Sampling-Based Reactive Motion Planning with Temporal Logic Constraints and Imperfect State Information

Felipe J. Montana[1(✉)], Jun Liu[2], and Tony J. Dodd[1]

[1] Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, UK
{fjmontanagonzalez1,t.j.dodd}@sheffield.ac.uk
[2] Department of Applied Mathematics, University of Waterloo, Waterloo, Canada
j.liu@uwaterloo.ca

Abstract. This paper presents a method that allows mobile systems with uncertainty in motion and sensing to react to unknown environments while high-level specifications are satisfied. Although previous works have addressed the problem of synthesising controllers under uncertainty constraints and temporal logic specifications, reaction to dynamic environments has not been considered under this scenario. The method uses feedback-based information roadmaps (FIRMs) to break the curse of history associated with partially observable systems. A transition system is incrementally constructed based on the idea of FIRMs by adding nodes on the belief space. Then, a policy is found in the product Markov decision process created between the transition system and a Rabin automaton representing a linear temporal logic formula. The proposed solution allows the system to react to previously unknown elements in the environment. To achieve fast reaction time, a FIRM considering the probability of violating the specification in each transition is used to drive the system towards local targets or to avoid obstacles. The method is demonstrated with an illustrative example.

1 Introduction

Efficient motion planning with imperfect state information is a desirable ability of systems operating in uncertain and dynamic environments. In these cases the system cannot decide the best actions based on a single deterministic state. Instead, a probability distribution over all possible states, called belief, is considered. This problem can be mathematically modelled as a partially observable Markov decision process (POMDP). Although several methods have adapted discrete POMDPs to motion planning, they have, in general, poor scalability with the number of states. This is caused by two main sources of complexity: (i) the so-called curse of dimensionality, for a system with n states, the belief space is an $(n-1)$-dimensional continuous space; and (ii) the curse of history [16], the number of distinct action-observation histories grows exponentially

Felipe J. Montana is supported by the Mexican National Council of Science and Technology (CONACyT). Jun Liu is supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Canada Research Chairs (CRC) program.

with the planning horizon. To alleviate this problem, sampling-based methods have been proposed, e.g., [1, 5, 17]. In these works, the objective is usually to optimally drive the system from an initial to a final state. Nevertheless, more complex objectives are required in some applications. This necessity has motivated the use of formal methods to automatically synthesise controllers for mobile systems such that high-level specifications are satisfied. Due to well-developed techniques in model checking using temporal logic, these specifications are commonly defined by linear temporal logic (LTL) formulae for robotic applications.

Using model checking techniques, several methods have been developed to solve the problem of control synthesis for stochastic systems with perfect state information, e.g., [9, 15]. However, only few solutions have been presented for stochastic systems with partially observable states. Wongpiromsarn et al. [21] propose a method to compute policies that maximise the probability of satisfying an LTL specification for partially known environments. They assume that the environment can be in one of several modes, which are modelled as Markov chains. Although the system does not know exactly which is the current mode of the environment at each time, all the possible environment models are known by the system. This is a limitation since in many applications these models are not available. The policies are computed using a parallel composition between an MDP modelling the system and the set of Markov chains. Vasile et al. [20] propose a specification language, called Gaussian Distribution Temporal Logic (GDTL), that permits including noise mitigation in the specification. The work uses the idea of information feedback roadmaps to break the curse of history.

In contrast to the solution proposed in this paper, the approaches above do not consider dynamic environments. To deal with changing environments, reactive controllers have been proposed. Fu et al. [8] solve a two-player partially observable game with an adversarial environment, where the actions of the environment cannot be seen by the system. Although the system has incomplete information about the environment, the solution is computed based on a strategy using complete information. To reach states where a control is defined, the system uses a series of sensing actions to reduce the uncertainty until such states are reached. Chatterjee et al. [6] present finite-state controllers as a solution to POMDPs with parity objectives. To reduce the complexity, a series of heuristics are designed to find the solution. A practical case based on the results in [6] is presented in [18]. In this work a quadrotor performs a surveillance task while avoiding a ground vehicle. The motion of the quadrotor is considered deterministic as opposed to the stochastic motion considered in this paper.

To the best knowledge of the authors, we address for the first time the problem of computing optimal policies for mobile systems with uncertainty in motion and state information which follow temporal logic specifications and operate in dynamic environments. Rather than reacting to an adversarial environment as presented above, in the proposed method, the system reacts to static local targets and obstacles found during the execution of a plan such that the probability of satisfying an LTL specifications is maximised. Our method is based on the work in [20]. However, our solution permits the reaction of the system to local targets and obstacles unknown during the offline computation of the policy. To break the curse of history, we use feedback-based information roadmaps (FIRMs) to create a transition system by sampling the state space of

the system. Based on results in probabilistic model checking, we find an optimal solution to the problem by constructing a product MDP with the transition system and a Rabin automaton representing the LTL specification. In order to permit a fast reaction to the environment, a FIRM with edge's cost equal to the probability of violating the LTL specification is computed offline. This computation is possible due to the property that the cost of the edges of the FIRM are independent of each other. This FIRM is then used to drive the system from its current state to a sensed local target or to avoid obstacles while the probability of violating the specification is minimised. Hence, the main contribution of this paper is a sampling-based framework that permits systems with imperfect state information and motion uncertainty to react to detected obstacles and local targets in real-time while a LTL specification is satisfied.

The rest of the paper is organised as follows. Section 2 presents definitions of formalisms used in the rest of the paper and the problem formulation. Section 3 explains in detail the proposed method. Finally, a numerical example and conclusions are shown in Sect. 4 and Sect. 5, respectively.

2 Preliminaries and Problem Definition

2.1 System Model

This paper focuses on dynamic systems with motion and sensing uncertainty that evolve according to the following system model:

$$x_{k+1} = f(x_k, u_k, w_k) \ , \tag{1}$$

where $x \in X \subseteq \mathbb{R}^{d_x}$ is the system state, $u \in U \subseteq \mathbb{R}^{d_u}$ is the control input and w_k is the process noise at time k. We consider w_k as a zero-mean Gaussian noise with covariance Q_k. In partially observable systems, the system state is observed according to an observation model:

$$z_k = h(x_k, v_k) \ , \tag{2}$$

where $z_k \in Z \subset \mathbb{R}^{d_z}$ denotes the observation and v_k is a zero-mean Gaussian noise with covariance R_k at time k.

