2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Motion Planning and Control from Temporal Logic Specifications with Probabilistic Satisfaction Guarantees

M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta

Abstract— **We present a computational framework for automatic deployment of a robot from a temporal logic specification over a set of properties of interest satisfied at the regions of a partitioned environment. We assume that, during the motion of the robot in the environment, the current region can be precisely determined, while due to sensor and actuation noise, the outcome of a control action can only be predicted probabilistically. Under these assumptions, the deployment problem translates to generating a control strategy for a Markov Decision Process (MDP) from a temporal logic formula. We propose an algorithm inspired from probabilistic Computation Tree Logic (PCTL) model checking to find a control strategy that maximizes the probability of satisfying the specification. We illustrate our method with simulation and experimental results.**

I. Introduction

In the "classical" motion planning problem [1], a specification is given simply as "go from A to B and avoid obstacles", where A and B are two regions of interest in some environment. However, a mission might require the attainment of either A or B, convergence to a region ("reach A eventually and stay there for all future times"), visiting targets sequentially ("reach A, and then B, and then C"), surveillance ("reach A and then B infinitely often"), or the satisfaction of more complicated temporal and logic conditions about the reachability of regions of interest ("Never go to A. Don't go to B unless C or D were visited"). To accommodate such increased expressivity in the specification language, recent works suggested the use of temporal logics, such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) as motion specification languages [2]–[6]. Algorithms inspired from model checking [7] or temporal logic games [8] are used to find motion plans and control strategies from such specifications.

The starting point for these works is to abstract the partitioned environment to its partition quotient graph, which is then interpreted as a transition system, or Kripke structure [7] during the model checking or game algorithm. To enable the application of such techniques, the existing temporal logic approaches to motion planning are based on two main assumptions. First, the transition system is either purely deterministic (*i.e.,* in each region, an available control action determines a unique transition to the next region) or purely nondeterministic (a control action in a region can enable transitions to several next regions, with no information on their likelihoods) [9]. Second, during its motion in the environment, the robot can determine its position precisely. In realistic robotic applications, both these assumptions might not hold. First, due to noisy actuators, a control action can never be guaranteed to produce the desired next transitions. However, the transition probabilities can be computed given the sensor and actuator noise model or through experimental trials. Second, due to noisy measurements, the current region of a robot cannot be known precisely, but a distribution over the set of regions can usually be inferred from a measurement. These observations lead to a Partially Observed Markov Decision Process (POMDP) model of robot motion [10]. If the motion specification is given simply as "go from A to B," then numerous algorithms exist to determine a robot control strategy [11]–[13]. The probabilistic counterparts of the rich, temporal logic specifications are probabilistic temporal logics, such as probabilistic LTL [14], probabilistic CTL (PCTL) [15] and the Continuous Stochastic Logic (CSL) [16]. However, the problem of generating a control strategy for a POMDP from a probabilistic temporal logic formula is currently not well understood.

In this paper, we consider a simpler version of the above problem. While we allow for actuation noise (*i.e.,* the control actions enable transitions with known probabilities), we assume that the robot can determine its current regions precisely. This assumption is not overly restrictive for indoor navigation applications, such as the one considered in this paper, where a large number of reliable RFID tags can be placed in the environment. The robot motion model becomes a Markov Decision Process (MDP). We consider specifications given as PCTL formulas and develop a framework for automatic synthesis of control strategies from such specifications. While our solution to this problem is based on a simple adaptation of existing PCTL model checking algorithms [17], the framework is, to the best of our knowledge, novel and quite general. In short, given a specification as a formula in a fragment of CTL, the algorithm returns the maximum satisfaction probability and a corresponding control strategy. To illustrate the method, we built a Robotic InDoor Environment (RIDE) [18], in which an iRobot iCreate platform can move autonomously through corridors and intersections that can be easily reconfigured.

The remainder of the paper is organized as follows. In Sec. II, we formulate the problem and outline our approach. The MDP control strategy is briefly described in Sec. III. The experimental platform and the MDP modeling technique for the robot motion are presented in Sec. IV, while experimental results are included in Sec. V. The paper concludes with final

This work is partially supported at Boston University by the NSF under grants CNS-0834260 and CMMI-0928776, the ARO under grant W911NF-09-1-0088, and the AFOSR under grant FA9550-09-1-0209.

