Iterative Temporal Motion Planning for Hybrid Systems in Partially Unknown Environments

Matthew R. Maly
mmaly@rice.edu

Morteza Lahijanian
morteza@rice.edu

Lydia E. Kavraki
kavraki@rice.edu

Hadas Kress-Gazit
hadaskg@cornell.edu

Moshe Y. Vardi
vardi@cs.rice.edu

ABSTRACT

This paper considers the problem of motion planning for a hybrid robotic system with complex and nonlinear dynamics in a partially unknown environment given a temporal logic specification. We employ a multi-layered synergistic framework that can deal with general robot dynamics and combine it with an iterative planning strategy. Our work allows us to deal with the unknown environmental restrictions only when they are discovered and without the need to repeat the computation that is related to the temporal logic specification. In addition, we define a metric for satisfaction of a specification. We use this metric to plan a trajectory that satisfies the specification as closely as possible in cases in which the discovered constraint in the environment renders the specification unsatisfiable. We demonstrate the efficacy of our framework on a simulation of a hybrid second-order car-like robot moving in an office environment with unknown obstacles. The results show that our framework is successful in generating a trajectory whose satisfaction measure of the specification is optimal. They also show that, when new obstacles are discovered, the reinitialization of our framework is computationally inexpensive.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms, Verification

Keywords

Motion Planning, Temporal Logic, Formal Synthesis

1. INTRODUCTION

In "classical" motion planning, robots with dynamics along with basic point-to-point robotic tasks are considered. These tasks are specified as, "go from A to B and avoid obstacles." To allow planning for more complex robots with complicated missions, various computational frameworks for planning with temporal logic specifications have been developed in the recent years (e.g., [10, 16, 17, 27, 32]). The increased expressivity as the result of the employment of temporal logics accommodates complex tasks that, for instance, require reaching one of a set of goals ("go to A or B"), visiting targets sequentially ("go to A, B, and C in this order"), and temporal conditions in reachability of targets ("first go to A or B and then eventually to C. If D is ever visited, then avoid B"). In this paper, we focus on planning with temporal goals for systems with general dynamics that can be realistically modeled as hybrid systems.

Most of the existing works in motion planning with temporal logic specifications consider static workspaces with full knowledge of their maps. Such assumptions, however, do not usually hold in real-world scenarios. For instance, a mobile robot in a warehouse setting may not be aware of a fallen box from a shelf that has blocked an aisle, or a mobile robot in an office environment may not know about the states of the office doors before its deployment. In such scenarios, it is reasonable to assume some information about the environment (e.g., the floor plan of the warehouse or the office building), but the motion-planning framework needs to have the capability of dealing with unforeseen obstacles in the environment. In fact, with complex specifications, it is imperative to consider the cases where the environment changes.

Recent works in synthesis-based approaches have begun to consider such cases (e.g., [25, 29]). In these works, synthesis involves the creation of a control strategy that can account for every possible environmental uncertainty. However, for the cases in which the number of environmental uncertainties is large, the problem becomes too complex. In these cases, it is advantageous to adopt an iterative temporal planning approach, in which online replanning is performed when unexpected environmental features are discovered. The difference between the above two approaches is discussed further in Section 1.1. The iterative planning approach poses the extra question of what to do in case the specification cannot be met due to newly discovered environmental restrictions. This gives rise to the need for a formal definition for a measure of satisfaction of a specification, a topic which is partially addressed in this paper.

