Probabilistic Analysis of Correctness of High-Level Robot Behavior with Sensor Error

Benjamin Johnson, Hadas Kress-Gazit
Sibley School of Mechanical and Aerospace Engineering, Cornell University
Ithaca, NY 14853
{blj39, hadaskg}@cornell.edu

***Abstract*—This paper presents a method for reasoning about the effects of sensor error on high-level robot behavior. We consider robot controllers that are synthesized from a set of high-level, temporal logic task specifications, such that the resulting robot behavior is guaranteed to satisfy these specifications when assuming perfect sensors and actuators. We relax the assumption of perfect sensing, and calculate the probability with which the controller satisfies a set of temporal logic specifications. We consider parametric representations, where the satisfaction probability is found as a function of the model parameters, and numerical representations, allowing for the analysis of large examples. We illustrate our approach with three examples of varying size that provide insight into unintuitive effects of sensor error that can inform the specification design process.**

I. Introduction

The creation of robot controllers for complex tasks is an arduous and error-prone process, requiring the iterative design and testing of large, complex controllers. In many cases, these controllers extend beyond single continuous controllers into large, hybrid controllers that allow for discrete switching among a set of individual, continuous controllers. Recently, researchers have developed methods for automatically synthesizing complex, hybrid controllers from high-level task specifications in a manner that provides guarantees about the behavior of the robot [1, 2, 5, 9, 10, 15, 17, 18, 20].

These methods cover a variety of capabilities and applications. The approach presented in [8], for example, facilitates the creation of a non-reactive controller from temporal logic specifications. Conversely, other approaches such as [11, 20] focus on creating controllers that react to the system's perception of environmental inputs. Another example of differing approaches is that of [4, 9] when compared to [1, 8]; the former two methods use feedback controllers, while the latter two use sampling-based methods to allow for motion planning involving complex dynamics and environments. The authors of [19] present a a receding horizon framework for reducing computational complexity, at the cost of completeness.

Previous work facilitated the creation of controllers that were guaranteed to satisfy their underlying specifications, but such guarantees were based on the assumption of perfect sensing and actuation. In this paper, we relax the assumption of perfect sensing, and use probabilistic analysis to investigate the correctness of the high-level behavior of the controllers.

Related work includes [13], in which the authors control linear, stochastic systems with temporal logic specifications by constructing a Markov Decision Process and using model checking algorithms to find an execution satisfying the specifications. Building on the work presented in [9], they then define a sequence of controllers to maximize the probability of following the determined execution.

Similarly, in [14] the authors propose an algorithm for finding a desirable control strategy for motion planning in the presence of noise. By treating the outcome of the low-level motion controllers in a probabilistic fashion, the authors were able to find, under known operating conditions, the control strategy most likely to satisfy the given specifications.

In this paper, we assume perfect actuation, and analyze the effect that sensor error has on the behavior of the robot. Specifically, we use probabilistic model-checking techniques to analyze the effects of sensor error on the satisfaction of behavioral specifications defined as temporal logic formulas. The analysis can then be used to adjust the specifications and change the controller to improve the robot's performance. This approach for analyzing the performance of the controller under sensor error is, to the best of our knowledge, novel.

We consider a parametric model checking algorithm, based on the algorithm described in [7]. This algorithm calculates the probability with which the system model satisfies a set of temporal logic formulas, as a function of the model parameters. Models with large state-spaces and numbers of transitions can become intractable for our current implementation of the algorithm; for these cases, we use the PRISM [12] probabilistic model checking tool with the same system model and specifications to find a numerical (rather than parametric) probability for the satisfaction of the specifications.

The paper is organized as follows. Section II briefly describes the background information for the problem, and Section III formally defines the problem statement. Section IV describes our approach and the algorithms we use to analyze the effect of sensor error. Section V presents three illustrative examples, and the paper concludes in Section VI.

II. Preliminaries

A. LTL Syntax and Semantics

The syntax of Linear Temporal Logic (LTL) is defined using a set of atomic propositions ($\pi \in AP$), the set of boolean operators ("not": $\neg$, "and": $\wedge$), and the set of temporal operators ("next": $\bigcirc$, "until": $\mathcal{U}$). The syntax for the language is then defined recursively as follows.

$$\phi ::= true \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi$$

This work was supported by NSF CNS-0931686.
We thank Cameron Finucane and Vasumathi Raman for all their help.

We can define the additional boolean operators $\vee$ ("or"), $\rightarrow$ ("implies") and $\leftrightarrow$ ("if and only if") using only $\neg$ and $\wedge$. Likewise, $\bigcirc$ and $\mathcal{U}$ can be used to define the operators $\Diamond$ ("eventually") and $\Box$ ("always").

To define the semantics of LTL, let ω represent an infinite sequence of states in a transition system (Kripke structure [3]), where each state is labeled with the set of truth assignments for the atomic propositions $\pi \in AP$. Let state ω_i in this sequence represent the truth assignments to π at the position i. Intuitively, $\bigcirc\phi$ is true if the formula ϕ is true in the next step (at state ω_{i+1}). The formula $\Diamond\phi = true\,\mathcal{U}\phi$ is true if ϕ holds in at least one state in ω, while $\Box\phi = \neg\Diamond\neg\phi$ is true if ϕ holds for all states in ω. For a formal definition, the reader is referred to [3].

B. Restricted form of LTL

We consider a restricted class of LTL formulae [16] of the form $\varphi = (\varphi_e \rightarrow \varphi_s)$ where φ_e represents the assumptions about the behavior of the environment and φ_s defines the desired behavior of the robot. We assign φ_e and φ_s the following structure.

