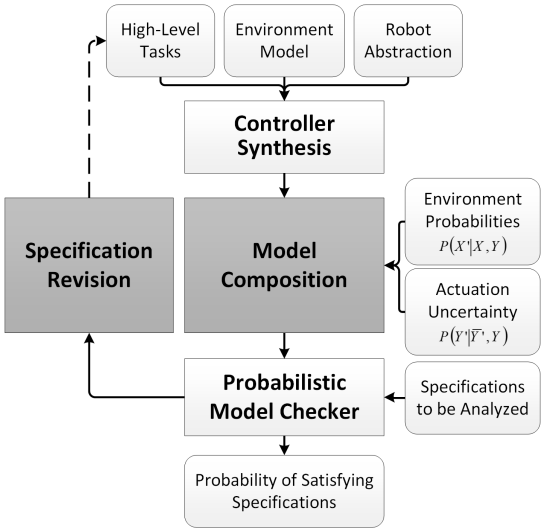



Fig. 3. Diagram of the approach presented in this paper, where an automatically synthesized robot controller is composed with models of the environment and actuation error, and used to find probabilistic guarantees on the behavior of the robot or to suggest revisions to the original specification.

### A. Error Analysis

The probabilistic model construction is described in Algorithm 1. The inputs to the algorithm are the controller automaton $\mathcal{A}$, the set of probabilities governing the changes in the state of the environment $P(X'|X,Y)$, and the set of probabilities characterizing the error in the outcome of the robot actuation $P(Y'|\overline{Y}',Y)$. The output of the algorithm, $\mathcal{D}$, is the DTMC for the complete system model.

The input probabilities for the environment and actuation uncertainty can be estimated using statistics from experimentation or simulation. In [11], for example, the authors discuss the experimental determination of outcome probabilities for motion primitives; these probabilities are then used to synthesize an optimal motion plan for the robot. In [13], the authors use simulated data to obtain estimated success rates for high-level sensor values, which are used to analyze the performance of a synthesized controller. Such methods could be similarly applied to obtain estimated values for the modeled uncertainty in actuation outcome.

Consider Example 1, from Section III. The synthesized 8-state controller, $\mathcal{A}$, is shown in Figure 4. Each state is labeled with the state number, the goal index for that state $\Gamma$ (given in parentheses), and its region (denoting the robot's location). Each transition is labeled with the input sensor value, where $x$ refers to the sensor being true and $\neg x$ refers to it being false ($\neg x$ is equivalent to a transition label of $X = \emptyset$).

For this example, let the sensor $x$ be modeled such that it is independent of both its previous value and the state of the robot, where $P(x') = 0.65$ and $P(\neg x') = 1 - P(x') = 0.35$. The actuator uncertainty, on the other hand, is modeled such that the region transition $Y'$ is dependent on both the current region $Y$ and the intended next region $\overline{Y}'$. These probabilities are modeled such that the robot can reliably stay in its current region: $P(R_i'|\overline{R_i}',R_i) = 1$ and $P(R_j'|\overline{R_i}',R_i) = 0$ for $j \neq i$. When moving out of its current region, the robot will move to its intended destination with a probability of 0.9, while it has