2.2 Belief Space

Since the state of the system is only partially known due to sensing uncertainty, the information available at each time k is a distribution over the set of possible states [16]:

$$b_k = Pr(x_k | z_k, u_{k-1}, z_{k-1}, \ldots, u_1, z_1, u_0, b_0) \ . \tag{3}$$

This distribution, called belief, compresses the history of observations $z_{0:k}$ and control actions $u_{0:k-1}$ taken from time 0 to time k and $k-1$, respectively. The updated belief for an applied control u_k and received observation z_{k+1} is given by:

$$b_{k+1} = \frac{Pr(z_{k+1}|x_{k+1})}{Pr(z_{k+1}|b_k, u_k)} \int_X b_k Pr(x_{k+1}|x_k, u_k) dx_k \ , \tag{4}$$

In a Gaussian belief space $\mathbb{B}$, the belief is characterised by the mean $\hat{x}$ and covariance P, i.e., $b_k = (\hat{x}_k, P_k) \in X \times \mathbb{S}_+^{d_x \times d_x}$, where $\mathbb{S}_+^{d_x \times d_x}$ represents the set of all possible positive semi-definite matrices with $d_x \times d_x$ entries.

2.3 Linear Temporal Logic

We use LTL to express system properties or desired behaviours. These properties are represented by a set Π of atomic propositions that indicate whether a property is true or false. A labelling function $L : x \to 2^{\Pi}$ maps the system state x to the set Π. Let $\boldsymbol{x} = x_0 x_1 \ldots$ be a sequence of states describing the behaviour of the system (1). A word $\omega = L(x_0)L(x_1)\ldots$ expresses this behaviour in terms of the atomic propositions.

Syntax: The syntax of LTL over Π is defined as follows:

$$\varphi := \pi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathscr{U} \varphi_2 \ ,$$

where $\pi \in \Pi$ is an atomic proposition; and $\neg$, $\vee$, $\wedge$, $\bigcirc$ and $\mathscr{U}$ represent the operators *negation*, *disjunction*, *conjunction*, *next* and *until*, respectively. The temporal operators *eventually* and *always* are defined as $\Diamond\pi = True\mathscr{U}\,\pi$ and $\Box\pi = \neg\Diamond\neg\pi$, respectively.

Semantics: The semantic of LTL formulae are defined with respect to infinite words over Π. Given an LTL specification φ, a sequence $\boldsymbol{x}$, and the satisfaction relation $\models$, we define the semantics inductively as follows: (i) $x_i \models \pi$ iff $\pi \in L(x_i)$; (ii) $x_i \models \varphi_1 \wedge \varphi_2$ iff $x_i \models \varphi_1$ and $x_i \models \varphi_2$; (iii) $x_i \models \varphi_1 \vee \varphi_2$ iff $x_i \models \varphi_1$ or $x_i \models \varphi_2$; (iv) $x_i \models \bigcirc\varphi$ iff $x_{i+1} \models \varphi$; and (v) $x_i \models \varphi_1 \mathscr{U} \varphi_2$ iff $\exists j \geq i : x_j \models \varphi_2$ and $x_k \models \varphi_1$, $\forall\, i \leq k < j$.

2.4 Deterministic Rabin Automaton

An LTL specification can be represented by a deterministic Rabin automaton (DRA), which accepts only words ω that satisfy the specification. A DRA $\mathscr{R}$ is a tuple $\mathscr{R} = (\Sigma, Q, q_0, \delta_{\mathscr{R}}, F)$, where: $\Sigma = 2^{\Pi}$ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta_{\mathscr{R}} : Q \times \Sigma \to Q$ is a transition function and $F = \{(L_1, K_1), \ldots, (L_r, K_r)\}$ is a set of pairs where $L_i, K_i \subseteq Q$ for all $i \in \{1, \ldots, r\}$.

A run on $\mathscr{R}$, produced by a word ω over the alphabet Σ, is a sequence $\rho = q_0 q_1 \ldots$ such that for every $i \geq 0$, there exists $\pi_i \in \Sigma$ and $\delta_{\mathscr{R}}(q_i, \pi_i) = q_{i+1}$. A run ρ is accepting if for a pair $(L_i, K_i) \in F$, the set L_i is intersected finitely many times while the set K_i is intersected infinitely many times. Figure 1 shows an example of a Rabin automaton.

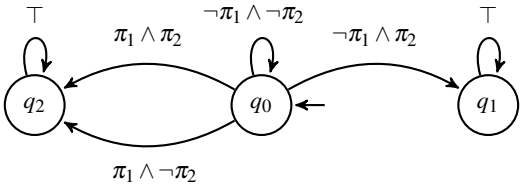


Fig. 1. Rabin automaton of LTL formula $\varphi = \neg\pi_2 \mathscr{U}\, \pi_1$, where $\pi_1, \pi_2 \in \Sigma$ are atomic propositions and $\mathscr{U}$ is the operator *until*. The formula indicates that the atomic proposition π_2 has to be avoided until π_1 is satisfied. The set F is formed by the pair $L = \{q_1\}$ and $K = \{q_2\}$. The arrow points to the initial state and $\top$ is unconditionally true.

2.5 Problem Formulation

Consider as an example a robot moving objects in a dynamically changing warehouse with two areas of interest, denoted by the atomic propositions π_1 and π_2, respectively. Using LTL, relevant behaviours can be specified. For example, the reachability formula $\Diamond\pi_1 \wedge \Diamond\pi_2$ can be used to indicate that the robot needs to eventually move an object to the areas π_1 and π_2. Safety formulae, e.g. $\Box\neg\pi_2$, indicates that certain properties remain invariant throughout the execution. In this work, we find a sequence of control inputs that maximises the probability of satisfying an LTL formula φ. Moreover, since the environment is dynamic, new local targets, e.g. objects in the warehouse, or obstacles can appear during the operation of the robot. Therefore, in addition to following the behaviour defined by φ, we allow the system to react to sensed local targets and obstacles in the environment while the probability of violating φ is minimised.

The labelling function L is used to identify the satisfaction of atomic propositions at each time k. That is, $L(x_k) = \pi_i$ if the system is in the region defined by π_i at time k. By labelling the system state at each time k, a word ω expressing the behaviour of the system in terms of the atomic propositions Π is obtained. Based on the definition of a Rabin automaton, a run $\boldsymbol{x} = x_0x_1\ldots$ of the system satisfies the specification φ if the word $\omega = L(x_0)L(x_1)\ldots$ is accepted by the Rabin automaton representing φ. Since the state is unknown in partially observable systems, instead of considering the state of the system to verify the satisfaction of the specification, we consider all the possible words generated during the transition between beliefs as presented in the next section. Now, we formally define the problem as follows.

Problem definition: Given a dynamic system with motion and sensing uncertainty of the form (1) and (2); and an LTL formula φ, compute a policy $\mu : \mathbb{B} \to U$ such that the probability of satisfying φ is maximised.