The authors are with the Department of Mechanical Engineering, Boston University, MA, USA, E-mail: morteza@bu.edu.

M. Lahijanian is the corresponding author.

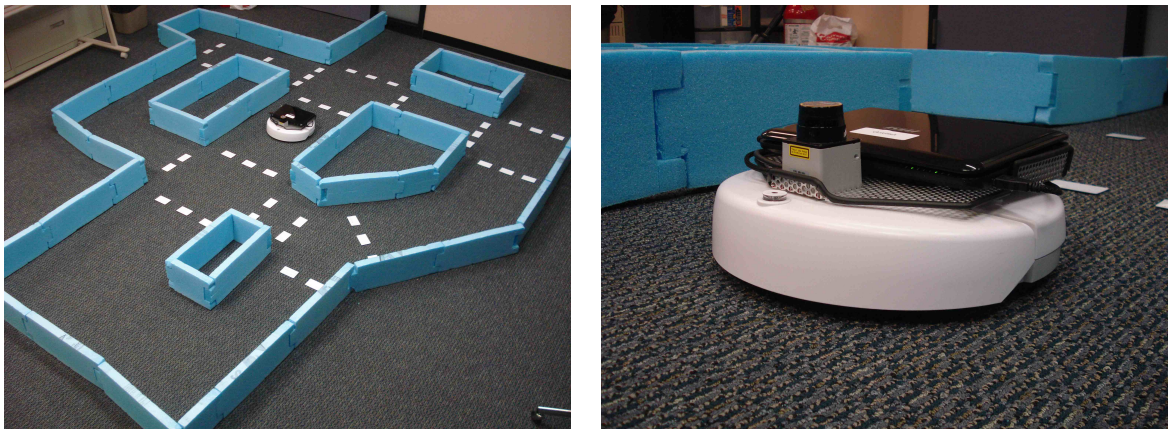

Fig. 1. Robotic InDoor Environment (RIDE). Left: An iCreate mobile platform equipped with a laptop, a laser range finder, and RFID reader moves autonomously through the corridors and intersection of an indoor-like environment, whose topology can be easily reconfigured by moving the foam walls. Right: A robot close-up.

remarks in Sec. VI.

II. Problem Formulation and Approach

Consider a robot moving in a partitioned environment, such as the one shown in Fig. 1 left. We assume that the robot is programmed with a small set of feedback control primitives allowing it to move inside each region and from a region to its adjacent one. We make the natural assumption that these control primitives are not completely reliable. In other words, if at a given region a control primitive designed to take the robot to a specific adjacent region is used, it is possible that the robot will instead transition to a different adjacent region. We also assume that the success/failure rates of such controllers are known. In a particular application, these rates can be determined experimentally, or with a combination of experiments and simulations, as we discuss later in the paper. We consider the following problem:

Problem 1: Given a motion specification as a rich, temporal logic statement about properties satisfied by the regions in a partitioned environment, find a robot control strategy that maximizes the probability of satisfying the specification.

Consider for example RIDE shown in Fig. 1, whose schematic representation is given in Fig. 2. The regions are roads and intersections, identified by $R_1, \ldots, R_9$ and $I_1, \ldots, I_5$, respectively. There are four properties of interest about the regions: *Safe* (the robot can safely drive through a road or intersection with this property), *Relatively safe* (the robot can pass through the region but it should avoid it if possible), *Unsafe* (the corresponding region should be avoided), *Medical supply* (there are medical supplies in the region associated with this property), and *Destination* (a region that is required to be visited by the robot has this property). Examples of temporal logic motion specifications include "Reach *Destination* and always avoid *Unsafe* regions", "Reach *Destination* while going through either only *Safe* regions or through *Relatively safe* regions only if *Medical Supply* is available at such regions".

As it will become clear later in the paper, the rich, temporal logic specification will be a formula of a fragment of CTL, which seems to be rich enough to accommodate a fairly large spectrum of robotic missions. A robot control strategy will be defined as an assignment of a control primitive to each region of the environment. Since the outcome of a control primitive is not guaranteed but characterized probabilistically, the satisfaction of the specification is defined in a probabilistic sense. Among all the possible control strategies, our solution to Problem 1 will have the highest rate of success.