**Algorithm 1** Create complete system model, with actuation error.

1: **procedure** COMPOSEMODEL($\mathcal{A} = \{S, S_0, \mathcal{X}, \delta, \mathcal{Y}, L, \Gamma\},\ P(X'|X,Y), P(Y'|\overline{Y}', Y)$)
2: $Q = \emptyset,\ Q_0 = \emptyset,\ \Delta = \emptyset$
3: **for** $s_j \in S$ **do**
4: $\mathcal{L}(q_j) = L(s_j) \cup X_j$ s.t. $\exists s_i, (s_i, X_j, s_j) \in \delta$
5: $Q = Q \cup q_j$
6: **if** $s_j \in S_0$ **then**
7: $Q_0 = Q_0 \cup q_j$
8: **for** $s_i \in S$ **do**
9: **for** $X_j \in 2^{\mathcal{X}}$ s.t. $\exists s_j \mid (s_i, X_j, s_j) \in \delta$ **do**
10: **for** $Y_k \in 2^{\mathcal{Y}}$ s.t. $p(Y_k|L(s_j), L(s_i)) > 0$ **do**
11: **if** $Y_k = L(s_j)$ and $(s_i, X_j, s_j) \in \delta$ **then** ▷ robot makes the correct state transition
12: $\Delta = \Delta \cup (q_i, p(X_j|X_i, L(s_i)) \cdot p(L(s_j)|L(s_j), L(s_i)), q_j)$ where $X_i = \mathcal{L}(q_i) \setminus L(s_i)$
13: **else if** $\exists s_h, s_k$ s.t. $(s_h, X_j, s_k) \in \delta$ and $L(s_k) = Y_k$ **then** ▷ labels describe existing states in $Q$
14: $s_l = argmin_{s_k}((\Gamma(s_j) - \Gamma(s_k))$ modulo $max(\Gamma))$ ▷ choose nearest preceding goal
15: $\Delta = \Delta \cup (q_i, p(X_j|X_i, L(s_i)) \cdot p(Y_k|L(s_j), L(s_i)), q_l)$ where $X_i = \mathcal{L}(q_i) \setminus L(s_i)$
16: **else** ▷ if an appropriate state does not exist
17: $\mathcal{L}(q_{j,k}) = X_j \cup Y_k$
18: $Q = Q \cup q_{j,k}$ ▷ add a deadlock state
19: $\Delta = \Delta \cup (q_i, p(X_j|X_i, L(s_i)) \cdot p(Y_k|L(s_j), L(s_i)), q_{j,k})$ where $X_i = \mathcal{L}(q_i) \setminus L(s_i)$
20: $\Delta = \Delta \cup (q_{j,k}, 1, q_{j,k})$ ▷ make the deadlock state a sink
21: **return** $\mathcal{D} = \{Q, Q_0, \Delta, \mathcal{X} \cup \mathcal{Y}, \mathcal{L}\}$

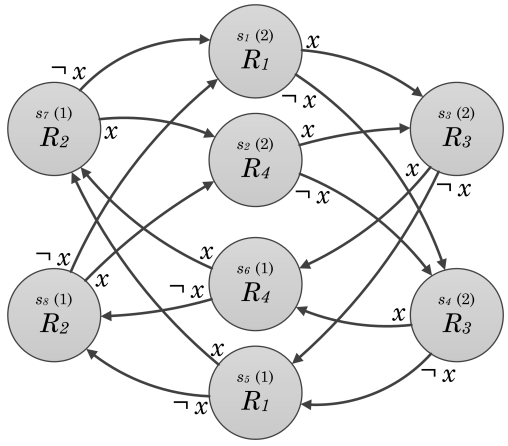



Fig. 4. Graphical representation of the robot controller for Example 1, as synthesized from the set of high-level specifications. Each state is labeled with the state number, the robot region, and the goal $\Gamma$, in parentheses. The state transitions are labeled with the sensor inputs.

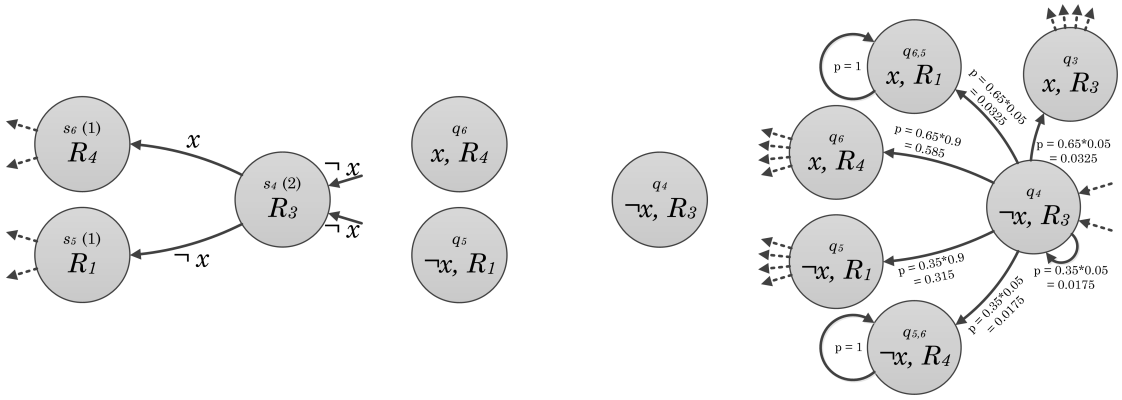
a 0.05 probability of moving into the other adjacent region and a 0.05 probability of remaining in its current region.