3 Solution

In this section, an overview of the proposed method is firstly presented followed by a detailed presentation. The main idea is to create a graph that represents the motion of the system in the environment. In this graph, vertices represent belief nodes and edges represent controllers that drive the system from one belief node to another, Figure 2. The graph is initialised with a single vertex, the initial belief of the system. Then, the graph is incrementally expanded by adding a new vertex that represents a new belief created by randomly sampling the state space of the system. After each expansion, it is verified whether there is a path such that the LTL specification is satisfied. If such a path does not exist, a new belief is added to the graph and the process is repeated until a path is found. Section 3.1 presents the computation of belief nodes and controllers. The expansion of the graph and the search of a path that satisfies the specification are explained in Sects. 3.2, 3.3 and 3.4. Because a dynamically changing environment is considered, the system must be able to react to local targets and obstacles. To allow fast reaction time to sensed objects, we precompute another graph, called FIRM, assigning to each edge, as the cost, the probability of violating the LTL specification in the transition. This FIRM is used to guide the system to the local targets or to avoid obstacles while the probability of violating the LTL specification is minimised, see Sects. 3.5 and 3.6.

3.1 Feedback-Based Information Roadmap

The main difficulty of solving POMDPs is the so-called curse of dimensionality. To alleviate this problem, we use feedback-based information roadmaps (FIRMs) [1]. FIRMs generalise probabilistic roadmaps (PRMs) [12] to account for motion and sensing uncertainty. In most of the works considering PRM-based methods and imperfect state information, each edge of the graph depends on the path traveled by the system, i.e., actions and observations taken from the initial belief, and therefore recalculation is necessary when the initial belief changes. In contrast, in a FIRM, each edge is independent of the others as a consequence of feedback controllers used to guarantee the convergency of the belief to predefined belief nodes. We exploit this property to perform most of the computation offline.

Without loss of generality, we use SLQG-FIRMs [1], where stationary linear quadratic Gaussian (SLQG) controllers are used as belief stabilisers. Any other type of controller can be used provided that the reachability of a belief is guaranteed. To construct a FIRM, a PRM is first constructed by sampling the state space of the system. Let $G = (V, E)$ represent the PRM, where V is the set of vertices (sampled states) $\nu \in X$ and E is the set of edges connecting the elements of V. Each node ν of the PRM is used to create a FIRM node as follows. First the system model (1) and observation model (2) are linearised with respect to a node ν resulting in the linear models:

$$x_{k+1} = A_\nu x_k + B_\nu u_k + w_k \ , \tag{5}$$

$$z_{k+1} = H_\nu x_k + v_k \ , \tag{6}$$

where $A_\nu \in \mathbb{R}^{d_x \times d_x}$, $B_\nu \in \mathbb{R}^{d_x \times d_u}$ and $H_\nu \in \mathbb{R}^{d_z \times d_x}$ are obtained through Jacobian linearisation.

A SLQG controller is designed to maintain the system state x as close as possible to ν while a Kalman filter is used to estimate the belief. Under the assumption that the pairs (A, B) and (A, H) are controllable and observable, respectively, the SLQG controller stabilises the system to an expected belief $b_\nu = (\nu, P_\nu)$, where the covariance P_ν can be determined offline for each node ν [1]. Hence, a belief node is defined as $\mathfrak{b} = \{b : \|b - b_\nu\|_b < \varepsilon\}$, where $\|\cdot\|_b$ is a suitable norm in $\mathbb{B}$ and ε determines the size of the belief node. Each node is associated with its SLQG controller, denoted by $\mu_\mathfrak{b}$, as belief stabiliser. The edges in E of the PRM are used to design time-varying LQG controllers that drive the system to the proximity of the FIRM nodes where the stabilisers can maintain the system within the nodes. Therefore, an edge between two FIRM nodes $\mathfrak{b}$ and $\mathfrak{b}'$ is formed by the combination of the time-varying LQG and the stabiliser controller and is denoted by $\mu_{\mathfrak{b},\mathfrak{b}'}$. A FIRM can be presented as a graph $\mathscr{G} = (\mathscr{B}, \mathscr{E})$, where $\mathscr{B}$ is the set of FIRM nodes and $\mathscr{E}$ is the set of controllers used as edges, Fig. 2.

In the next subsection, we use the procedure for creating FIRM nodes and controllers to incrementally construct a transition system on which a path satisfying the LTL specification is sought.

3.2 Incremental Transition System

Recall that using feedback controllers that guarantee the convergency of the belief to predefined belief nodes, the curse of dimensionality can be broken. Hence, we use the

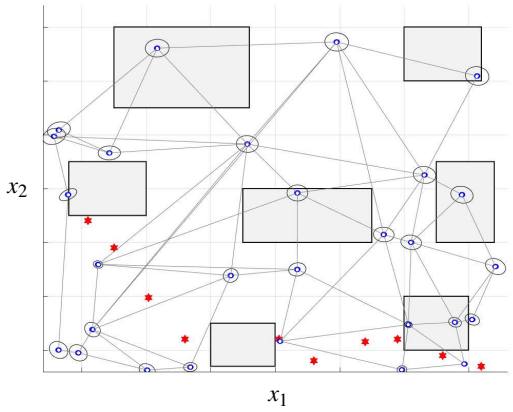


Fig. 2. FIRM created using a PRM on an environment in which the vertices represent the position (x_1, x_2) of the system. The grey rectangles and red stars are areas of interest and landmarks respectively. The landmarks are used by the system to localise itself. Hence, the uncertainty on the system state increases with the distance to the landmarks. The centre $b_v = (v, P_v)$ of the FIRM nodes is represented with a white disk and the 3σ ellipse (region where the true value lies with a probability of .988) of the associated covariances. The blue area around v denotes the part of the node corresponding to the mean $\hat{x}$, i.e., $\{\hat{x} : \|\hat{x} - v\| < \varepsilon\}$, where ε is a constant.

idea of FIRMs to create a transition system with the same property. In this subsection, an incremental construction of such a transition system is presented. A transition system is a tuple $\mathscr{T} = (\mathscr{B}_\mathscr{T}, \mathfrak{b}_0, \delta_\mathscr{T})$, where $\mathscr{B}_\mathscr{T}$ is a finite set of nodes $\mathfrak{b}_\mathscr{T}$, $\mathfrak{b}_0 \in \mathscr{B}_\mathscr{T}$ is an initial node and $\delta_\mathscr{T} \subseteq \mathfrak{b}_\mathscr{T} \times \mathfrak{b}'_\mathscr{T}$ is a transition relation.