Central to our approach to Problem 1 is an MDP (Sec. III-A) representation of the motion of the robot in the environment. Each state of the MDP in general corresponds to an ordered set of regions in the partition while the actions are labels for the feedback control primitives. The construction of an MDP model for the motion of the robot is described in Sec. IV. Under the additional (and restrictive) assumption that, at any given time, the robot knows precisely its current region, generating a robot control strategy solving Problem 1 reduces to finding an MDP control strategy that maximizes the probability of satisfying a PCTL formula. This problem is treated in Sec. III. While the framework that we develop in this paper is quite general, we focus on RIDE (Sec. IV-A).

When the criterion is expressed as the optimization of a cost function subject to an MDP, there are a variety of tools for solving the problem, such as dynamic programming and successive approximation. These methods are polynomial in complexity in the size of the state space [19]. Since Problem 1 involves a maximization over the dynamics of an MDP, it is possible to convert the specification into a cost function and take advantage of such tools. To do so in general, however, requires state augmentation to capture the expressivity of the temporal operators since the cost must be expressed as a summation of per stage costs, each of which can only depend on the current state and choice of action. As discussed in Sec. III, the model checking scheme we adopt is also polynomial in the size of the state space and thus conversion to a cost function form is less computationally efficient, even when ignoring the difficulty of finding a cost function that captures the specification.

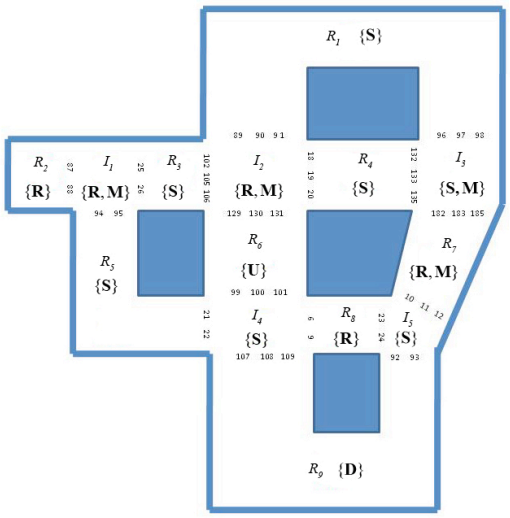

Fig. 2. Schematic representation of the environment from Fig. 1. Each region has a unique identifier ($R_1, \ldots, R_9$ for roads and $I_1, \ldots, I_5$ for intersections, respectively). The properties satisfied at the regions are shown between curly brackets inside the regions: **S** = *Safe*, **R** = *Relatively safe*, **U** = *Unsafe*, **M** = *Medical supply*, and **D** = *Destination*.

III. MDP Control Strategies from PCTL Specifications

In this section we describe a procedure for generating an MDP control policy that maximizes the probability of satisfying a given PCTL specification, thereby solving Problem 1. Our approach is an adaptation of the PCTL model checking algorithm [20]. Due to space constraints, we give a somewhat informal description and illustrate the concepts through simple examples.

A. Markov Decision Process

Given a set Q, let 2^Q and $|Q|$ denote its power set and cardinality respectively. We give the following definition.

Definition 1: A Markov Decision Process (MDP) is a tuple $\mathcal{M} = (Q, q_0, Act, Steps, L)$ where:

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- Act is the set of actions;
- $Steps : Q \to 2^{Act \times \Sigma(Q)}$ is a transition probability function, where $\Sigma(Q)$ is the set of all discrete probability distributions over the set Q;
- $L : Q \to 2^{\Pi}$ is a labeling function assigning to each $q \in Q$ possibly several elements of a set Π of properties.

The set of actions available at $q \in Q$ is denoted $\mathcal{A}(q)$. The function $Steps$ is often represented as a matrix with $|Q|$ columns and $\sum_{i=0}^{|Q|-1} |\mathcal{A}(q_i)|$ rows (c.f. Eqn. (1) below). For each action $a \in \mathcal{A}(q)$, we denote the probability of transitioning from state q_i to state q_j under the action a as $\sigma_a^{q_i}(q_j)$ and the corresponding probability distribution function as σ_a. Each σ_a corresponds to one row in the matrix representation of $Steps$.