Figure 5 illustrates Algorithm 1 for the portion of the synthesized automaton shown in Figure 5(a), including states $s_4$, $s_5$, and $s_6$. For each state $s_j$ in the controller automaton, a new state $q_j$ is created for the composed system (robot and environment), and labeled (by the new labeling function $\mathcal{L}$) with the labels on the corresponding automaton state $L(s_j)$ as well as the inputs $X_j$ (representing the truth values of the sensor propositions) into that automaton state (lines 3-5). As in Figure 4, $X = \neg x$ is used in place of $X = \emptyset$ to represent the robot's sensor $x$ being false. If the automaton state in question is an initial state, then the corresponding new system state is added to the set of initial system states (lines 6-7). Figure 5(b) shows the composed system states $q_4$, $q_5$, and $q_6$, that are created from the automaton states $s_4$, $s_5$, and $s_6$, respectively.

Line 8 of the algorithm loops through each state $s_i$ in the automaton. Figure 5(c) shows the results for $s_i = s_4$. Line 9 then loops through each possible configuration for environment (each label $X_j$ on the transitions out of the automaton state $s_i$), and line 10 loops through each possible new robot configuration (any configuration $Y_k$ with a positive transition probability originating in $s_i$, with a desired next state as prescribed in the automaton for the inputs $X_j$). For this example, line 9 of the algorithm will loop through all possible sensor configurations: $X_j \in \{x, \neg x\}$. When $X_j = x$, the desired next automaton state is $s_j = s_6$, and line 10 loops through each possible next region when attempting to move from $R_3$ to $R_4$ (as is the prescribed action for $s_6$): $Y_k \in \{R_1, R_3, R_4\}$. The region $R_2$ is omitted from this set, since the robot cannot transition directly to it from $R_3$

If the robot makes the desired transition (i.e. the new robot configuration matches the labels on the desired next state $Y_k = L(s_j)$), then a transition is added to the corresponding system state with the appropriate probability, as evaluated from $P(X'|X,Y)$ and $P(Y'|\overline{Y}', Y)$ (lines 11-12). This can be seen in the example when $X_j = x$ and $Y_k = R_4$, and two such states exist in the automaton ($s_2$ and $s_6$). As $s_6$ is the desired next state in the automaton, a new transition is added to the corresponding state in the composed system $q_6$.

If, on the other hand, the environment and robot configuration match the labels on some other state in the automaton, then the transition to the corresponding new system state is added with the appropriate probability. Because the automaton may contain multiple states that share the same set of




(a) An excerpt of the synthesized controller for Example 1.

(b) Composed states for the Example 1 automaton excerpt (lines 3-7).

(c) Excerpt of the complete probabilistic model for Example 1.

Fig. 5. Illustrative example for Algorithm 1, applied to a excerpt of the synthesized automaton for Example 1, introduced in Section III.

labels, the state $s_l$ with a goal index most closely preceding the desired state $s_j$ is used, and the transition is added to the corresponding state $q_l$ in the composed model (lines 13-15). This is seen in the example, where $X_j = x$ and $Y_k = R_3$, where the one allowable state that exists in the automaton ($s_3$) is not the desired next state for the automaton. As such, the goal index $\Gamma$ of the state is compared to all other existing states with the corresponding labels (though in this case, no such states exist), and a transition with the appropriate probability is made to the corresponding state with the most closely preceding goal (in this case, $q_3$).

For the final case, if no such state, where $\mathcal{L}(q) = X_j \cup Y_k$, exists in the automaton, a new state $q_{j,k}$ is created and a transition to it is added with the appropriate probabilities (lines 16-19). That new state is then made a sink (transitions to itself with probability 1), since the automaton does not include a prescribed transition from such a state (line 20). This can be seen in the example, when $X_j = x$ and $Y_k = R_1$, since moving into region $R_1$ is not allowed when $x$ is true. The new state $q_{6,5}$ is then created, a transition is added with the appropriate transition probability, and the new state is made a sink.

This process is repeated for $X_j = \neg x$, resulting in transitions to $q_5$, $q_4$, and the creation of and transition to the new sink state $q_{5,6}$. The resulting excerpt of the output DTMC is shown in Figure 5(c).

The DTMC $\mathcal{D}$ that is output by this algorithm can be analyzed using a probabilistic model checker, and can be used to provide automated feedback to the user. The examples given in Section V of this paper are analyzed using the PRISM model checking software [24], which offers, among other capabilities, the ability to calculate the probability with which a DTMC model will satisfy an LTL formula.