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \;; \quad \varphi_s = \varphi_i^s \wedge \boldsymbol{\varphi_t^s} \wedge \boldsymbol{\varphi_g^s}$$

In the above formulas, φ_t^s represents the **safety** requirements for the robot, which require certain behaviors to always be satisfied, and consists of a conjunction of formulas of the form $\Box B_i$, where each B_i is a Boolean formula over $AP \cup \bigcirc AP$. φ_g^s represents the specified **liveness** for the robot, which requires that some desirable behavior occurs infinitely often, and consists of a conjunction of formulas of the form $\Box\Diamond B_j$, where each B_j is a boolean formula over AP. Because the focus of this paper is on φ_t^s and φ_g^s, the other formulas are not explicitly defined here. For more information, see [16].

C. Labeled Transition Systems (LTS)

A Labeled Transition System (LTS) is a tuple $\mathcal{A} = \{S, S_0, \Sigma, \delta, \Pi, L\}$ where S is a set of states, S_0 is the set of initial states and Σ is the input alphabet. The transition relation $\delta : S \times 2^{\Sigma} \times S$ defines possible labeled transitions (s_i, σ_{ij}, s_j) for $\sigma_{ij} \subseteq \Sigma$. The labeling function $L : S \rightarrow 2^{\Pi}$ labels each state with a set of symbols belonging to the set Π.

D. Discrete Time Markov Chains

A Discrete Time Markov Chain (DTMC) is defined as a tuple $\mathcal{D} = \{Q, Q_0, \Delta, \Pi, \mathcal{L}\}$, where Q defines a finite set of states with the set of initial states Q_0. The transition function $\Delta : Q \times p \times Q$ defines the probability with which state $q_i \in Q$ transitions to state $q_j \in Q$. The labeling function $\mathcal{L} : Q \rightarrow 2^{\Pi}$ labels each state with a set of symbols belonging to the set Π.

III. Problem Statement

We consider automatically-synthesized robot controllers (e.g. [11]) for which the robot is guaranteed to achieve a given high-level specification, if perfect sensing and actuation are assumed. To analyze the behavior of the controller under sensor error, we must define models for the robot controller, the environment behavior and the sensor error.

The robot controller, *R*, is defined as an LTS where the input set Σ is a set of binary propositions $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_m\}$ that represent the information that the robot can attain about the environment via its sensors. The set Π, used to label the states, is a set of binary propositions for the robot, $Y = \{r_1, \ldots, r_n, a_1, \ldots, a_k\}$. The set of propositions $\{r_1, \ldots, r_n\}$ represents the location of the robot in a partitioned environment, where proposition r_i is true if and only if the robot is in region *i*. The set $\{a_1, \ldots, a_k\}$ is the set of propositions that can be activated by the robot, where proposition a_j is true if and only if the robot performs action *j*.

The environment behavior, E, is captured by the set of transition probabilities $P(X'|X, Y)$ defining the probability of the next environment proposition values $X' = \{x_1', \ldots, x_m'\}$, given the current environment and robot values, X and Y respectively. This formulation allows us, if applicable, to place assumptions on the behavior of the environment (by setting transition probabilities to 0 or 1). Such assumptions enable us to generate controllers that do not need to satisfy the desired specification for any arbitrary environment, but rather only for a restricted set of allowable environments. The environmental assumptions, which restrict the values of X, are captured in the temporal logic formula φ_e.

The sensor behavior, $\bar{E}$, is modeled with the set of probabilities $P(\bar{X}'|X', \bar{X}, Y)$ defining the values of the next sensor propositions $\bar{X}' = \{\bar{x}_1', \ldots, \bar{x}_m'\}$ given the next environment values, X', and the current proposition values of the sensors and the robot, $\bar{X}$ and Y respectively.

The final piece we define is the system specification, Φ, for the robot. It is expressed as a set of LTL formulas $\{\phi_1, \ldots, \phi_l\}$ that are part of the temporal logic formula φ_s [11].

Previous work has shown that, given assumptions on the environment behavior and specifications for the robot behavior, a controller can be synthesized that is guaranteed to behave correctly in all admissible environments (i.e. $E \parallel R \models \phi\,, \forall\phi \in \Phi$) [11]. This guarantee, however, can only be made when perfect sensors and actuators are assumed (i.e. $E = \bar{E}$).

The focus of this paper is on the following problem.

Problem: Given a model of the robot controller *R*, and models of the environment E and sensors $\bar{E}$, determine the probability that the synthesized robot controller will satisfy a set of high level specifications when the sensor outputs contain false positives and false negatives. That is, given that $E \neq \bar{E}$, find $p(E \parallel \bar{R} \models \phi)\ \forall\phi \in \Phi$ where $\bar{R} = \bar{E} \parallel R$.

IV. Approach

To analyze the behavior of a synthesized controller (R) under sensor error, we compose it with models of the environment (E) and sensors ($\bar{E}$), and use probabilistic model checking techniques on the resulting DTMC to assess the performance of the controller with respect to a set of LTL formulas. Figure 1 is a graphical overview of this process.

In the following, we describe an algorithm for creating a DTMC representing the composition of the controller with the environment and sensor model and discuss two approaches to analyzing the robot's behavior. The primary method we use is a parametric model checking algorithm, based on the

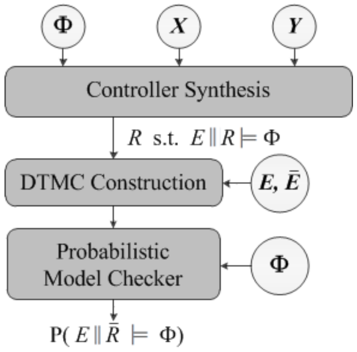

Fig. 1. A flow chart of the approach presented in this paper for analyzing the correctness of a controller that is generated from high-level task specifications.

work described in [7]. This algorithm returns the probabilities as rational functions over a set of variables, in our case, representing the environmental event frequency and sensor error rates. Thus, we are able to provide information not only on the probability a specification is satisfied given the sensor error but also on the sensor error bounds that are required for a given desired specification probability.