To illustrate these definitions, a simple MDP is shown in Fig. 3. The actions available at each state are $\mathcal{A}(q_0) = \{a_1\}$, $\mathcal{A}(q_1) = \{a_2, a_3, a_4\}$, and $\mathcal{A}(q_2) = \mathcal{A}(q_3) = \{a_1, a_4\}$. The

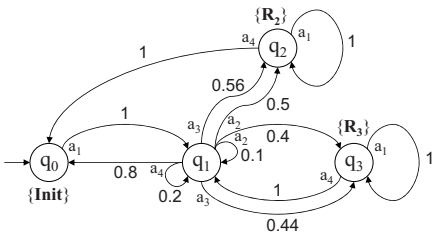

Fig. 3. A four-state MDP.

labels are $L(q_0) = \{\textbf{Init}\}$, $L(q_2) = \{\mathbf{R_2}\}$, and $L(q_3) = \{\mathbf{R_3}\}$. The matrix representation of $Steps$ is given by

$$Steps = \begin{array}{c} \\ q_0;a_1 \\ q_1;a_2 \\ q_1;a_3 \\ q_1;a_4 \\ q_2;a_1 \\ q_2;a_4 \\ q_3;a_1 \\ q_3;a_4 \end{array} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.1 & 0.5 & 0.4 \\ 0 & 0 & 0.56 & 0.44 \\ 0.8 & 0.2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \quad (1)$$

B. Paths, Control Policies, and Probability Measures

A path ω through an MDP is a sequence of states $\omega = q_0 q_1 \ldots q_i q_{i+1} \ldots$ where each transition is induced by a choice of action at the current step i. We denote the set of all finite paths by $Path^{fin}$ and of infinite paths by $Path$.

A *control policy* is a function $A : Path^{fin} \to Act$. That is, for every finite path, a policy specifies the next action to be applied. Under a policy A, an MDP becomes a Markov chain, denoted $\mathcal{D}_A$. Let $Path_A \subseteq Path$ and $Path_A^{fin} \subseteq Path^{fin}$ denote the set of infinite and finite paths that can be produced under A. Because there is a one-to-one mapping between $Path_A$ and the set of paths of $\mathcal{D}_A$, the Markov chain induces a probability measure over $Path_A$ as follows.

First, define a measure $Prob_A^{fin}$ over the set of finite paths by setting the probability of $\omega_{fin} \in Path_A^{fin}$ equal to the product of the corresponding transition probabilities in $\mathcal{D}_A$. Then, define $C(\omega_{fin})$ as the set of all (infinite) paths $\omega \in Path_A$ with the prefix ω_{fin}. The probability measure on the smallest σ-algebra over $Path_A$ containing $C(\omega_{fin})$ for all $\omega_{fin} \subset Path_A^{fin}$ is the unique measure satisfying

$$Prob_A(C(\omega_{fin})) = Prob_A^{fin}(\omega_{fin}) \, \forall \omega_{fin} \in Path_A^{fin}. \quad (2)$$

To illustrate this measure, consider the MDP shown in Fig. 3 and the simple control policy defined by the mapping

$$\begin{aligned} A_1(q_0) = a_1,\ A_1(\cdots q_1) = a_2, \\ A_1(\cdots q_2) = a_4,\ A_1(\cdots q_3) = a_1, \end{aligned} \quad (3)$$

where $\cdots q_i$ denotes any finite path terminating in q_i. The initial fragment of the resulting Markov chain is shown in Fig. 4. From this fragment it is easy to see that the probability of the finite path $q_0 q_1 q_2$ is $Prob_{A_1}^{fin}(q_0 q_1 q_2) = 0.5$. Under A_1, the set of all infinite paths with this prefix is

$$C(q_0 q_1 q_2) = \{\overline{q_0 q_1 q_2},\ q_0 q_1 q_2 q_0 \overline{q_1},\ q_0 q_1 q_2 q_0 q_1 \overline{q_3}, \ldots\}$$

where the sequence under the over-line is repeated infinitely. According to (2), we have that

$$Prob_{A_1}(C(q_0 q_1 q_2)) = Prob_{A_1}^{fin}(q_0 q_1 q_2) = 0.5.$$

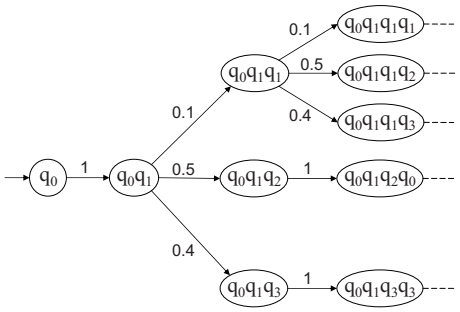

Fig. 4. Fragment of DTMCs $\mathcal{D}^{A_1}$ for control policy A_1.