### B. Revision Suggestion

In Algorithm 2 we present a method for providing automated feedback to the user, in the form of an LTL formula that may be added to the original specification to improve the behavior of the robot. This approach uses the complete system model (from Algorithm 1), along with the set of goal states $Q_{goal}$, deadlock states $Q_{deadlock}$, a specified number of look ahead steps $N$, and a temporal logic formula $\Psi$ describing undesirable robot behavior. One example for the formula $\Psi$ would be the negated conjunction of all of the original safety specifications (i.e. the formula representing all unsafe behaviors). The algorithm returns a possible LTL revision to the original task specification, for consideration and inclusion by the user. Intuitively, the algorithm finds the set of states that is most likely to cause the robot to exhibit undesirable behavior, and returns an LTL formula representing the common characteristics of those states.

On line 2 of the algorithm, the set of considered states $Q_{check}$ is restricted to exclude the initial states (which the robot *cannot* avoid), the goal states (which the robot does not *want* to avoid), and the deadlock states (which the robot is already *trying* to avoid). Line 3 then calls an external procedure (such as the previously mentioned model-checker: PRISM) to find the probability that each of the states in the restricted set exhibits, within $N$ steps, the undesirable behavior described by $\Psi$. If this procedure does not find any states with a positive probability of exhibiting the undesirable behavior, the algorithm exits with the suggestion of increasing the number of look-ahead steps $N$ (lines 4-5).

On line 6, the algorithm finds the set of states $Q^*$ that have the maximum probability of exhibiting the undesirable behavior. On lines 7 and 8, the algorithm finds the set of robot propositions $Y_T$ that are true in every state $q \in Q^*$ and the set of robot propositions $Y_F$ that are false in every state $q \in Q^*$. If both of these sets are empty (that is, there are no robot propositions that maintain a constant value for all of the states in $Q^*$), then the set of states $Q^*$ is reduced to the largest set of states that share a consistent value (either true or false) for at least one robot proposition (lines 9-15).

Lines 18 and 19 then find the set of environment propositions $X_T$ that are true and the set of environment propositions $X_F$ that are false, in every state $q \in Q^*$. The output for the

**Algorithm 2** Suggest revisions to the task specification.

1: **procedure** SUGGESTREVS($\mathcal{D} = \{Q, Q_0, \Delta, \mathcal{X} \cup \mathcal{Y}, \mathcal{L}\}$, $Q_{goal}, Q_{deadlock}, N, \Psi$)
2: $Q_{check} = Q \setminus (Q_0 \cup Q_{goal} \cup Q_{deadlock})$
3: $\mathcal{P} \leftarrow N\text{-}StepProbability(Q_{check}, \Psi, N)$
4: **if** $p = 0 : \forall p \in \mathcal{P}$ **then**
5: **return** No problematic states: try increasing N.
6: $Q^* = argmax_{q \in Q_{check}} \mathcal{P}(q)$
7: $Y_T = \{y \in \mathcal{Y} \ : \ y \in \mathcal{L}(q) \ \forall \ q \in Q^*\}$
8: $Y_F = \{y \in \mathcal{Y} \ : \ y \notin \mathcal{L}(q) \ \forall \ q \in Q^*\}$
9: **if** $Y_T = \emptyset$ **and** $Y_F = \emptyset$ **then**
10: $y^* = argmax_{y \in \mathcal{Y}}(|\{q \in Q^* : y \in \mathcal{L}(q)\}|)$
11: $y_* = argmax_{y \in \mathcal{Y}}(|\{q \in Q^* : y \notin \mathcal{L}(q)\}|)$
12: **if** $|\{q \in Q^* : y_* \notin \mathcal{L}(q)\}| > |\{q \in Q^* : y^* \in \mathcal{L}(q)\}|$ **then**
13: $Q^* \leftarrow \{q \in Q^* : y_* \notin \mathcal{L}(q)\}$
14: **else**
15: $Q^* \leftarrow \{q \in Q^* : y^* \in \mathcal{L}(q)\}$
16: $Y_T = \{y \in \mathcal{Y} \ : \ y \in \mathcal{L}(q) \ \forall \ q \in Q^*\}$
17: $Y_F = \{y \in \mathcal{Y} \ : \ y \notin \mathcal{L}(q) \ \forall \ q \in Q^*\}$
18: $X_T = \{x \in \mathcal{X} \ : \ x \in \mathcal{L}(q) \ \forall \ q \in Q^*\}$
19: $X_F = \{x \in \mathcal{X} \ : \ x \notin \mathcal{L}(q) \ \forall \ q \in Q^*\}$
20: **if** $X_T \neq \emptyset$ **or** $X_F \neq \emptyset$ **then**
21: $\Phi = \square((\bigwedge_{x \in X_T} \bigcirc x \wedge \bigwedge_{x \in X_F} \neg \bigcirc x) \rightarrow \neg(\bigwedge_{y \in Y_T} \bigcirc y \wedge \bigwedge_{y \in Y_F} \neg \bigcirc y))$
22: **else**
23: $\Phi = \square(\neg(\bigwedge_{y \in Y_T} \bigcirc y \wedge \bigwedge_{y \in Y_F} \neg \bigcirc y))$
24: **return** $\Phi$