Consider, for instance, a janitor robot in the office building whose schematic representation is shown in Figure 1. The office environment consists of a lobby, which includes an obstacle shown as a black rectangle, and five rooms each

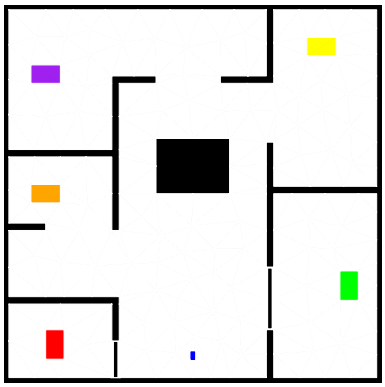


Figure 1: A schematic representation of an office building consisting of a lobby and five rooms. The lobby includes a black rectangular obstacle (center of figure), and each room has a door. The doors of three rooms are open and two are closed. The robot is shown as a blue rectangle in the lobby. The properties of interest in this environment are represented by the red, orange, purple, yellow, and green rectangles.

with a door. As the robot moves in the environment, its dynamics are subject to a set of restrictions associated with the features in the rooms (e.g., floor material and existence of sensitive objects). The properties of interest in this office environment are shown as orange, purple, red, green, and yellow rectangles. They represent office objects such as plants, a desk, a coffee maker, a blackboard, and a supply cabinet in the rooms. An example of a motion specification is as follows.

> "Visit the red region (to water the plants), go to the green region (to turn off the coffee maker), go to the yellow region (to clean the blackboard), and go to the purple region (to pick up the duster from supply cabinet) before visiting the orange region (to dust the desk). Do these tasks in any order and always avoid obstacles."

In this example, the robot initially has no knowledge of the obstacle in the lobby and the state of the office doors. It discovers them as it moves in the environment. Note that (for the instance captured in this figure) the robot will not be able to visit the red and green regions since the doors of their rooms are closed. Thus, there are parts of the above specification that cannot be satisfied. Nevertheless, for such tasks, we should allow the robot to continue with the mission even if it fails to satisfy parts of the specification due to unknown environmental constraints.

In this paper, we consider such realistic scenarios of planning in a partially unknown workspace for robots with complex and nonlinear dynamics. We focus on hybrid systems as they reveal the generality of our approach and lead to interesting application scenarios. Furthermore, we study the meaning of partial satisfaction of a temporal logic specification and how to measure it. Thus, the following problem emerges as a key challenge:

> "Given a mission expressed as a temporal logic specification for a mobile robot in a partially unknown workspace, find a plan for the robot to satisfy the mission's goals as closely as possible."

The main contribution of this work is a framework for iterative temporal motion planning. This framework allows for motion planning to be performed in a partially unknown environment. It enables a robot with nonlinear hybrid dynamics to modify its plan online, when it discovers obstacles that were not initially known, to satisfy a temporal logic specification without having to return to its original point and replan from scratch. The method is designed to avoid the recomputation of the automaton that is associated with the satisfaction of the temporal specification.

Another contribution of this paper is a scheme to measure closeness of satisfaction of a temporal logic specification and a method to maximize that measure. Thus, the framework can handle and provide guarantees of maximum satisfaction of the specification for the cases in which the discovered obstacles block regions of interest and render the temporal specification unsatisfiable. Then, instead of aborting the mission, the robot modifies its plan such that the measure of closeness of satisfaction of the specification is maximized. For a typical robotic task specification, optimizing this measure corresponds to satisfying as many requirements in the specification as the environment allows.

We tested this framework on a second-order car-like robot with hybrid dynamics in an office environment where regions of interest are known *a priori*. The robot was given a temporal specification requiring it to visit the regions of interest, without initial knowledge of obstacles that make some of the regions unreachable. In our simulation experiments, we varied the number of regions the robot should visit as well as the accuracy of the robot's initial map of the workspace. In all cases, our framework successfully found a trajectory for the robot to satisfy the specification as closely as possible.

The remainder of the paper is organized as follows. Section 1.1 contains related work. Section 2 describes our hybrid robot model and the type of temporal logic that we use. Section 3 details the problem we consider and gives a high-level overview of our approach to solving it. We present our iterative temporal motion-planning framework in Section 4. In Section 5, we introduce our simulation experiment and demonstrate results. The paper concludes with final remarks and a discussion of future work in Section 6.

1.1 Related Work

Much work has been done toward the problem of planning for robotic systems to satisfy high-level temporal logic specifications. For general robot models with nonlinear dynamics, static workspaces, and temporal goals, motion planning approaches have been proposed to solve the problem using deterministic μ-calculus specifications [14] and co-safe LTL specifications [3–5, 27].