Because the complexity of the problem depends on the number of nodes in the transition, the transition system is incrementally expanded by adding new nodes until the specification is satisfied. The transition system is constructed based on the idea of Rapidly-exploring Random Graphs (RRGs) [11] to allow satisfying words of infinite length and is constructed as follows, see Alg. 1. Initially, the transition system $\mathscr{T}$ includes only the initial node $\mathfrak{b}_0$ which contains the initial belief of the system. To add a new node, a state $v_{\text{sample}} \in X$ is sampled from the state space. This state is used to compute the FIRM node $\mathfrak{b}^{\text{new}}_\mathscr{T}$ including a belief stabiliser as presented in Sect. 3.1. Then, the closest node $\mathfrak{b}^{\text{near}}_\mathscr{T}$ in $\mathscr{B}_\mathscr{T}$ is sought. This process is repeated considering, in each iteration, the nearest nodes in $\mathscr{B}_\mathscr{T}$ in the half-space containing v_{sample} but not the previously considered nearest nodes $\mathfrak{b}^{\text{near}}_\mathscr{T}$. Once no more nodes are available, the new node $\mathfrak{b}^{\text{new}}_\mathscr{T}$ is added to $\mathscr{T}$ with the transitions $(\mathfrak{b}^{\text{new}}_\mathscr{T}, \mathfrak{b}^{\text{near},i}_\mathscr{T})$ and $(\mathfrak{b}^{\text{near},i}_\mathscr{T}, \mathfrak{b}^{\text{new}}_\mathscr{T})$, where i is the index of the nearest nodes found in the process described above. For each transition $(\mathfrak{b}_\mathscr{T}, \mathfrak{b}'_\mathscr{T}) \in \delta_\mathscr{T}$, the edge controller $\mu_{\mathfrak{b}_\mathscr{T}, \mathfrak{b}'_\mathscr{T}}$ is computed. This process continues until a path that satisfies the LTL specification is found, see Sect. 3.4.

In order to reduce the number of nodes in $\mathscr{T}$ and at the same time cover most of the workspace, a coarse partition is computed over the workspace. A segment of partition

Algorithm 1 Transition System Expansion

$\mathscr{B}_{\mathscr{T}} \leftarrow \mathfrak{b}_0$
while φ not satisfied **do**
  $\mathscr{X} \leftarrow X$, $i \leftarrow 1$
  Get a new sample state $v_{\text{sample}} \in X$
  Create node $\mathfrak{b}^{\text{new}}_{\mathscr{T}}$ with centre $b_v = (v_{\text{sample}}, P_v)$
  while $\mathscr{B}_{\mathscr{T}} \cap \mathscr{X} \neq \emptyset$ **do**
    Find closest node $\mathfrak{b}^{\text{near},i}_{\mathscr{T}} \in \mathscr{B}$ to $\mathfrak{b}^{\text{new}}_{\mathscr{T}}$ such that $\mathfrak{b}^{\text{near},i}_{\mathscr{T}} \cap \mathscr{X}$
    $i \leftarrow i+1$
    $\mathscr{X} \leftarrow \mathscr{X} \setminus H$, where H is the half-space containing $\mathfrak{b}^{\text{near},i}_{\mathscr{T}}$ but not $\mathfrak{b}^{\text{new}}_{\mathscr{T}}$
  $\mathscr{B}_{\mathscr{T}} \leftarrow \mathscr{B}_{\mathscr{T}} \cup \mathfrak{b}^{\text{new}}_{\mathscr{T}}$
  $\delta_{\mathscr{T}} \leftarrow \delta_{\mathscr{T}} \cup \mu_{\mathfrak{b}^{\text{new}}_{\mathscr{T}}, \mathfrak{b}^{\text{near},j}_{\mathscr{T}}} \cup \mu_{\mathfrak{b}^{\text{near},j}_{\mathscr{T}}, \mathfrak{b}^{\text{new}}_{\mathscr{T}}} \quad \forall j \in \{1,\ldots,i\}$

is randomly selected based on the number of samples associated with this segment. Then, a state is sampled uniformly such that $\Gamma(v_{\text{sample}})$ is constrained by the selected segment, where $\Gamma : X \to \mathbb{R}^{d_\Gamma}$ is the projection of the system state to the workspace.

Based on results from probabilistic verification [3], the product MDP of the transition system $\mathscr{T}$ and the Rabin automaton representing the LTL specification is computed and used to find a path such that the LTL specification is satisfied. The computation of this product MDP is presented in the next subsection.

3.3 Product MDP

A product MDP $\mathscr{P} = \mathscr{T} \times \mathscr{R}$ is a tuple $\mathscr{P} = (S, s_0, A, P, F_{\mathscr{P}})$, where: $S = \mathscr{B}_{\mathscr{T}} \times Q$ is a finite set of states, $s_0 = (\mathfrak{b}_0, q_0)$ is an initial state, A is a finite set of actions, $P(\cdot|\cdot,\cdot) : S \times S \times A \to [0,1]$ is the probability of transitioning to the state s' from the state s under action $a \in A$ and $F_{\mathscr{P}} = \{(L^{\mathscr{P}}_1, K^{\mathscr{P}}_1), \ldots, (L^{\mathscr{P}}_r, K^{\mathscr{P}}_r)\}$, where $L^{\mathscr{P}}_i = \mathscr{B}_{\mathscr{T}} \times L_i$ and $K^{\mathscr{P}}_i = \mathscr{B}_{\mathscr{T}} \times K_i$ for all $i \in \{1,\ldots,r\}$.

A run on $\mathscr{P}$ is defined as a sequence $\rho_{\mathscr{P}} = s_0 s_1 \ldots$, where $P(s_{i+1}|s_i, a) > 0$ for all $i \geq 0$. The set of actions A corresponds to the computed controllers associated with each transition in $\mathscr{T}$. Therefore, the set of actions available at state $s = (\mathfrak{b}_{\mathscr{T}}, q)$, denoted as $A(s)$, are the controllers computed for the transitions $(\mathfrak{b}_{\mathscr{T}}, \mathfrak{b}'_{\mathscr{T}}) \in \delta_{\mathscr{T}}$. The probability $P(s'|s, \mu_{\mathfrak{b}_{\mathscr{T}}, \mathfrak{b}'_{\mathscr{T}}})$, where $s = (\mathfrak{b}_{\mathscr{T}}, q)$ and $s' = (\mathfrak{b}'_{\mathscr{T}}, q')$, is the probability of ending on the DRA state q' starting from q when the transition $(\mathfrak{b}_{\mathscr{T}}, \mathfrak{b}'_{\mathscr{T}}) \in \delta_{\mathscr{T}}$ is performed using the control $\mu_{\mathfrak{b},\mathfrak{b}'} \in A(s)$.