C. PCTL Control Generation and Model Checking

With the measure defined above, one can determine the probability of satisfying a specification by calculating the probability of the paths that satisfy it. We use PCTL [20] , a probabilistic extension of CTL that includes a probabilistic operator $\mathcal{P}$. Formulas of PCTL are interpreted over states of an MDP and are constructed by connecting properties from a set Π using standard Boolean operators (including $true$, $\neg$ ("negation"), $\wedge$ ("conjunction"), and $\rightarrow$ ("implication")), the temporal operator $\mathcal{U}$ denoting "until", and the probabilistic operator $\mathcal{P}$.

To solve Problem 1, we are interested only in finding the control policy producing the *maximum* probability of satisfying a given specification. Such PCTL formulas have the form $\mathcal{P}_{max=?}[\phi_1 \mathcal{U} \phi_2]$ where ϕ_1 and ϕ_2 are arbitrary formulas involving only Boolean operators. The control generation and model checking algorithm takes such a PCTL formula, ϕ, and an MDP, $\mathcal{M}$, and returns both the maximum probability over all possible policies that ϕ is satisfied and a control policy that produces this probability.

The method proceeds as follows. The state space Q is partitioned into three subsets. The first, Q^{yes}, contains all those states that satisfy the formula with probability 1 for some control policy A. The second, Q^{no} contains those states for which the probability of satisfying the formula is 0 for all control policies, while $Q^{?}$ contains the remaining states.

Let x_{q_i} denote the probability of satisfying ϕ from state $q_i \in Q$. For all $q_i \in Q^{yes}$, we have $x_{q_i} = 1$ and for all $q_i \in Q^{no}$ we have $x_{q_i} = 0$. The remaining values are determined by the following linear optimization problem.

$$\text{Minimize} \sum_{q_i \in Q^{?}} x_{q_i} \text{ subject to:}$$

$$x_{q_i} \geq \sum_{q'_j \in Q^{?}} \sigma_a^{q_i}(q'_j) \, . \, x_{q'_j} + \sum_{q'_j \in Q^{yes}} \sigma_a^{q_i}(q'_j)$$

$$\text{for all } q_i \in Q^{?} \text{ and } (a, \sigma_a) \in Steps(q_i).$$

Finding the unique solution to this problem yields the optimal probabilities x_{q_i}, actions, and their corresponding probability distributions (a, σ_a). The desired control policy is thus the function mapping each state to the action identified by the solution to this linear optimization problem.

To illustrate the scheme, consider once again the example MDP shown in Fig. 3. We choose the specification $\mathcal{P}_{max=?}[\neg \mathbf{R}_3 \, \mathcal{U} \, \mathbf{R}_2]$. In words, this formula states that we wish to find the policy that maximizes the probability of reaching the region satisfying $\mathbf{R}_2$ without passing through the region satisfying $\mathbf{R}_3$. State q_2 satisfies the formula while state q_3 does not. Therefore, $Q^{yes} = \{q_2\}$, $Q^{no} = \{q_3\}$, and $Q^{?} = \{q_0, q_1\}$. From this we have that $x_{q_2} = 1$ and $x_{q_3} = 0$. The solution to the linear optimization problem can be found to be $x_{q_0} = x_{q_1} = 0.56$ under the policy defined by mapping the states q_0 and q_1 to the actions a_1 and a_3. Thus, the maximum probability of satisfying ϕ starting from q_0 is 0.56.

Note that the expressivity of PCTL is limited. Nevertheless, it is a useful specification language in the context of mobile robotics in the stochastic setting. The combination of the probabilistic operator with Boolean and temporal operators allows for formulas capturing complex specifications (*e.g.,* the case studies in Sec. V). One of the features of this approach is that the complexity of PCTL model checking is linear in the length of the specification formula ϕ, defined as the number of logical connectives and temporal operators plus the sizes of the temporal operators, and polynomial in the size of the MDP. It is therefore feasible to apply the scheme to realistic robotic scenarios. Finally, our implementation for determining a control strategy as described above is based on PRISM version 3.3 [21].