In addition to the parametric model checking algorithm, we discuss the use of the off-the-shelf probabilistic model checker PRISM [12] to evaluate probabilities when numeric model parameters are given (non-parametric).

A. Building the DTMC

Analysis of a synthesized controller using the probabilistic model checking algorithm requires first that a model of the complete system be composed from the known models of the environment, sensors, and robot automaton. This process is described in Algorithm 1. The inputs are the LTS for the robot controller, the set of atomic propositions $AP = X \cup \bar{X} \cup Y$, the environment transition probabilities $P(X'|X,Y)$ and the sensor transition probabilities $P(\bar{X}'|X',\bar{X},Y)$.

Given these inputs, the algorithm defines a set of probabilistic states for each deterministic state $s_i \in S$, such that the new probabilistic state is labeled with the combination of the environment propositions $X^j = (x_1,\ldots,x_n) \in 2^X$, the labels on the transitions into the deterministic state σ_{*i} (sensor propositions), and the labels of the deterministic robot state (lines 3-6). Any probabilistic state created from $s_i \in S_0$ is then added to the set of initial states for the complete system (lines 7-8). The probabilistic transition function $\Delta : Q \times \mathcal{F}_V \times Q$ is then defined for all pairs of states (q_{ij}, q_{kl}) such that the underlying deterministic pair (s_i, s_k) is in the transition function δ, and the probability function $f_{ijkl} \in \mathcal{F}_V$ (the rational function representing the probability of transitioning from state q_{ij} to q_{kl}) is non-zero (lines 9-13). The resulting DTMC $\mathcal{D} = \{Q, Q_0, \Delta, AP, \mathcal{L}\}$ can then be used to analyze the performance of the controller.

Additionally, we can define a bounded system by unfolding the DTMC obtained from Algorithm 1 over the range of time $0 \le t \le T$ where T is the desired time bound. Intuitively, this process can be seen as building a tree such that root nodes are defined as the set of initial states Q_0, and each new level (referring to the next time step) is defined as the union of the sets of successors for each state in the preceding level of

Algorithm 1 Define Probabilistic DTMC

```
1: procedure ProbDTMC($\mathcal{A} = \{S, S_0, \bar{X}, \delta, Y, L\}$,
   $AP = X \cup \bar{X} \cup Y, P(X'|X,Y), P(\bar{X}'|X',\bar{X},Y)$)
2:     $Q = \emptyset,\ Q_0 = \emptyset$
3:     for $s_i \in S$ do
4:         for $X^j \in 2^X$ do
5:             $Q = Q \cup q_{ij}$
6:             $\mathcal{L}(q_{ij}) = X^j \cup \sigma_{*i} \cup L(s_i) \mid \sigma_{*i} \subseteq \Sigma$
7:             if $s_i \in S_0$ then
8:                 $Q_0 = Q_0 \cup q_{ij}$
9:     for $(s_i, s_k) \in \delta$ do
10:        for $(q_{ij}, q_{kl}) \in Q \times Q$ do
11:            $f_{ijkl} = P(X^l|X^j,Y^i) \times P(\bar{X}^l|X^l,\bar{X}^j,Y^i)$
12:            if $f_{ijkl} \neq 0$ then
13:                $\Delta = \Delta \cup (q_{ij}, f_{ijkl}, q_{kl})$
14:    return $\mathcal{D} = \{Q, Q_0, \Delta, AP, \mathcal{L}\}$
```

the tree. This process is repeated for each time step, and the resulting DTMC is the bounded system.

B. Parametric Model Checking

Given a probabilistic system model as defined previously, we find the function representing the probability with which it satisfies a specification of the form $\phi = \Diamond\varphi$ (resp. $\phi = \Box\varphi = \neg\Diamond\neg\varphi$). For the latter case, we find the probability of always satisfying φ by calculating the probability of eventually satisfying $\neg\varphi$, and subtracting it from 1 (giving us the probability of not satisfying $\neg\varphi$). This process is described in Algorithm 2, and is based on the algorithm in [7]. Intuitively, lines 2-5 of the algorithm find the set of states $B \subseteq Q$ that satisfy the formula φ (resp. $\neg\varphi$). From this set of target states, we use a minimum fixed-point operation to reduce the model to the set of states from which at least one target state is reachable (line 6). Line 7 then calls Algorithm 4, to find the probability of reaching the set of target states from an initial state. The function $\mathcal{P}$ (resp. $1 - \mathcal{P}$) represents the probability that the formula $\phi = \Diamond\varphi$ (resp. $\phi = \Box\varphi$) is satisfied by the DTMC.

Algorithm 2 Parametric Model Checking

```
1: procedure ParametricMC($\mathcal{D} = \{Q, Q_0, \Delta, AP, \mathcal{L}\}$,
   $\phi, X, Y$)
2:     if $\phi = \Box\varphi$ then
3:         $(B, \Delta) = TargetStates(Q, \Delta, \neg\varphi, \mathcal{L}, X, Y)$
4:     else
5:         $(B, \Delta) = TargetStates(Q, \Delta, \varphi, \mathcal{L}, X, Y)$
6:     $Q_{reach} = ReduceStates(Q, B, \Delta)$
7:     $\mathcal{P} = Eliminate(Q_{reach}, Q_0, B, \Delta)$
8:     if $\phi = \Box\varphi$ then
9:         $\mathcal{P} = 1 - \mathcal{P}$
10:    return $\mathcal{P}$
```

Algorithm 3 defines the process of finding the set of states that satisfy a given formula ϕ. To do this, we look at two distinct forms of ϕ. In the first case, ϕ is a boolean formula over the values of X and Y in a single state (in this paper, we examine the performance of the controller with respect to the environment, and do not use $\bar{X}$ in any of the formulas). For such a formula, the set of target states can be found by looping through all of the states in the model. At each state, we map each atomic proposition in the formula to a T/F value, based on the state label (line 6). If the resulting formula evaluates to true, we add that state to the set of target states and remove all transitions out of it, treating it as a sink (lines 7-9).