Let $\boldsymbol{b} = b_0 b_1 \ldots b_n$ be the sequence of beliefs followed after applying $\mu_{\mathfrak{b}_{\mathscr{T}}, \mathfrak{b}'_{\mathscr{T}}}$, such that $b_0 \in \mathfrak{b}_{\mathscr{T}}$ and $b_n \in \mathfrak{b}'_{\mathscr{T}}$. To find the DRA state q' reached in $\mathscr{R}$ after the transition $(\mathfrak{b}_{\mathscr{T}}, \mathfrak{b}'_{\mathscr{T}}) \in \delta_{\mathscr{T}}$, the word ω produced by $\boldsymbol{b}$ is used as an input word in the DRA $\mathscr{R}$, starting from the state $q \in Q$. The last state of the run ρ on $\mathscr{R}$, produced by ω, is used as a state q' for the transition $s = (\mathfrak{b}_{\mathscr{T}}, q)$ to $s' = (\mathfrak{b}'_{\mathscr{T}}, q')$. As an example, consider the initial state q_0 of the Rabin automaton in Fig. 1 and assume that during the transition $(\mathfrak{b}_{\mathscr{T}}, \mathfrak{b}'_{\mathscr{T}})$ in $\mathscr{T}$ the words $\omega_1 = \{\neg\pi_1\neg\pi_2\}\{\neg\pi_1\neg\pi_2\}\{\pi_1\}$ and $\omega_2 = \{\neg\pi_1\neg\pi_2\}\{\neg\pi_1\neg\pi_2\}\{\pi_2\}$ are generated with probability 0.90 and 0.10, respectively. Therefore, the probability of transitioning from state $(\mathfrak{b}_{\mathscr{T}}, q_0)$ to $(\mathfrak{b}'_{\mathscr{T}}, q_2)$ is 0.90 and to $(\mathfrak{b}'_{\mathscr{T}}, q_1)$ is 0.10.

Recall that a specification is satisfied by the system if the word ω produces a run on $\mathscr{R}$ such that it visits finitely often times the set L_i and infinitely many times the set K_i, for $i \in \{1,\ldots,r\}$. Because during the transition s to s' in $\mathscr{P}$, more than one DRA state can be reached, in order to find a run on $\mathscr{P}$ satisfying a specification, each transition in $\mathscr{P}$ is associated with a probability of visiting a state in a pair $(L_i,K_i) \in F$. These probabilities are denoted as $P^{L_i}_{s,s'}$ and $P^{K_i}_{s,s'}$, respectively.

Computing probabilities of transitioning from s to $s' \in S$ is computationally expensive [1]. In this work, we approximate them using particle-based methods. The probability $P((\mathfrak{b}'_{\mathscr{T}},q')|(\mathfrak{b}_{\mathscr{T}},q),\mu_{\mathfrak{b}_{\mathscr{T}},\mathfrak{b}'_{\mathscr{T}}})$ is computed based on the number of particles that produced a word ω, during the transition $\mathfrak{b}_{\mathscr{T}}$ to $\mathfrak{b}'_{\mathscr{T}}$ under $\mu_{\mathfrak{b}_{\mathscr{T}},\mathfrak{b}'_{\mathscr{T}}}$ and starting from q, finishing in q'. A similar procedure is used to calculate the probability of intersecting the pairs $(L_i,K_i) \in F$ during the transition from s to s'.

The product MDP $\mathscr{P}$ is updated with each new node $\mathfrak{b}^{\text{new}}_{\mathscr{T}}$ added to $\mathscr{T}$. After each update, it is checked whether the LTL specification can be satisfied. In the next subsection, the computation of a policy $\mu_{\mathscr{P}} : S \to A$ in $\mathscr{P}$ that satisfies the LTL specification is presented. Using $\mu_{\mathscr{P}}$, a policy $\mu : \mathbb{B} \to U$ that solves the formulated problem is finally obtained.

3.4 Optimal Policy Computation

This subsection presents the calculation of the policy that maximises the probability of satisfying a LTL specification φ. A run $\rho_{\mathscr{P}} = s_0 s_1 \ldots$ on $\mathscr{P}$ is accepting if there exists a pair $(L^{\mathscr{P}}_i,K^{\mathscr{P}}_i) \in F_{\mathscr{P}}$ such that $L^{\mathscr{P}}_i$ and $K^{\mathscr{P}}_i$ are visited finitely and infinitely many times, respectively. Thus, we define an accepting end component (AEC) as follows. An AEC of $\mathscr{P}$ for a pair $(L^{\mathscr{P}}_i,K^{\mathscr{P}}_i) \in F_{\mathscr{P}}$ is a subgraph of $\mathscr{P}$ where each state is reachable from every other state, $P^{L_i}_{s,s'} = 0$ for all transitions and there exists a transition with $P^{K_i}_{s,s'} > 0$. After each increment of the transition system $\mathscr{T}$, the existence of an AEC is checked. Once an AEC is found, an optimal policy is computed.

It has been shown in probabilistic model checking that maximising the probability of reaching an AEC is equivalent to maximising the probability of satisfying φ [3]. A policy $\mu_{\mathscr{P}}(s)$ on $\mathscr{P}$, where $s = (\mathfrak{b}_{\mathscr{T}},q)$, induces a policy $\mu(\mathfrak{b}_{\mathscr{T}})$ on $\mathscr{T}$ by defining $\mu(\mathfrak{b}_{\mathscr{T}}) = \mu_{\mathscr{P}}(s)$. Hence, computing a policy on $\mathscr{P}$ that maximises the probability of reaching an AEC is equivalent to finding a policy on $\mathscr{T}$ that maximises the probability of satisfying the LTL specification. We use value iteration to compute the optimal policy by maximising the value function:

$$V(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a)V(s') \ , \tag{7}$$

$$\mu_{\mathscr{P}}(s) = \arg\max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a)V(s') \ , \tag{8}$$

for all $s \notin$ AEC and $V(s) = 1$ for all $s \in$ AEC.

Since the product MDP is updated with each addition of nodes to $\mathscr{T}$, the end components of $\mathscr{P}$ must be maintained after each update. The complexity of maintaining the end components on $\mathscr{P}$ is $O(|F||S|^{\frac{3}{2}})$ [20], where the number of states in S is

$|\mathscr{B}_{\mathscr{T}}| \times |Q|$. On the other hand, the running time of each iteration to find the optimal policy is $O(|S||A|^2)$ [14].

3.5 Local Targets

Approximating the probability of each transition on $\mathscr{P}$ using particle-based methods is in general a slow process [1, 20]. The construction and computation of a policy for $\mathscr{T}$ is computed offline and hence this slow task can be tolerated. Nevertheless, for fast reactions to targets or obstacles sensed in real-time, this long time is restrictive. To solve this problem, an offline computation of a FIRM is performed. In addition to permitting reactions in a short period of time, PRM-like structures such as FIRM can present better performance than methods using RRG techniques on difficult scenarios [10].

To maximise the coverage of the workspace and to obtain a dynamic FIRM (see Sect. 3.6), an offline partition of the environment is first created. In our method we used a grid-based partition. Then, the process of selecting and sampling in cells is performed similar to the process presented in Sect. 3.2. After a minimum number of samples on each cell are obtained, a FIRM $\mathscr{G} = (\mathscr{B}, \mathscr{E})$ is created as presented in Sect. 3.1.