Synthesis-based approaches to such problems require strong assumptions on the robot's dynamics and a construction of a finite discrete model for the motion of the robot in its workspace with a simulation relation to the continuous model. A provably correct hybrid controller can then be generated as a state machine that encodes the robot actions necessary to satisfy the task [18]. The synthesis of this hybrid controller requires time and space polynomial in the size of the reachable state space of the system; this is often called the state explosion problem [19]. One way to address this problem is to use a coarser abstraction, which requires a stronger assumption on controllers. Other suggestions include receding horizon techniques [32]. Work has also been

done to address the issue of controller uncertainty, modeling the robot as a Markov decision process [9, 22, 23].

The issue of synthesis from high-level specifications in an unknown or dynamic environment has been studied both for abstract systems (e.g., [6]) and specifically for robotics (e.g., [7, 15, 25, 29]). If the geometry of the environment changes, whether due to an unknown region becoming reachable [29] or a known region becoming unreachable [25], then the hybrid controller must be updated to incorporate the change. As global resynthesis of the hybrid controller is expensive, there exist approaches to locally patch the controller to incorporate the changes in less time; still, initial work in this area has shown that patching the hybrid controller can still require significant time to complete [25]. In addition, the work in [12] addresses the issue of motion planning for a mobile robot to move from a start region to a goal region, where secondary regions of interest can potentially be discovered along the way. When a secondary target is discovered, the robot replans a new trajectory to visit the target along its way to the goal region.

Our work is most closely related to [3–5, 27] in that we are taking a motion-planning approach instead of using synthesis. Synthesis is a process that generates a strategy to ensure system correctness in all possible scenarios. Such approaches require knowledge of all possible environmental uncertainties. However, in real-world applications, it may not be feasible to obtain enough information of the environment for synthesis. Also, when there are too many unknowns, synthesizing a plan can be unsuccessful. In such cases, iterative planning methods are more natural and viable to employ since they do not require a full knowledge of the environmental restrictions. Therefore, in this paper, we propose an online iterative planning approach to deal with partially unknown environments. Rather than accounting for everything that can go wrong, we plan based on what we know and deal with new restrictions only when they are discovered. In other words, we plan a trajectory given the currently known state of the environment. During execution of the trajectory, if an unforeseen problem is encountered, we replan a new trajectory from the current state on-the-fly. This framework is inspired by replanning scenarios in robotics [1, 2].

A key advantage of our approach lies in the high-level structure through which we guide a low-level continuous motion planner. This high-level structure is a product of the abstraction of the hybrid system and an automaton that derives from the temporal logic specification. The abstraction is achieved quickly by a geometric partition of the robot's workspace. However, the automaton from the specification can be very expensive to compute. By keeping the automaton and the abstraction separate, we prevent changes to the environment from requiring us to recompute the automaton. Changes to the environment simply require modifications to the decomposition of the workspace and, hence, to the abstraction, which is an inexpensive operation to perform. This is in contrast to other works that use synthesis approaches to plan for robots to satisfy temporal logic specifications. In these works, typically the task specification and assumptions on the environment must be encoded into a single hybrid controller that can be expensive to change [18, 25]. Another advantage of our framework is the use of motion planning, which supports systems with any type of high-dimensional hybrid (possibly nonlinear) dynamics. Synthesis-based approaches deal with a restricted class of robot systems that typically involve linear dynamics. When the dynamics of the system are sufficiently complex, it is difficult (if not impossible) to synthesize provably correct controllers [5].

The issue of what to do when a specification is determined to be unsatisfiable has been explored before. In [28], an algorithm was defined to report a reason as to why a GR(1) LTL specification is unrealizable. The work in [15] presents a method of changing an unsatisfiable nondeterministic Büchi automaton into the "closest" satisfiable one, where all actions of the robot are represented using a finite state machine. Our approach to unsatisfiable specifications differ in that we do not change the automaton; instead, we provide a simple metric to define partial satisfaction that is meaningful for an interesting set of scenarios.