algorithm, $\Phi$, is one of the two LTL formulas defined on lines 21 and 23. In either case, $\Phi$ is a safety formula requiring that the robot avoid the configuration of robot propositions that is identified as consistent across the set of states $Q^*$. The formula defined on line 21 is the more specific case, where the configuration of consistent environment propositions is used as a precondition for the configuration that the robot is required to avoid, while the formula defined on line 23 is the more general case, when no specific precondition is found.

This algorithm describes a preliminary approach for semi-automated specification revision, and it is important to note that the suggested revisions are not guaranteed to improve the behavior of the robot once included in the specification, and may even result in an unsynthesizable specification (i.e. a task description that the robot can not accomplish). While the suggested revisions cannot guarantee a better performing controller, the process described in Algorithm 2 ensures that the suggested revision is constructed to force the robot to avoid the single behavior that is most likely to lead to a violation of the given specification. Furthermore, this approach is meant to provide feedback to the user, who can then perform an analysis of the new controller (via Algorithm 1) to determine whether or not to include the suggested revision. By including the user in the revision process, this approach also ensures that any changes to the specification preserve the intent of the original specification.

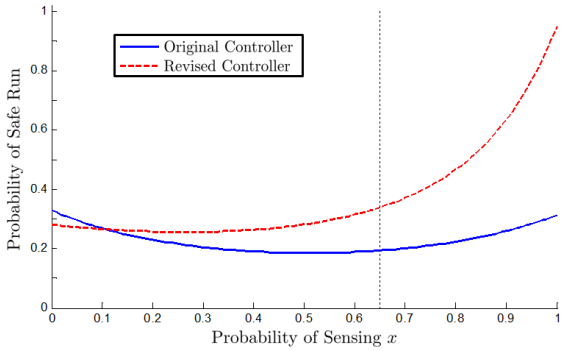



Fig. 6. Comparison of safety results for Example 1, of the original and revised controllers, over varying probabilities of sensing $x$. The analysis was performed for a time-bound of 50 discrete transitions.

## V. EXAMPLES

### A. Basic Example

**Scenario and Probabilities:** Consider the scenario presented in Example 1, where the robot must navigate the partitioned workspace shown in Figure 2 while avoiding $R_1$ when its sensor $x$ is true, and avoiding $R_4$ when $x$ is false. The probabilistic behavior of the environment and the error in the robot's actuation are described in Section IV-A.

**Analysis and Revision:** The probabilistic system model was analyzed with respect to the following formula, which is a combination of the safety specifications in the original task, and represents the overall safety of the system.

$$\square((\bigcirc x \rightarrow \bigcirc \neg R_1) \wedge (\bigcirc \neg x \rightarrow \bigcirc \neg R_4))$$

Additionally, the approach described in Section IV-B was applied to this example to automatically obtain a revision for the controller specification. The returned revision to the original specification, $\square(\bigcirc x \rightarrow \neg \bigcirc R_4)$, requires that the robot avoid region $R_4$ even while $x$ is true. Figure 6 shows a comparison of the probability that the robot will safely complete 50 discrete state transitions, for both the original and the revised controller. The figure shows the probability that the robot remains safe, for a range of values on the probability that the robot senses $x$; the revision was found for $p(x) = 0.65$, indicated by the vertical line.

As expected, the probability that the robot remains safe is higher for the revised controller than the original controller, for high values of the sensor frequency. For the specified system model (where the probability that $x$ is true is 0.65 in any time step), this plot shows that including the suggested revision in the synthesis of the controller results in a significant improvement in the performance of the robot with regards to the original safety specifications.