When a local target is sensed by the system at time k, the FIRM is used to drive the system from its current belief b_k to a predefined service region of the local target while the specification is satisfied. To use the transition system and the FIRM, three aspects have to be considered: (i) the connection of the current belief to a node in FIRM; (ii) the optimal path in the FIRM; and (iii) the reconnection to $\mathscr{T}$ after the local target has been attended. This procedure is presented in Alg. 2.

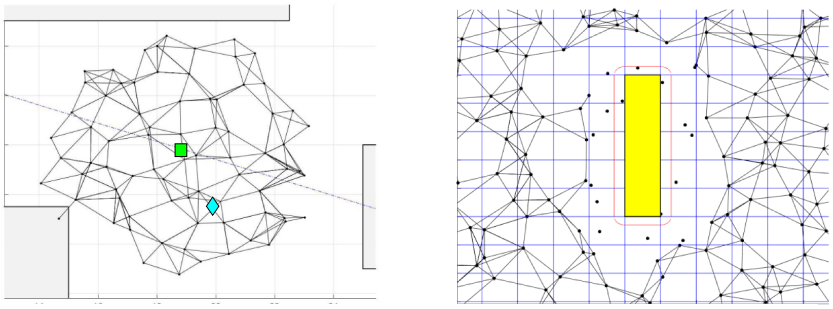
Algorithm 2 Path to Local Target

1: $\mathscr{G}' = (\mathscr{B}', \mathscr{E}')$, where $\mathscr{B}' = \{\mathfrak{b} \mid \mathfrak{b} \in \mathscr{B}, \|\Gamma_b(\mathfrak{b}) - \Gamma_b(b_k)\| \leq r\}$, $\Gamma_b : \mathbb{B} \to \Gamma(\hat{x})$ and $\mathscr{E}' \subset \mathscr{E}$
2: $\mathfrak{b}_{\text{near}} \leftarrow$ Nearest$(b_k, \mathscr{B}')$, $\mathfrak{b}_{\text{target}} \leftarrow$ Nearest$(target, \mathscr{B}')$
3: Apply $\mu_{\mathfrak{b}_{\text{near}}}$
4: $path \leftarrow$ OptimalPath$(\mathfrak{b}_{\text{near}}, \mathfrak{b}_{\text{target}})$
5: Follow $path$ applying edge controllers in $\mathscr{E}'$
6: $\mathfrak{b}_{\text{close}} \leftarrow$ Nearest$(\mathfrak{b}_{\mathscr{T}}, \mathscr{B}')$, where $\mathfrak{b}_{\mathscr{T}} \in \mathscr{B}_{\mathscr{T}}$ and $V(s) > 0$ such that $s = (\mathfrak{b}_{\mathscr{T}}, q)$
7: $path \leftarrow$ OptimalPath$(\mathfrak{b}_{\text{target}}, \mathfrak{b}_{\text{close}})$
8: Follow $path$ applying edge controllers in $\mathscr{E}'$
9: Apply $\mu_{\mathfrak{b}_{\mathscr{T}}}$

In the first step, when a local target is sensed by the system, a subgraph of the FIRM is created within the sensing area with radius r, Fig. 3(a). In this subgraph, the nearest FIRM node $\mathfrak{b}_{\text{near}}$ to the current belief b_k is sought. Then, the local stabiliser of $\mathfrak{b}_{\text{near}}$ is applied to drive the system to the FIRM node. Once the system is in the subgraph of the FIRM, an optimal path to the local target is computed. This path is optimal in terms of minimising the probability of violating the specification. To achieve this, it is necessary to verify which transitions of the FIRM do not violate the LTL specification. A similar problem has been solved in the literature for deterministic systems

with perfect state information [2, 19] using a monitor [4] which identifies if a specification has been satisfied or falsified as early as possible. In this work, since the state of the system is unknown, we use the Rabin automaton instead. Recall that in order to satisfy a specification, for a pair $(L_i, K_i) \in F$, the set L_i must be visited only finitely many times. Therefore, we calculate the probability of reaching states in L_i with a self transition, Fig. 1. Similar to the computation of $P^{L_i}_{s,s'}$ and $P^{K_i}_{s,s'}$ in $\mathscr{T}$, the probability of reaching such states starting in the Rabin state q during the transition from one node to another in the FIRM is computed during the FIRM construction. These probabilities are assigned as a weight on each transition on the FIRM. Since the probability of reaching a state L_i on a transition $(\mathfrak{b}, \mathfrak{b}')$ depends on the DRA state q, the current DRA state is tracked all the time during the online operation. Because all the transitions are precomputed offline, the only computation online is a shortest path graph search on the subgraph using the weights according to the current DRA state q. This problem can be solved efficiently by methods such as Dijkstra's algorithm, which has a time complexity $O(|\mathscr{B}|^2)$ [7].

After the computed path is followed, the last FIRM node in the path has to be connected to the transition system $\mathscr{T}$ in order to continue with the specification. This is achieved by searching the closest node $\mathfrak{b}_{\mathscr{T}}$ in $\mathscr{T}$ such that $V(s) > 0$ and $s = (\mathfrak{b}_{\mathscr{T}}, q)$, where q is the current $\mathscr{R}$ state after following the path in the FIRM. Once this state has been found, the closest node $\mathfrak{b}_{\text{close}}$ of the FIRM to $\mathfrak{b}_{\mathscr{T}}$ is sought. Then, the path in the FIRM is computed between the current node and $\mathfrak{b}_{\text{close}}$. After following the path, the stabiliser of the node $\mathfrak{b}_{\mathscr{T}}$ is applied to make the connection.



(a) (b)

Fig. 3. FIRM used to drive the system close to local targets or to avoid obstacles. (a) Subgraph of the FIRM within the sensing area of the system. The offline path obtained by solving the product MDP $\mathscr{P}$ is shown as a blue dotted line. The current belief and local target are represented by a green rectangle and blue diamond, respectively. (b) Subgraph of the FIRM without transitions affected by the obstacle. The obstacle and estimated position of it are shown with a yellow and red rectangle, respectively. The cells (shown in blue) occupied by the obstacle determine the invalid nodes and transitions of the FIRM.

3.6 Obstacle Avoidance

Similar to the local target case, the FIRM is used to avoid detected obstacles during the online operation. The main difference is that the presence of obstacles invalidates parts of the computed FIRM. Because edges of the FIRM are independent of each other, ideas from dynamic roadmaps [10, 13] can be applied.