IV. Construction and Validation of an MDP Model for Robot Motion

A. Experimental Platform and Simulation Tool

To test the algorithms proposed in this paper, we built RIDE (Fig. 1), which consists of corridors of various widths and lengths and intersections of several shapes and sizes. The walls were constructed from pieces of extruded polystyrene with a "jigsaw" pattern cut into each end so neighboring pieces could be interlocked. The shape of the pieces and their non-permanent fastening ability allow for an easy-to-assemble, reconfigurable, and scalable environment. Each corridor and intersection (identified by $R_1, \ldots, R_9$ and $I_1, \ldots, I_5$ in Fig. 2) is bounded by a line of RFID tags (white patches in Fig. 1 left and small ID numbers in Fig. 2) meant to trigger a transition event. The correct reading of such an event guarantees that the robot knows precisely its current region at any time. This, together with the satisfaction of the Markovian property enforced as described in Sec. IV-B, allows us to model the robot motion as an MDP, and therefore the control strategy presented in Sec. III can be used for deployment. The mobile platform is an iRobot iCreate fitted with a Hokoyu URG-04LX laser range finder, APSX RW-210 RFID reader, and an MSI Wind U100-420US netbook. Commands are provided to the iCreate from Matlab using the iRobot Open Inteface and the iCreate Matlab Toolbox [22]. The communications between the sensors, netbook, and robot occur through the use of USB connections.

The robot's motion is determined by specifying a forward velocity and angular velocity. At a given time, the robot implements one of the following four controllers (motion primitives) - FollowRoad, GoRight, GoLeft, and GoStraight.

Each of these controllers operates by obtaining data from the laser scanner and calculating a "target angle." The target angle represents the desired heading of the robot in order to execute the specified command, and this angle is translated into a proportional control law for angular velocity. The target angle in each case is found by utilizing two laser data points at certain angles relative to the robot's heading (different for each controller) and finding the midpoint between them. The target angle is then defined as the angle between the robot's heading and a line connecting the midpoint and the center of the robot. For each of these four controllers, the forward velocity control specified is based on the distance to the nearest obstacle in any direction. Therefore, as the robot approaches an obstacle, it tends to slow down. A cap is placed on the maximum velocity to keep the robot at reasonable speeds in more open areas of the environment. Each controller also provides for obstacle avoidance and emergency actions.

To test the robot control strategies before experimental trials and also to generate sample data necessary for the construction of the MDP model of the robot motion (Sec. IV-B), we built a RIDE simulator (Fig. 5). The simulator was designed to resemble the actual experiment very closely. Specifically, it emulates experimentally measured response times, sensing and control errors, and noise levels and distributions in the laser scanner readings. The configuration of the environment in the simulator is also easily reconfigurable to capture changes in the topology of the experimental environment.

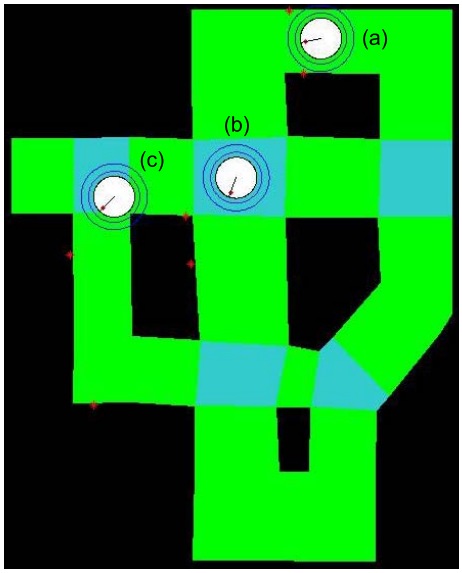

Fig. 5. Snapshots from the RIDE simulator. The robot is represented as a white disk. The arrow inside the white disk shows the robot's heading. The inner circle around the robot represents the "emergency" radius (if there is an obstacle within this zone, the emergency controller is used). The outer circle represents the radius within which the forward speed of the robot varies with the distance to the nearest obstacle. If there are no obstacles in this area, the robot moves at maximum speed. The red dots are the laser readings used to define the target angle. (a) The robot centers itself on a stretch of road by using FollowRoad; (b) The robot applies GoRight in an intersection but fails to turn right because the laser readings do not properly identify the road on the right; (c) The robot applies GoLeft in an intersection and successfully turns left because the laser readings fall inside the road.