Algorithm 3 Find Target States

```
1:  procedure TargetStates($Q, \Delta, \phi, \mathcal{L}, X, Y$)
2:      if $\phi$ is a formula over $\{X, Y\}$ then
3:          $B = \emptyset$
4:          for $q_i \in Q$ do
5:              for $a \in \{X, Y\}$ do
6:                  $\phi[a \mapsto (a \in \mathcal{L}(q_i))]$
7:              if $eval(\phi)$==True then
8:                  $B = B \cup q_i$
9:                  $\Delta = \Delta \backslash (q_i, f_{ij}, q_j) \quad \forall\, q_j \in post(q_i)$
10: else if $\phi$ is a formula over $\{X, Y, \bigcirc X, \bigcirc Y\}$ then
11:         $Q = Q \cup q^*$
12:         $B = \{q^*\}$
13:         for $q_i \in Q \backslash q^*$ do
14:             $f_{i*} = 0$
15:             for $q_j \in post(q_i)$ do
16:                 for $a \in \{X, Y\}$ do
17:                     $\phi[a \mapsto (a \in \mathcal{L}(q_i))]$
18:                     $\phi[\bigcirc a \mapsto (a \in \mathcal{L}(q_j))]$
19:                 if $eval(\phi)$==True then
20:                     $f_{i*} = f_{i*} + f_{ij}$
21:                     $\Delta = \Delta \backslash (q_i, f_{ij}, q_j)$
22:             if $f_{i*} \neq 0$ then
23:                 $\Delta = \Delta \cup (q_i, f_{i*}, q^*)$
24:     return $B, \Delta$
```

The second case is that for which ϕ is a boolean formula over propositions in the current and next state (X, Y, $\bigcirc X$, $\bigcirc Y$). For this case, we create a single goal state q^* and loop though each pair of states (q_i, q_j) where q_j is a successor of q_i. We then map each atomic proposition to a T/F value based on the state q_j if it is under the scope of a "next" operator (line 18), or based on the state q_i otherwise (line 17). If the formula is true when evaluated over the pair, the transition from q_i to q_j is removed and the transition probability is added to the transition from q_i to q^* (lines 19-21).

Given the set of initial states Q_0, the set of goal states B, the set of states Q_{reach} that can reach B, and the transition function Δ, we can now eliminate all intermediate states from the model, until we are left with only the initial states, the goal states, and the associated transition probabilities. Algorithm 4 describes this process, and the calculation of the probability with which our model will reach a target state from a specified initial state. This algorithm follows the process described in [7], and can intuitively be described as removing an intermediate state q_i and accounting for the probability with which a predecessor $q_1 \in Pre(q_i)$ will transition through q_i to a successor $q_2 \in Post(q_i)$, for all pairs (q_1, q_2) (lines 5-9). Once all intermediate states are eliminated, the probability with which the system will reach a state satisfying ϕ is the sum of transition probabilities from the initial state to each goal state (adjusting for self-transitions) (line 10).

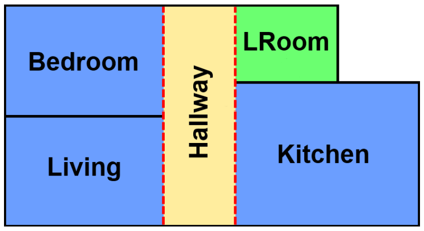

Fig. 2. Workspace for the example of a housekeeping robot. The robot can transition across boundaries marked with dashed red lines, but not those marked with solid black lines.

Algorithm 4 State Elimination

```
1:  procedure Eliminate($Q_{reach}, Q_0, B, \Delta$)
2:      for $q_0 \in Q_0$ do
3:          $Q_{elim} = Q_{reach} \backslash \{B, q_0\}$
4:          for $q_i \in Q_{elim}$ do
5:              for $(q_1, q_2) \in Pre(q_i) \times Post(q_i)$ do
6:                  $f_{12} = f_{12} + f_{1i} \frac{1}{1-f_{ii}} f_{i2}$
7:                  $(q_1, f_{12}, q_2) \in \Delta$
8:                  $\Delta = \Delta \backslash \{(q_1, f_{1i}, q_i), (q_i, f_{i2}, q_2)\}$
9:              $Q_{elim} = Q_{elim} \backslash q_i$
10:         $\mathcal{P}(q_0, B) = \frac{1}{1-f_{00}} \sum_{b \in B} f_{0b}$
11:     return $\mathcal{P}$
```

Our implementation of the preceding algorithms is done in Python, using the SympyCore toolbox to perform the symbolic manipulations. An alternate implementation of the algorithms may alleviate some inefficiencies and problems that occur when analyzing large models.

C. *PRISM Non-Parametric Model Checking*

In place of the Parametric Model Checking algorithm discussed above, we can also use the off-the-shelf model checking software PRISM, with the same system model and specifications, to analyze the performance of the controller. The PRISM software allows us to calculate the probability that our system model satisfies an LTL formula, but it does not use parametric algorithms, restricting the evaluation to specific numerical valuations of the system model. In particular, we use PRISM to evaluate models that are too large for our current implementation of the parametric algorithm.