2. PRELIMINARIES

2.1 Robot Hybrid Model

In this paper, we consider a general mobile robot whose dynamics are subject to restrictions in the regions of a partially unknown environment. We describe its motion in such an environment by the hybrid system $H = (S, s_0, \text{INV}, \text{SENSE}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW}, \Pi, L)$, where

- $S = Q \times X$ is the hybrid state space that is a product of a set of discrete modes, $Q = \{q_1, q_2, \ldots, q_m\}$ for some finite $m \in \mathbb{N}$, by a set of continuous state spaces $X = \{X_q \subseteq \mathbb{R}^{n_q} : q \in Q\}$;
- $s_0 \in S$ is the initial state;
- $\text{INV} = \{\text{INV}_q : q \in Q\}$, is the set of invariants, where $\text{INV}_q : X_q \to \{\top, \bot\}$;
- $\text{SENSE} : X_q \to \{\top, \bot\}$, is the sensing function that returns *true* if an unknown obstacle is detected;
- $E \subseteq Q \times Q$ describes discrete transitions between modes in Q;
- $\text{GUARD} = \{\text{GUARD}_{q_i,q_j} : (q_i, q_j) \in E\}$, where $\text{GUARD}_{q_i,q_j} : X_{q_i} \times \{\top, \bot\} \to \{\top, \bot\}$ is a guard function that enables transitions between different modes given the continuous state of the robot and the unknown-obstacle detector readings (i.e., output of SENSE);
- $\text{JUMP} = \{\text{JUMP}_{q_i,q_j} : (q_i, q_j) \in E\}$, where $\text{JUMP}_{q_i,q_j} : X_{q_i} \to X_{q_j}$ is the jump function. In this paper, we assume that each JUMP_{q_i,q_j} is the identity function;
- $U = \{U_q \subset \mathbb{R}^{m_q} : q \in Q\}$ is the set of input spaces;
- $\text{FLOW} = \{\text{FLOW}_q : q \in Q\}$, where $\text{FLOW}_q : X_q \times U_q \times \mathbb{R}^{\geq 0} \to X_q$ is the flow function that describes the continuous dynamics of the system through a set of differential equations;
- Π is a set of atomic propositions;
- $L : S \to 2^{\Pi}$ is a labeling function assigning to each hybrid state possibly several elements of Π.

A pair $s = (q, x) \in S$ denotes a hybrid state of the system. $\text{FLOW}_q(x, u, t)$ gives the continuous state of the system when the input u is applied for t time units starting from state x.

2.2 Syntactically Co-safe LTL

We use syntactically co-safe LTL to write the specifications of robotic tasks. Its syntax and semantics are defined below.

DEFINITION 1 (SYNTAX). *Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ be a set of Boolean atomic propositions. A syntactically co-safe LTL formula over Π is inductively defined as following:*

$$\phi := \pi \mid \neg\pi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathcal{X}\phi \mid \phi \mathcal{U} \phi \mid \mathcal{F}\phi$$

where $\pi \in \Pi$, $\neg$ (negation), $\vee$ (disjunction), and $\wedge$ (conjunction) are Boolean operators, and $\mathcal{X}$ ("next"), $\mathcal{U}$ ("until"), and $\mathcal{F}$ ("eventually") are temporal operators.