Recall that the environment is partitioned into cells. Each of these cells is associated with FIRM nodes and transitions as follows. During the computation of the probabilities from node $\mathfrak{b}$ to $\mathfrak{b}'$, see Sect. 3.5, the probability of visiting a cell c_i during a transition can be computed. Let $p^k_{0:T_k}$ be the sample path of the k-th particle p from $\mathfrak{b}$ at time zero to $\mathfrak{b}'$ at time T^k. The probability of the system reaching a state such that $\Gamma(x)$ is on the cell c_i during the transition from $\mathfrak{b}$ to $\mathfrak{b}'$ is approximated by:

$$Pr_{\mathfrak{b},\mathfrak{b}'}(c_i) \approx \sum_{k=1}^{K} w^k \mathbb{1}_{c_i}(p^k_{0:T_k}) \ , \tag{9}$$

where w^k is a weight assigned to the particle p^k and $\mathbb{1}_{c_i}(\cdot)$ is an indicator that returns one, if a particle enters the cell c_i, and zero otherwise. Based on these probabilities, a cell is associated with the FIRM nodes $\mathfrak{b}, \mathfrak{b}'$ and its transition if $Pr_{\mathfrak{b},\mathfrak{b}'}(c_i) > 0$, Fig. 3(b).

When an obstacle is detected, the cells occupied by the obstacle are computed. Then, the nodes and transitions associated with these cells are invalidated from the FIRM. Since the current state of the system is uncertain, i.e., given by a mean and covariance over the belief space, the exact location of the obstacle cannot be determined by the system. To include the uncertainty on the obstacle's location, we compute the Minkowski sum of the detected obstacle and the contour of the 3σ ellipse of the current Gaussian. Assume that the system is transitioning between the nodes $\mathfrak{b}_{\mathscr{T}}$ and $\mathfrak{b}'_{\mathscr{T}}$ in $\mathscr{T}$ when an obstacle is detected. A subgraph of the FIRM is created within the sensing area as presented in Sect. 3.5. Note that this subgraph does not include any of the nodes affected by the estimation of the obstacle's location. In this subgraph, the closest node $\mathfrak{b}$ to the current belief b_k is sought. The stabiliser of $\mathfrak{b}$ is applied to drive the system to this node. Then, a path between $\mathfrak{b}$ and $\mathfrak{b}'$, the closest node to $\mathfrak{b}'_{\mathscr{T}}$, is computed using the weights as in the local target case. If, after applying the edge controllers of the computed path, the obstacle is still detected, a new subgraph is computed removing the invalid nodes. This process is repeated until the obstacle is not sensed. Then, the FIRM is connected to $\mathscr{T}$ as presented in Sect. 3.5. Algorithm 3 shows the procedure described above.

4 Example

In this section a numerical example is presented to illustrate the proposed method. We consider a robot in a workspace with 7 areas associated with the atomic propositions π_1, π_2, π_3 and π_4, Fig. 4. The mission of the robot is to visit the areas marked by the atomic proposition π_1, π_2 and π_3, in any order, while the areas marked with π_4 are avoided. Formally, this specification can be written as $\varphi = (\neg\pi_4 \mathscr{U} \pi_1) \wedge (\neg\pi_4 \mathscr{U} \pi_2) \wedge (\neg\pi_4 \mathscr{U} \pi_3)$. The example is implemented in MATLAB on a computer with a 3.1 GHz i7 processor and 8 GB of RAM.

Algorithm 3 Obstacle avoidance

while obstacle detected **do**
  $obstacleposition \leftarrow$ EstimatedPosition$(b_k, P_k, obstacle)$
  $C \leftarrow$ AffectedCells$(obstacleposition)$
  $\mathscr{G}' = (\mathscr{B}', \mathscr{E}')$, where $\mathscr{B}' = \{\mathfrak{b} \mid \mathfrak{b} \in \mathscr{B}, \|\Gamma_b(b_k) - \Gamma_b(\mathfrak{b})\| \leq r\}$, $\Gamma_b : \mathbb{B} \to \Gamma(\hat{x})$,
  $\mathscr{E}' \subset \mathscr{E} \setminus \mathscr{E}''$ and $(\mathfrak{b}, \mathfrak{b}') \in \mathscr{E}''$ iff $\exists\, c \in C$ s.t. $Pr_{\mathfrak{b},\mathfrak{b}'}(c) > 0$
  $\mathfrak{b}_{\text{near}} \leftarrow$ Nearest$(b_k, \mathscr{B}')$, $\mathfrak{b}_{\text{target}} \leftarrow$ Nearest$(\mathfrak{b}'_{\mathscr{T}}, \mathscr{B}')$
  $path \leftarrow$ OptimalPath$(\mathfrak{b}_{\text{near}}, \mathfrak{b}_{\text{target}})$
  Follow $path$ applying edge controllers in $\mathscr{E}'$
$\mathfrak{b}_{\text{close}} \leftarrow$ Nearest$(\mathfrak{b}_{\mathscr{T}}, \mathscr{B}')$, where $\mathfrak{b}_{\mathscr{T}} \in \mathscr{B}_{\mathscr{T}}$ and $V(s) > 0$ such that $s = (\mathfrak{b}_{\mathscr{T}}, q)$
$path \leftarrow$ OptimalPath$(\mathfrak{b}_{\text{target}}, \mathfrak{b}_{\text{close}})$
Follow $path$ applying edge controllers in $\mathscr{E}'$
Apply $\mu_{\mathfrak{b}_{\mathscr{T}}}$

The three-wheel omnidirectional mobile robot model presented in [1] is considered. For this robot (1) becomes:

$$f = \begin{pmatrix} -\frac{2}{3}\sin(\theta) & -\frac{2}{3}\sin(\frac{\pi}{3}-\theta) & \frac{2}{3}\sin(\frac{\pi}{3}+\theta) \\ \frac{2}{3}\cos(\theta) & -\frac{2}{3}\cos(\frac{\pi}{3}-\theta) & -\frac{2}{3}\cos(\frac{\pi}{3}+\theta) \\ \frac{1}{3l} & \frac{1}{3l} & \frac{1}{3l} \end{pmatrix} u + w \quad . \tag{10}$$

The state $x = [x_1, x_2, \theta]^T$ is composed of the robot position (x_1, x_2) and the orientation θ. The control input $u = [u_1, u_2, u_3]^T$ is formed of the linear velocities of each wheel. The distance of the wheels from the centre of the robot is denoted by l. The process noise w is a zero-mean Gaussian with covariance Q.