V. Examples

A. *Example 1: Housekeeping Robot*

Scenario: A housekeeping robot moves about an environment that is divided into five separate regions, as shown in Figure 2. It is tasked with searching the *Bedroom*, *Kitchen*, and *Living* rooms, and performing the action *Clean* whenever it senses a *Mess*. Additionally, when the robot senses *LDone* (signaling that the laundry is done), it must return to the *LRoom* (laundry room), and perform the action *Fold*. Doing so marks the end of the robot's tasks.

Controller: The controller for this example was synthesized from a set of high-level task specifications, using the LTLMoP

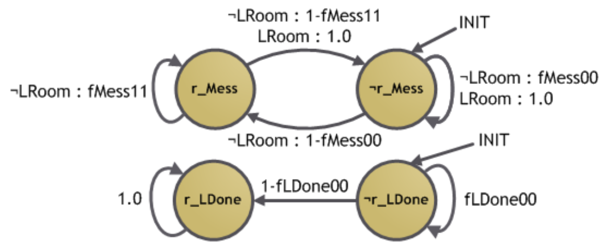

Fig. 3. Graphical representation of the environment finite state machine, with probabilities and necessary conditions given for each transition.

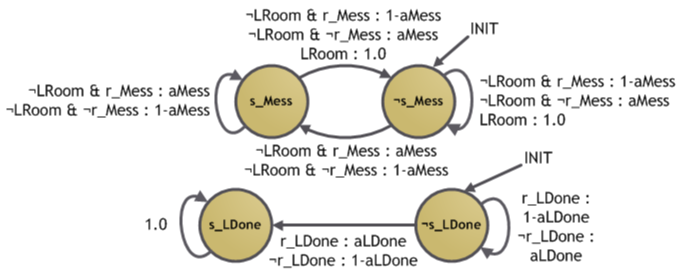

Fig. 4. Graphical representation of the sensor finite state machine, with probabilities and necessary conditions given for each transition.

[6] toolkit. An abridged set of the LTL specifications used to generate the controller is given below.

- $\square(\bigcirc Mess \leftrightarrow \bigcirc Clean)$
- $\square((\bigcirc LDone \wedge LRoom) \leftrightarrow \bigcirc Fold)$
- $\square\lozenge((\neg LDone \wedge \neg Mess) \rightarrow Bedroom)$
- $\square\lozenge((LDone \wedge \neg Mess) \rightarrow LRoom)$

Environment: The behavior of the environment propositions *r_Mess* and *r_LDone* (the prefix *r_* is used to indicate that it is a reflection of the "real" world), is captured by the automaton shown in Figure 3. The transition probabilities are defined with the parameters *fMess00*, *fMess11*, and *fLDone00*, which represent the probability that r_Mess remains false, r_Mess remains true, and r_LDone remains false, respectively. Each environment proposition is also restricted according to the environmental assumptions defined in φ_e, and listed below.

- $\square(LRoom \rightarrow \neg \bigcirc Mess)$ "No *Messes* in *LRoom*"
- $\square(LDone \rightarrow \bigcirc LDone)$ "Once true, *LDone* stays true"

Sensors: The sensor propositions, *s_Mess* and *s_LDone*, were modeled as shown in Figure 4. The prefix *s_* indicates that the variables refer to the "sensed" values of the corresponding environment propositions. The transition probabilities were defined by the parameters *aMess*, referring to the probability with which the *s_Mess* sensor correctly mimics the value of *r_Mess*, and *aLDone*, which represents the probability of a correct reading of *r_LDone*.

Analysis properties: For this example problem, our robot controller, R, consisted of 70 states. Composing the controller with the probabilistic environment and sensor models yielded a system with 280 states with probabilistic transitions. We analyzed the controller with respect to the following properties.

1) $\square(Clean \leftrightarrow r\_Mess)$ "*Clean* any and all *Messes*"
2) $\square(\bigcirc Fold \leftrightarrow (\bigcirc r\_LDone \wedge LRoom))$ "*Fold* if and only if *r_LDone* is true and you are in the *LRoom*"
3) $\lozenge(Bedroom)$ "Visit the *Bedroom*"

Using the parametric model checking algorithm described earlier, we find a symbolic formula for the probability of satisfying each property, as a function of the environment event frequency and sensor accuracy. For the following results, the values of *fMess00*, *fMess11*, and *fLDone00* were set at 0.75, 0.5, and 0.95, respectively. For the results in Figure 5, the value of *aMess* was held at 0.75.

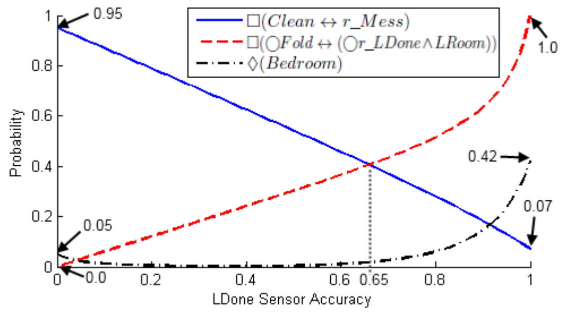

Fig. 5. Plot of the probability with which each of the properties is satisfied for a range of *LDone* sensor accuracies, evaluated at steps of 0.01.

Non-monotonic performance: Figure 5 shows the performance of the controller over a range of values for *aLDone*. We see that the probability of satisfying the third specification does not monotonically increase or decrease with respect to changes in the accuracy of *s_LDone*. Rather, it has a minimum at a sensor accuracy of approximately 0.38. This specification is satisfied when *s_LDone* remains false long enough that the robot reaches the *Bedroom*. This can occur when *r_LDone* immediately becomes true and *s_LDone* remains false, accounting for the small peak at very low sensor accuracies. Alternately, this can occur when *r_LDone* remains false and *s_LDone* correctly mimics it, accounting for the peak at high values of *aLDone*. At intermediate accuracies, *s_LDone* is more likely to become true (either correctly or incorrectly), causing the robot to go to *LRoom* before it reaches *Bedroom*.