B. Construction and validation of the MDP model

In this section we discuss the construction of an MDP model (Def. 1) for the motion of the robot in the environment. In summary, the set of actions of the MDP is the set of controllers - FollowRoad, GoRight, GoLeft, and GoStraight. Each state of the MDP is a collection of regions such that the Markovian property is satisfied (*i.e.,* the result of an action at a state depends only on the current state). The set of actions available at a state is the set of controllers available at the last region in the set of regions corresponding to the state. More details on the construction of the MDP are given below.

The environment (Fig. 2) consists of nine roads, within which only the controller FollowRoad is available. There are also two 4-way and three 3-way intersections in the environment. The controllers available at 4-way intersections are GoRight, GoLeft, and GoStraight, while at the 3-way intersections only GoRight and GoLeft controllers are available. Through extensive experimental trials, we concluded that, by grouping two adjacent regions (a road and an intersection) in a state, we achieve the Markovian property, for all pairs of adjacent regions. For example, the connecting regions of R_1-I_2 represent one state of the MDP, which has transitions to states I_2-R_4, I_2-R_3, and I_2-R_6 enabled by actions GoLeft, GoRight, and GoStraight. When designing the robot controllers, we also made sure that the robot never gets stuck in a region, *i.e.,* the robot can only spend a finite amount of time in each region. Thus, the states are of the form intersection-road and road-intersection (states such as I_i-I_i or R_i-R_i do not exist). The resulting MDP for the environment shown in Fig. 2 has 34 states. The set of actions available at a state of the MDP is the set of controllers available at the second region of the state. For example, when in state R_1-I_2 only those actions from region I_2 are allowed.

As already outlined, since extensive experimentation in RIDE is time consuming, we used the RIDE simulator to compute the transition probabilities associated to each action. We performed a total of 500 simulations for each controller available in each MDP state. In each trial, the robot was initialized at the beginning of the first region of each state. If this region was a road, then the FollowRoad controller was applied until the system transitioned to the second region of the state. If the first region was an intersection then the controller most likely to transition the robot to the second region was applied. Once the second region was reached, one of the allowed actions was applied and the resulting transition was recorded. The results were then compiled into the transition probabilities. The set of properties of the MDP is $\Pi = \{\mathbf{S}, \mathbf{R}, \mathbf{U}, \mathbf{M}, \mathbf{D}\}$, where **S** = *Safe*, **R** = *Relatively safe*, **U** = *Unsafe*, **M** = *Medical supply*, and **D** = *Destination*. Each state of the MDP is mapped to the set of properties that are satisfied at the second region of the state (Fig. 2).

To make sure that the MDP obtained through this extensive simulation procedure was a good model for the actual motion of the robot in the experimental platform,

we randomly selected four transition probability distributions and experimentally determined the transition probabilities. We then compared the simulated-based and experimental-based probabilities using either the Fisher exact test [23] when only a few experimental trials were available or the chi-square test for homogeneity if there were a large number of experimental trials. These tests confirmed that the experimental data of four randomly selected transition probabilities were not significantly different from simulation results with a minimum certainty of 0.95.

V. Case Studies

Consider the RIDE configuration shown in Figs. 1 and 2 and the following two motion specifications:

Specification 1: "Reach *Destination* by driving through either only *Safe* regions or through *Relatively safe* regions only if *Medical Supply* is available at such regions."

Specification 2: "Reach *Destination* by driving through *Safe* or *Relatively safe* regions only."

Specifications 1 and 2 translate naturally to PCTL formulas ϕ_1 and ϕ_2, respectively, where

$$\phi_1 \quad : \quad \mathcal{P}_{max=?}\left[(\mathbf{S} \vee (\mathbf{R} \wedge \mathbf{M}))\,\mathcal{U}\,\mathbf{D}\right] \tag{4}$$

$$\phi_2 \quad : \quad \mathcal{P}_{max=?}\left[(\mathbf{S} \vee \mathbf{R})\,\mathcal{U}\,\mathbf{D}\right] \tag{5}$$

Assuming that the robot is initially at R_1 oriented towards I_2, we use the computational framework described in this paper to find control strategies maximizing the probabilities of satisfying the above specifications. The maximum probabilities for Specifications 1 and 2 are 0.227 and 0.674, respectively. To confirm these predicted probabilities, we performed 500 simulation and 35 experimental runs for each of the optimal control strategies. The simulations showed that the probabilities of satisfying ϕ_1 and ϕ_2 were 0.260 and 0.642, respectively. From the experimental trials, we inferred that the probabilities of satisfying ϕ_1 and ϕ_2 were 0.229 and 0.629, respectively. By using the chi-square and Fisher's exact statistical tests, we concluded that the frequency of trials satisfying the specifications in the experiment matched the simulation data with a minimum certainty of 0.95.