At low probabilities of sensing $x$, the most likely cause of unsafe behavior results from the robot unintentionally remaining in $R_1$ while the value of $x$ becomes false, and the revised controller performs slightly worse than the original controller. Indeed, had the probability of $x$ used to get the revision been low (e.g. 0.35), the states most likely to cause undesirable behavior would be those in which the robot is in $R_1$, and the suggested revision would have instead been to avoid region $R_1$ while $x$ is false: $\square(\bigcirc \neg x \rightarrow \neg \bigcirc R_1)$.

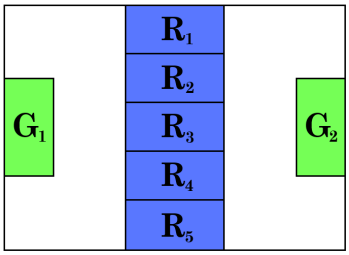



Fig. 7. Partitioned workspace for Example 2 (adversary avoidance).

The resulting controller would show analogous results, but with improvement occurring at lower sensor probabilities (essentially, mirrored about $P(x') = 0.5$).

### B. Adversary Avoidance Example

**Example 2:** For this example, the robot is tasked with moving between regions $G_1$ and $G_2$, in Figure 7, while avoiding an adversary that patrols regions $R_1 - R_5$. because of the workspace topology, this task requires the robot to traverse through regions $R_1 - R_5$, but do so in a manner that it avoids being in the same region as the adversary.

The robot has the ability to sense the current location of the adversary, denoted by the set of mutually exclusive propositions $A_{R_1} - A_{R_5}$, and can move freely between adjacent regions. The synthesized controller for this example has 60 states, and proceeds across the adversarial regions in the following manner.

- If the adversary is in regions $R_2 - R_5$ (i.e. one of $A_{R_2} - A_{R_5}$ is true), then the robot proceeds through $R_1$.
- If the adversary is in region $R_1$ (i.e. $A_{R_1}$ is true), then the robot proceeds through $R_4$.

The motion of both the robot and adversary are modeled as instantaneous transitions by each, where the robot reacts to the motion of the adversary. If the robot's motion is perfectly reliable, it will always safely pass through the adversarial regions and accomplish its assigned task. If, however, the motion of the robot is imperfect, it may end up in the same region as the adversary, violating the task specification.

**Probabilities:** For this example, the motion of the adversary is modeled probabilistically, such that, at each discrete time step, it has a probability of 0.5 of remaining in its current region; if it does not remain in the same region, it moves to an adjacent region with uniform probability among the adjacent regions. That is, if it is in region $R_1$ (resp. $R_5$), the adversary has a 0.5 probability of staying where it is and a 0.5 probability of transitioning to $R_2$ (resp. $R_4$). If the adversary is in $R_2$, $R_3$, or $R_4$, it has a 0.5 probability of staying in the same region, and a 0.25 probability of moving to each of the adjacent regions.

The motion of the robot is also modeled probabilistically, allowing for error in the robot's actuation. Its motion is modeled such that whenever it attempts to move to a new region, it has a 0.9 probability of correctly doing so, and a 0.1 probability of erroneously remaining in the same region (i.e. getting stuck). The robot is assumed to be able to remain stopped without error, so if it desires to remain in the same region, it will do so with a probability of 1.0.

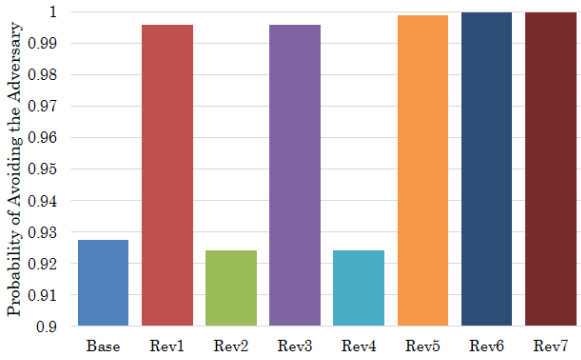



Fig. 8. Probability that the robot avoids the adversary, for the original and revised controllers, over a time-bound of 50 discrete transitions.