Sensor effect on different specifications: Figure 5 also shows the effects of the *LDone* sensor accuracy on the first and second specifications. We see that improvements to the accuracy of the *LDone* sensor have a negative effect on the probability with which the robot satisfies the first property, which does not depend directly on *r_LDone*. This is because, due to the low probability of *r_LDone* becoming true, a lower sensor accuracy is more likely to result in a value of true for *s_LDone*, which causes the robot to return to *LRoom*, where the value of *Mess* is fixed as false, so no errors are made. The second specification is directly influenced by the *LDone* sensor, and improves as *s_LDone* becomes more accurate. A comparison of the two properties shows that a sensor accuracy of about 0.65 would be desired for equal performance of the controller with regards to each of the two specifications.

Picking sensor accuracies: If we wish to find the sensor accuracies needed for the robot to obtain a particular performance, it may be useful to look at the evaluation of a function over ranges of multiple sensors. Figure 6 shows a surface plot of the weighted average of all three specifications, with the added requirements that the first and second specifications maintain a probability greater than 0.25. The three specifications are weighted by 9, 2, and 5, respectively. This plot shows that, while the best performance results from high accuracies for both sensors, if the *Mess* sensor is inaccurate, a highly

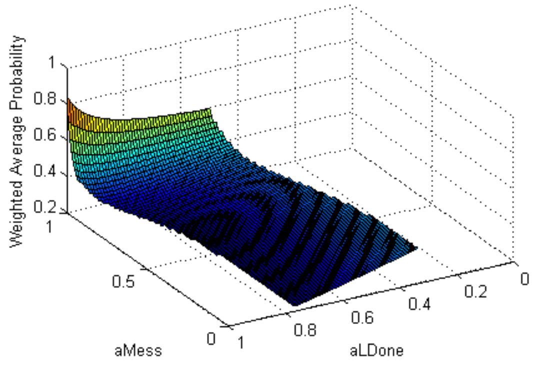

Fig. 6. Plot of the surface representing the weighted average probability of the three properties as a function of sensor accuracies for both the *Mess* and *LDone* sensors. Additional requirements that the first and second properties have a probability greater than 0.25 are imposed, constraining the surface.

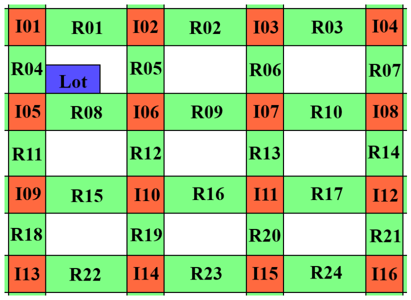

Fig. 7. Workspace for the example of a an autonomous Taxi. Roads are marked in green, with names beginning with *R*. Intersections are marked in red, with names beginning with *I*. The Lot region is marked in blue.

accurate *LDone* sensor may result in undesirable behavior (i.e. a point that is not part of the surface, due to the constraints).

B. Example 2: Taxi

Scenario: The second example we present is that of an autonomous Taxi. The robot operates in a 9-block grid of roads and intersections, with a single parking *Lot*. The discretized workspace, which has 41 different regions, is shown in Figure 7. The robot is tasked to continually visit each of the roads and, when it senses a *Person*, pick them up, activating the action *Passenger*. Once *Passenger* is activated, the taxi is required to take them to the *Lot* and drop them off. While doing this, the robot is required to *Stop* for any *RedLight*s it senses. The final sensor the robot has is used to detect when the lot is *Full*; once it is sensed, the robot *Park*s and the task is complete.

Controller: The deterministic controller that is synthesized from the temporal logic specifications has a total of 4,102 states. Composing this controller with the models of the environment and sensors results in a probabilistic model of the system with 24,612 states. Due to the large number of states and transitions in the model, the current implementation of the parametric model checking algorithm introduced in Section IV cannot efficiently calculate the probabilities with which the controller satisfies a given set of properties. As such, we use PRISM to analyze the performance of the controller.

Environment: For this example, the environment is restricted in several ways. Firstly, a *Person* will attempt to flag-down the taxi only while in *Road* regions. In those regions, we model the proposition with a 0.2 probability of being true. The second is that only the intersections contain stop lights and, therefore, *r_RedLight* can be true only at intersections. For this proposition, we model it such that it has a 0.5 chance of becoming true when previously false, and a 0.75 chance of remaining true (approximating the tendency of stoplights to remain red for a short duration). The final restriction we place on the environment is that *r_Full* has a 0.1 chance of becoming true, after which it remains true.

Sensors: In each of the states where the sensor values were not restricted by the assumptions placed on the environment, the sensors were modeled such that they correctly mimicked the corresponding environment proposition with a predefined probability. For the following analysis, each of the sensors that were not being varied were held fixed at accuracies of 0.9 and 0.75 for the *s_Full* and *s_Person* sensors, respectively. The *s_RedLight* sensor accuracy was held at a value of 0.85 in all of the intersections, with the exception of intersections I02, I08, and I10. For these three intersections, the sensor had a lower accuracy, held constant at 0.7.

Analysis properties: We analyze the performance of the controller with respect to the two properties listed below.

1) $\square(r\_RedLight \leftrightarrow Stop)$ "*Stop* at any and all *RedLights*"
2) $\square(Park \rightarrow \neg Passenger)$ "No *Passengers* when *Parking*"

Unrealizable specification: The second property, which requires that the robot never have a *Passenger* when it *Park*s, is interesting because it can not be included in the controller specification, as it is un-synthesizeable. We cannot guarantee that, even if the sensors are perfect, the controller will always satisfy this specification. If we analyze the specification under perfect sensor accuracies we find that the controller has a 0.625 probability of satisfying the specification. In contrast, analysis of the first property, which is part of the specification, shows 1.0 probability of satisfaction under perfect sensor accuracies.