Movies showing the experimental trials obtained by applying the control strategies resulted from Specifications 1 and 2 are available for download from [18]. In these videos, it can be seen that, for example, the motion of the robot produced by the control strategy obtained from Specification 2 follows the route $R_1 I_2 R_3 I_1 R_5 I_4 R_8 I_5 R_9$ which satisfies ϕ_2 because **S** or **R** are true until **D** is satisfied.

VI. Conclusion

We presented a computational framework for automatic deployment of a mobile robot from a temporal logic specification about properties of interest satisfied at the regions of a partitioned environment. We modeled the motion of the robot as an MDP, and mapped the robot deployment problem to the problem of generating an MDP control strategy maximizing the probability of satisfying a PCTL formula. For the latter, we used a modified PCTL model checking algorithm. We illustrated the method with experimental results in our RIDE.

Acknowledgements

The authors would like to thank K. Ryan and B. Chang-Yun Hsu from Boston University for their help with the development of the RIDE simulator.

References

[1] S. M. LaValle, *Planning algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[2] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *43rd IEEE Conference on Decision and Control*, December 2004.

[3] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot motion planning: A timed automata approach," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004, pp. 4417–4422.

[4] H. K. Gazit, G. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *IEEE Conference on Robotics and Automation*, Rome, Italy, 2007.

[5] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: a temporal logic approach," in *Proceedings of the 2005 IEEE Conference on Decision and Control*, Seville, Spain, December 2005.

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

[7] E. M. M. Clarke, D. Peled, and O. Grumberg, *Model checking*. MIT Press, 1999.

[8] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive(1) designs," in *VMCAI*, Charleston, SC, 2006, pp. 364–380.

[9] M.Kloetzer and C. Belta, "Dealing with non-determinism in symbolic control," in *Hybrid Systems: Computation and Control: 11th International Workshop*, ser. Lecture Notes in Computer Science, M. Egerstedt and B. Mishra, Eds. Springer Berlin / Heidelberg, 2008, pp. 287–300.

[10] J. Pineau and S. Thrun, "High-level robot behavior control using POMDPs," in *AAAI Workshop notes*, Menlo Park, CA, 2002.

[11] D. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific, 1995, vol. 1.

[12] J. Pineau, G. Gordon, and S. Thrun, "Anytime point-based approximations for large POMDPs," *Journal of Artificial Intelligence Research*, vol. 27, pp. 335–380, 2006.

[13] N. L. Zhang and W. Zhang, "Speeding up the convergence of value iteration in partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 14, pp. 29–51, 2001.

[14] C. Baier, "On algorithmic verification methods for probabilistic systems," Ph.D. dissertation, 1998.

[15] L. D. Alfaro, "Model checking of probabilistic and nondeterministic systems." Springer-Verlag, 1995, pp. 499–513.

[16] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time markov chains," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6.

[17] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 6, no. 2, pp. 128–142, 2004.

[18] "Robotic indoor environment." [Online]. Available: hyness.bu.edu/ride/

[19] C. Papadimitriou and J. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[20] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), ser. CRM Monograph Series. American Mathematical Society, 2004, vol. 23.

[21] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, ser. LNCS, H. Hermanns and J. Palsberg, Eds., vol. 3920. Springer, 2006, pp. 441–444.

[22] J. M. Esposito and O. Barton, "Matlab toolbox for the irobot create," www.usna.edu/Users/weapsys/esposito/roomba.matlab/, 2008.

[23] A. Trujillo-Ortiz, R. Hernandez-Walls, A. Castro-Perez, L. Rodriguez-Cardozo, N. Ramos-Delgado, and R. Garcia-Sanchez., "Fisherextest:fisher's exact probability test." http://www.mathworks.com/matlabcentral/fileexchange/5957, 2004.