**Analysis and Revision:** For this example, a series of revisions were found using the approach outlined in Section IV-B. After each revision was added to the specification, a new controller was synthesized and modeled, and a revision was found for the new controller. The controller revisions were generated for the undesirable behavior where the robot ends up in the same region as the adversary.

$$\Psi = (A_{R_1} \wedge R_1) \vee (A_{R_2} \wedge R_2) \vee (A_{R_3} \wedge R_3) \vee (A_{R_4} \wedge R_4) \vee (A_{R_5} \wedge R_5)$$

A series of 7 revisions were found and added to the original controller specification. These revisions are given below, in order. It should be noted that, while revisions 1, 3, and 5 were found with a look-ahead of $N = 1$ steps, revisions 2, 4, and 6 required looking ahead $N = 2$ steps and revision 7 required a look ahead of $N = 3$ steps.

Rev1. $\square(\bigcirc A_{R_2} \rightarrow \neg \bigcirc R_1)$
Rev2. $\square(\bigcirc A_{R_3} \rightarrow \neg \bigcirc R_1)$
Rev3. $\square(\bigcirc A_{R_3} \rightarrow \neg \bigcirc R_4)$
Rev4. $\square(\bigcirc A_{R_3} \rightarrow \neg \bigcirc R_5)$
Rev5. $\square(\bigcirc A_{R_3} \rightarrow \neg \bigcirc R_2)$
Rev6. $\square(\bigcirc A_{R_2} \rightarrow \neg \bigcirc R_4)$
Rev7. $\square(\bigcirc A_{R_1} \rightarrow \neg \bigcirc R_4)$

The resulting probability that the robot successfully avoids the adversary robot during the bounded execution of 50 discrete transitions is shown in Figure 8. These results show that, while the suggested revisions do not always improve the performance of the robot (e.g. revisions 2 and 4), they can result in a dramatic improvement over the original (base) controller (e.g. revisions 1, 3, 5, 6, and 7).

As a result of the first revision, the robot chooses to avoid $R_1$ whenever the adversary is in $R_2$, and instead elects to go through $R_4$. This puts the robot further away from the adversary when it traverses the middle regions, so it is less likely to get stuck long enough for the adversary to enter the same region as it. The result is a marked improvement in performance. The second revision, on the other hand, requires that the robot avoid $R_1$ when the adversary is in $R_3$; as a result, the revised controller elects to go through $R_4$ in such a case, placing the robot closer to the adversary, and making it more likely to fail.

The next three revisions (Rev3-5) continue to restrict the behavior of the robot when the adversary is in $R_3$ to the

point where, in the revised controller that includes revisions 1-5, the robot elects to stop and wait until the adversary is in a region that is further from the one the robot is traversing.

After the inclusion of all 7 revisions, the behavior of the robot is dramatically different from the original controller: whenever the adversary is in regions $R_4$ or $R_5$ the robot still elects to traverse through region $R_1$, but whenever the adversary is in regions $R_1$ or $R_2$ the robot traverses through region $R_5$, and whenever the adversary is in $R_3$ the robot waits for it to move to a more easily avoidable region. As a result, it stays consistently further away from the adversary than for the behavior created by the original controller.

## VI. Conclusion

This paper presents a novel approach for analyzing the probabilistic behavior of correct-by-construction robot controllers when influenced by uncertainty in the outcomes of the robot's actuation. The presented algorithm composes probabilistic models of environmental state changes and the actuation uncertainty with the synthesized controller to obtain a DTMC for the complete probabilistic system. This DTMC is then analyzed with respect to a set of desired task specifications (given as LTL formulas) using probabilistic model checking techniques.

Additionally, this paper presents a novel technique for providing feedback to the user in the form of possible revisions to the original specification. By analyzing the system to find the set of states most likely to yield an undesirable behavior (such as violating the original safety specifications), the presented approach can suggest an LTL specification aimed towards avoiding such states and reducing the probability of exhibiting the undesired behavior.

Computationally, the most restrictive step in the presented approach is the model checking of the constructed probabilistic model. LTL model checking is exponential in the size of the formula, and polynomial in the size of the model [25].

The presented techniques, along with methods for synthesizing robot controllers from temporal logic task specifications, provides a method for easing the time and difficulty burdens inherent in programming and testing robot controllers for complex tasks. Future work with the presented approach will include combining it with the method presented in [18] and [19] for the assessment of a synthesized controller with uncertainty in both the sensing and actuation, as well as the inclusion of error-recovery capabilities. In addition, the authors will explore the extension of the presented approach to fully-automated specification revision, and the synthesis of controllers that are robust to some modeled uncertainty.