Discontinuous case: Figure 8 shows the results of analyzing the properties over variations in the accuracy of *s_Full*. The second specification shown in this figure has a discontinuous data point at an accuracy of 0, where the sensor will always have the wrong reading for the *Full* proposition. As a result, *s_Full* will always be true if the first *r_Full* environment value is false (once the sensor turns true, it remains so). Alternately, if *r_Full* becomes true immediately, the sensor will continually be false. For the former case, the taxi will park before picking up a passenger, guaranteeing that the specification will be satisfied. For the latter case, the taxi will never park, again guaranteeing that the specification will be satisfied. At sensor accuracies greater than 0, the latter of the two cases will no longer hold, and the sensor will eventually become true, accounting for the discontinuity.

Adjusting the controller: Recognizing that the three intersections with lower *RedLight* sensor accuracy may adversely affect the performance of the robot, we can adjust the controller by adding the specification $\square\neg(\bigcirc I02 \vee \bigcirc I08 \vee \bigcirc I10)$, which requires that the robot always avoid these three regions. Analyzing this new controller under the default model values yields a 0.7004 probability of satisfying the first specification

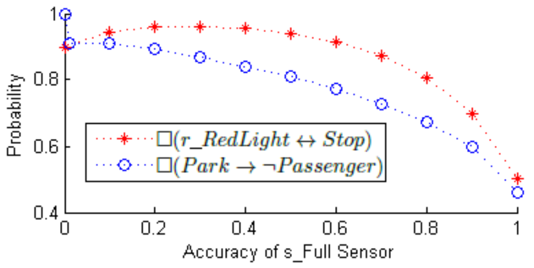

Fig. 8. Plot of the performance of the autonomous Taxi, over a range of different sensor accuracies for the *Full* sensor.

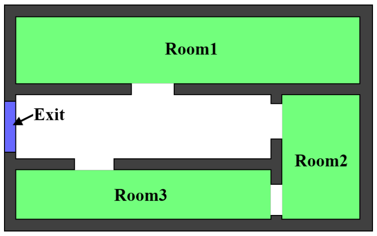

Fig. 9. Discretized workspace for the search and rescue example problem.

(as compared to the probability of 0.6986 for the original controller) and a 0.5962 probability of satisfying the second specification (as compared to 0.5964). We can see from these results that the bad intersections had negligible effect on the performance of the controller, due to the large number of intersections and the relatively small impact of the poor accuracies in just those three. For a smaller environment, or one where the bad sensors had more impact, such an adjustment may result in significant differences in the performance of the controller.

C. Example 3: Search and Rescue

Scenario: The final example we present is that of a Search and Rescue robot. We consider a small workspace, with 3 rooms, a hallway, and a designated exit point, as shown in Figure 9. The robot is tasked with searching the area for people and, when it senses a *Person*, to lead them to the *Exit*. While the robot does this, it is required to do the action *Clear* when it senses *Debris*, and to *Extinguish* when it senses a *Fire*. When the robot senses *DistressSignal*, however, it is required to ignore any *Debris* or *Fire* and urgently search for a *Person*. Additionally, if the robot is leading a *Person* to the *Exit* and it senses an *Alarm*, it is required to stop moving and wait for the *Alarm* to turn off.

Controller: The resulting controller for this specification consists of 164 deterministic states. Because of the large number of sensors in this example, the probabilistic system expands to 1640 distinct states. Due to the size of the example, particularly the large number of transitions, we use PRISM to analyze the performance of the controller.

Time bound: Unlike the previous two examples, the controller for this examples does not have an inherent ending point (such as *LDone* or *Full* sensors). As such, the behavior is infinite, and unbounded performance proves uninteresting. For the results presented below, the evaluation of the controller was restricted to a time bound of 30 discrete steps.

Environment: The environment proposition *r_Person* was modeled with a 0.25 chance of becoming true, and once it was true, it remained true until the robot reached the *Exit*.

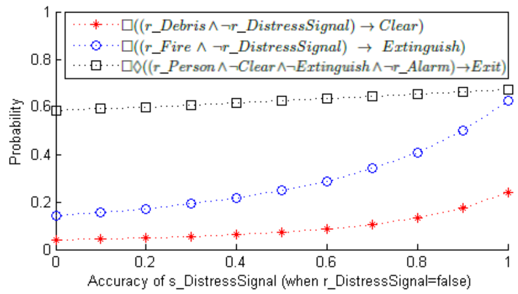

Fig. 10. Plot of the results for the Search and Rescue example over varying sensor accuracies for the *DistressSignal* sensor (false positives only).

The *r_Fire* proposition had a 0.25 probability of becoming true when previously false, and a 0.5 chance of remaining true. Similarly, the *r_Debris* proposition had a 0.5 chance of becoming true, and a 0.75 chance of remaining true. All three of these propositions were assumed to be false in the *Exit* region. Finally, the *r_DistressSignal* and *r_Alarm* propositions had a 0.2 and 0.1 chance of becoming true, respectively. Once true, both propositions remain so with a 0.9 probability.

Sensors: For this example, the *s_Fire*, *s_Debris*, and *s_Person* sensors had accuracies that were fixed at symmetric values of 0.85, 0.75, and 0.9, respectively. The *s_DistressSignal* sensor had a 0.8 and 0.95 probability of correctly sensing false and true values of *r_DistressSignal*, respectively. Similarly, the *s_Alarm* sensor had values of 0.85 and 0.95 for correctly sensing false and true, respectively.