The robot uses landmarks, with known location on the workspace, to localise itself, Fig. 4. Let (LM_1^i, LM_2^i) denote the location of the i-th landmark; and η_r, σ_b^r, η_θ and σ_b^θ be constants. The observation model (2) with respect to the i-th landmark is expressed as:

$$z^i = [\|d^i\|, \text{atan2}(d_2^i, d_1^i) - \theta]^T + v^i \quad , \tag{11}$$

where $d = [x_1, x_2] - [LM_1^i, LM_2^i]$ and v^i is zero-mean Gaussian noise with covariance R:

$$R^i = \text{diag}((\eta_r \|d^i\| + \sigma_b^r)^2, (\eta_\theta \|d^i\| + \sigma_b^\theta)^2) \quad . \tag{12}$$

The results presented below were obtained from 20 simulations, but for the purpose of clarity, only one run is presented in Fig.4. In average (mean), the offline path is found in 82.46 seconds and the number of states in $\mathscr{T}$ and $\mathscr{P}$ are 31.61 and 284.5, respectively. The Rabin automaton $\mathscr{R}$ has 9 states and one pair (L, K) with $|L| = 1$ and $|K| = 1$. Computing the probability in each transition requires 0.536 seconds. The PRM used to create the FIRM has 1224 vertices, each vertex is connected to its seven nearest vertices. The FIRM requires on average 5022.61 seconds to be constructed. Since computing the probabilities for each edge of the FIRM is the most computationally demanding operation, the time to construct the FIRM could be reduced by limiting the number of edges on each vertex. Note that all the previous computations are performed offline. Finding a path in the FIRM, online, to reach the local target and to avoid the obstacle requires 0.097 and 0.698 seconds, respectively. Based on these results, it can be

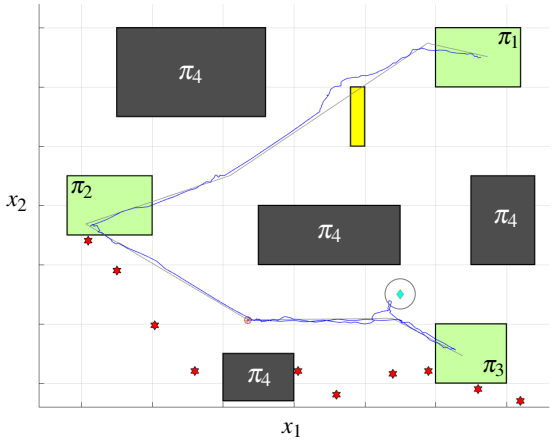


Fig. 4. Environment containing seven areas identified by the atomic proposition π_1, π_2, π_3 and π_4; a local target (blue diamond) with its service region (grey disk), an unknown obstacle (yellow rectangle) and ten landmarks (red stars). The objective of the robot is to visit the areas marked as π_1, π_2 and π_3 while areas π_4 have to be avoided. The grey line shows the path computed offline. The blue line shows a sample path of the system followed after detecting the local target and previously unknown obstacle. The initial position is marked by a red disk.

observed that computing a path in the FIRM to reach targets or avoid obstacles would require less time than expanding the transition system with the purpose of finding an alternative path.

5 Conclusions

In this paper we have introduced a new method to design control policies for mobile robots that can react to unknown environments under uncertainty in motion and sensing, while maximising the probability of satisfying high-level specifications. Although previous works have considered synthesis of controllers under uncertainty constraints and temporal logic specifications, reaction to unknown elements of the environment had not been considered under this scenario. An offline policy that maximises the probability of satisfying the specification is computed using an incrementally constructed transition system and a Rabin automaton. To achieve short reaction times, we precomputed a feedback-based information roadmap, considering the probability of violating the specification in each transition. Once the system finds an unknown element on the environment, the FIRM is used to reach or avoid this element. This task requires the connection of the current belief to the FIRM and the computation of a path that minimises the probability of violating the specification. Results show that using the FIRM requires less time than trying to find a path online by extending the transition system.

References

1. Agha-Mohammadi, A.A., Chakravorty, S., Amato, N.M.: FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. The International Journal of Robotics Research 33(2), 268–304 (2014)
2. Ayala, A.M., Andersson, S.B., Belta, C.: Temporal logic motion planning in unknown environments. In: Proc. of IROS. pp. 5279–5284. IEEE (2013)
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press Cambridge (2008)
4. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. In: Proc. of FSTTCS. vol. 20, pp. 1–68. ACM (2006)
5. Bry, A., Roy, N.: Rapidly-exploring random belief trees for motion planning under uncertainty. In: Proc. of ICRA. pp. 723–730. IEEE (2011)
6. Chatterjee, K., Chmelík, M., Gupta, R., Kanodia, A.: Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. In: Proc. of ICRA. pp. 325–330. IEEE (2015)
7. Cormen, T.H.: Introduction to algorithms. MIT press (2009)
8. Fu, J., Topcu, U.: Integrating active sensing into reactive synthesis with temporal logic constraints under partial observations. In: Proc. of ACC. pp. 2408–2413. IEEE (2015)
9. Horowitz, M.B., Wolff, E.M., Murray, R.M.: A compositional approach to stochastic optimal control with co-safe temporal logic specifications. In: Proc. of IROS. pp. 1466–1473. IEEE (2014)
10. Kallman, M., Mataric, M.: Motion planning using dynamic roadmaps. In: Proc. of ICRA. vol. 5, pp. 4399–4404. IEEE (2004)
11. Karaman, S., Frazzoli, E.: Sampling-based motion planning with deterministic μ-calculus specifications. In: Proc. of CDC/CCC. pp. 2222–2229. IEEE (2009)
12. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
13. Leven, P., Hutchinson, S.: Algorithmic and Computational Robotics: New Directions, chap. Toward real-time path planning in changing environments, pp. 363–376. A K Peters (2000)
14. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: Proc. of UAI. pp. 394–402. Morgan Kaufmann Publishers Inc. (1995)
15. Montana, F.J., Liu, J., Dodd, T.J.: Sampling-based stochastic optimal control with metric interval temporal logic specifications. In: Proc. of CCA. pp. 767–773. IEEE (2016)
16. Pineau, J., Gordon, G., Thrun, S., et al.: Point-based value iteration: An anytime algorithm for POMDPs. In: Proc. of IJCAI. vol. 3, pp. 1025–1032 (2003)
17. Prentice, S., Roy, N.: The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In: Robotics Research, pp. 293–305. Springer (2010)
18. Svoreňová, M., Chmelík, M., Leahy, K., Eniser, H.F., Chatterjee, K., Černá, I., Belta, C.: Temporal logic motion planning using POMDPs with parity objectives: case study paper. In: Proc. of HSCC. pp. 233–238. ACM (2015)
19. Vasile, C.I., Belta, C.: Reactive sampling-based temporal logic path planning. In: Proc. of ICRA. pp. 4310–4315. IEEE (2014)
20. Vasile, C.I., Leahy, K., Cristofalo, E., Jones, A., Schwager, M., Belta, C.: Control in belief space with temporal logic specifications. In: Proc. of CDC. pp. 7419–7424. IEEE (2016)
21. Wongpiromsarn, T., Frazzoli, E.: Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications. In: Proc. of CDC. pp. 7644–7651. IEEE (2012)