## References


[1] S. Loizou and K. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on ltl specifications," in *IEEE Conference on Decision and Control*. IEEE, 2004, pp. 153–158 Vol.1.

[2] G. Fainekos, H. Kress-Gazit, and G. Pappas, "Hybrid controllers for path planning: A temporal logic approach," *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 4885–4890, 2006.

[3] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from ltl specifications," in *Hybrid Systems: Computation and Control*, vol. 3927, 2006, pp. 333–347.

[4] P. Tabuada and G. Pappas, "Linear time logic control of discrete-time linear systems," *Automatic Control, IEEE Transactions on*, vol. 51, no. 12, pp. 1862 –1877, dec 2006.

[5] D. Conner, H. Kress-Gazit, H. Choset, and A. Rizzi, "Valet parking without a valet," *Proc. of IEEE/RSJ*, pp. 572–577, oct 2007.

[6] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic $\mu$-calculus specifications," in *IEEE Conference on Decision and Control*, 2009, pp. 2222–2229.

[7] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-logic-based reactive mission and motion planning," *Robotics, IEEE Transactions on*, vol. 25, no. 6, pp. 1370–1381, dec 2009.

[8] M. Lahijanian, S. Andersson, and C. Belta, "A probabilistic approach for control of a stochastic system from ltl specifications," in *IEEE Conference on Decision and Control*, Shanghai, P.R. China, 2009, pp. 2236–2241.

[9] A. Bhatia, L. Kavraki, and M. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 2689–2696.

[10] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, Taipei, Taiwan, October 2010, pp. 1988 – 1993.

[11] M. Lahijanian, J. Wasniewski, S. Andersson, and C. Belta, "Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, Anchorage, AK, may 2010, pp. 3227–3232.

[12] T. Wongpiromsarn, U. Topcu, and R. Murray, "Automatic synthesis of robust embedded control software," *AAAI Spring Symposium on Embedded Reasoning: Intelligence in Embedded Systems*, 2010.

[13] B. Johnson, F. Havlak, M. Campbell, and H. Kress-Gazit, "Execution and analysis of high-level tasks with dynamic obstacle anticipation," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[14] J. J. Amend, E. Brown, N. Rodenberg, H. Jaeger, and H. Lipson, "A positive pressure universal gripper based on the jamming of granular material," *IEEE Transactions on Robotics*, vol. 28, pp. 341–350, 2012.

[15] X. C. Ding, J. Wang, M. Lahijanian, I. C. Paschalidis, and C. A. Belta, "Temporal logic motion control using actor-critic methods," in *2012 IEEE International Conference on Robotics and Automation*. Ieee, may 2012, pp. 4687–4692.

[16] A. I. Medina Ayala, S. B. Andersson, and C. Belta, "Probabilistic control from time-bounded temporal logic specifications in dynamic environments," in *2012 IEEE International Conference on Robotics and Automation*. Ieee, may 2012, pp. 4705–4710.

[17] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *2012 IEEE International Conference on Robotics and Automation*. Ieee, may 2012, pp. 5163–5170.

[18] B. Johnson and H. Kress-Gazit, "Probabilistic analysis of correctness of high-level robot behavior with sensor error," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[19] ——, "Probabilistic guarantees for high-level robot behavior in the presence of sensor error," *Autonomous Robots*, vol. 33, pp. 309–321, 2012, 10.1007/s10514-012-9301-4. [Online]. Available: http://dx.doi.org/10.1007/s10514-012-9301-4

[20] K. Kim, G. Fainekos, and S. Sankaranarayanan, "On the revision problem of specification automata," in *Proceedings of the IEEE Conference on Robotics and Automation*, 2012.

[21] G. E. Fainekos, "Revising temporal logic specifications for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 40–45.

[22] E. Emerson, "Temporal and modal logic," *Handbook of Theoretical Computer Science*, pp. 995–1072, 1990.

[23] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," *Verification, Model Checking, and Abstract*, pp. 364–380, 2006.

[24] M. Kwiatkowska, G. Norman, and P. D., "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, vol. 6806. Springer, 2011, pp. 585–591.

[25] C. Courcoubetis and M. Yannakakis, "The complexity of probabilistic verification," *Journal of ACM*, vol. 42, no. 4, pp. 587–907, Jul 1995.