Analysis properties: For this example, we analyzed the performance of the controller with respect to three properties:

1) $\square((r\_Debris \wedge \neg r\_DistressSignal) \rightarrow Clear)$ "*Clear* all encountered *Debris* if there is no *DistressSignal*"
2) $\square((r\_Fire \wedge \neg r\_DistressSignal) \rightarrow Extinguish)$ "*Extinguish* all encountered *Fire* if there is no *DistressSignal*"
3) $\square\lozenge((r\_Person \wedge \neg Clear \wedge \neg Extinguish \wedge \neg r\_Alarm) \rightarrow Exit)$ "Lead any *Person* to the *Exit*, if not *Clearing* or *Extinguishing*, and there is no *Alarm*"

Indirect Effects: The effects of changes in the accuracy of the *s_DistressSignal* sensor (when *r_Distress* is false) are shown in Figure 10. We see, in this figure, that the third specification, which is not directly related to the sensor in question, improves as the sensor accuracy increases. This indirect effect of the *s_DistressSignal* sensor on the third specification can be attributed to the rate at which the robot senses *DistressSignal* to be true. As the sensor accuracy increases, there are less occurrences of false positives, and the robot more often activates *Clear* and *Extinguish*, either of which satisfy the specification.

Event Frequency Effects: Figure 11 shows the effects of changes in the rate at which *r_DistressSignal* becomes true. We can see that an increase in the frequency of *DistressSignal* will dramatically affect the performance of our controller. The probability that the controlloer will satisfy the third specification is better at low frequencies of *r_DistressSignal*, where the robot would more often activate *Clear* or *Extinguish* (either of which satisfy the specification). A larger rate of occurrence, however, would improve the probability that the controller would satisfy the first two specifications, as both

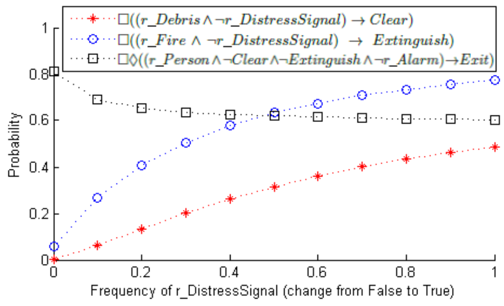

Fig. 11. Results for the Search and Rescue example over varying frequencies for the *r_DistressSignal* proposition changing from false to true.

are satisfied when *r_DistressSignal* is true.

VI. Conclusion

In this paper, we present a novel approach for assessing the performance of automatically synthesized controllers under the presence of erroneous sensors. By composing a model of the system, including the behavior of the environment and sensors, we can use probabilistic model checking techniques to compute the probability with which the controller will satisfy a set of high-level specifications. We discuss the calculation of parametric formulas representing the satisfaction probability as a function of the model parameters (frequency of environment changes and sensor accuracies). The algorithms presented in this paper allow for the inclusion of time bounds and the nesting of a single "next" temporal operator in the analyzed formulas. The applicability of this approach is presented with reference to three different examples.

The work presented here facilitates the assessment and redesign of a controller synthesized from temporal logic specifications and provides a foundation for further related work. In the future, we intend to extend the applicability of this approach to include actuation uncertainty in the assessment of controller performance, and to incorporate such analysis into the automated selection and synthesis of the controller.

References

[1] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2696. IEEE, 2010.

[2] D.C. Conner, H. Kress-Gazit, H. Choset, and A.A. Rizzi. Valet parking without a valet. *Proc. of IEEE/RSJ*, pages 572–577, October 2007.

[3] E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, E(16):995–1072, July 1990.

[4] G. Fainekos, A. Girard, and G. Pappas. Hierarchical synthesis of hybrid controllers from temporal logic specifications. In *Hybrid Systems: Computation and Control*, pages 203–216. Springer, 2007.

[5] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. Hybrid Controllers for Path Planning: A Temporal Logic Approach. *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4885–4890, 2006.

[6] C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP : Experimenting with Language , Temporal Logic and Robot Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.*, pages 1988–1993, Taipei, Taiwan, 2010.

[7] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *International Journal on Software Tools for Technology Transfer*, 13:3–19, April 2010.

[8] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conference on Decision and Control*, pages 2222–2229, 2009.

[9] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from LTL specifications. In *Hybrid Systems: Computation and Control*, volume 3927, pages 333–347, 2006.

[10] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Translating Structured English to Robot Controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

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

[12] M. Kwiatkowska, G. Norman, D. Parker, and M. Kattenbelt. PRISM Probabilistic Model Checker, 2010. URL http://www.prismmodelchecker.org/.

[13] M. Lahijanian, S.B. Andersson, and C. Belta. A Probabilistic Approach for Control of a Stochastic System from LTL Specifications. In *IEEE Conference on Decision and Control*, pages 2236–2241, Shanghai, P.R. China, 2009.

[14] M. Lahijanian, J. Wasniewski, S.B. 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*, pages 3227–3232, Anchorage, AK, May 2010.

[15] S.G. Loizou and K.J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on LTL specifications. In *IEEE Conference on Decision and Control*, pages 153–158 Vol.1. IEEE, 2004.

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

[17] P. Tabuada and G.J. Pappas. Model checking LTL over controllable linear systems is decidable. *Hybrid systems: computation and control*, pages 498–513, 2003.

[18] P. Tabuada and G.J. Pappas. Linear time logic control of discrete-time linear systems. *Automatic Control, IEEE Transactions on*, 51(12):1862 –1877, December 2006.

[19] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding Horizon Temporal Logic Planning. In *IEEE Conference on Decision and Control (CDC)*, pages 5997–6004. Ieee, December 2009.

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