DEFINITION 2 (SEMANTICS). *The semantics of syntactically co-safe LTL formulas are defined over infinite traces over 2^{Π}. Let $\sigma = \{\tau_i\}_{i=0}^{\infty}$ with $\tau_i \in 2^{\Pi}$ be an infinite trace and $\sigma^i = \tau_i, \tau_{i+1}, \ldots$ and $\sigma_i = \tau_0, \tau_1, \ldots, \tau_{i-1}$. σ_i is a prefix of the trace σ. $\sigma \models \phi$ indicates that σ satisfies formula ϕ and is recursively defined as following:*

- *$\sigma \models \pi$ if $\pi \in \tau_0$;*
- *$\sigma \models \neg\pi$ if $\pi \notin \tau_0$;*
- *$\sigma \models \phi_1 \vee \phi_2$ if $\sigma \models \phi_1$ or $\sigma \models \phi_2$;*
- *$\sigma \models \phi_1 \wedge \phi_2$ if $\sigma \models \phi_1$ and $\sigma \models \phi_2$;*
- *$\sigma \models \mathcal{X}\phi$ if $\sigma^1 \models \phi$;*
- *$\sigma \models \phi_1 \mathcal{U} \phi_2$ if $\exists k \geq 0$, s.t. $\sigma^k \models \phi_2$, and $\forall i \in [0, k)$, $\sigma^i \models \phi_1$;*
- *$\sigma \models \mathcal{F}\phi$ if $\exists k \geq 0$, s.t. $\sigma^k \models \phi$.*

An important property of syntactically co-safe LTL formulas is that, even though they have infinite-time semantics, finite traces are sufficient to satisfy them. Hence, syntactically co-safe LTL is an appropriate specification language to describe robotic tasks which are required to be realized in finite horizon.

From a syntactically co-safe LTL formula ϕ, a deterministic finite automaton (DFA) can be constructed that accepts precisely all of the formula's satisfying finite traces [21]. Such a DFA is given by a tuple $\mathcal{A}_\phi = (Z, \Sigma, \delta, z_0, F)$, where

- Z is a finite set of states;
- $\Sigma = 2^{\Pi}$ is the input alphabet, where each input symbol is a truth assignment to the propositions in Π;
- $\delta : Z \times \Sigma \to Z$ is the transition function;
- $z_0 \in Z$ is the initial state;
- $F \subseteq Z$ is the set of accepting states.

A finite run of $\mathcal{A}_\phi$ is a sequence of states $\omega = \omega_0\omega_1 \ldots \omega_n$, where $\omega_0 = z_0$ and $\omega_i \in Z$ for $i = 1, \ldots, n$. ω is called an accepting run if $\omega_n \in F$. An input trace that realizes an accepting run is a ϕ-satisfying trace.

3. PROBLEM DESCRIPTION AND OVERALL APPROACH

In this paper, we consider a mobile robot with complex and possibly nonlinear hybrid dynamics moving in an environment. Some of the features and obstacles in the environment may not be known before the deployment of the robot. We make the natural assumption that the robot can detect an unknown obstacle when it is within the robot's obstacle-detector range. This is an assumption commonly made in related work [2]. The regions in the environment impose different sets of restrictions on the dynamics of the robot. Each point in the environment holds a set of properties (propositions). Let Π denote the set of all environmental propositions. We assume that while the robot has full information of the propositions and their locations in the environment, it has only partial *a priori* knowledge of the regions (obstacles) of the environment. We are interested in deploying this robot in such a partially unknown environment with temporal logic specifications.

Due to possible unknown obstacles in the environment, the satisfaction of the specification cannot be guaranteed. Nevertheless, we do not want the robot to abort the mission if it realizes that fragments of the specification cannot be met. Instead, we require the robot to satisfy the specification as closely as possible. We envision many scenarios where this can be an advantageous approach (e.g., the janitor robot example in Section 1). We formally define and discuss the definition of satisfying a specification as closely as possible below and in Section 4.3. We now focus on the following problem.

PROBLEM: *Given a partially unknown environment and a task specification expressed as a syntactically co-safe LTL formula ϕ over Π, find a robot motion plan that satisfies ϕ as closely as possible.*

We model the motion of the robot in the environment as a hybrid system (Section 2.1). This allows us to capture the changes in the robot dynamics by the transitions between the modes of the hybrid system. We assume that the robot moves in a two-dimensional environment. We choose the continuous states of the hybrid system in mode q, $x \in X_q$, such that its first two components refer to the position of the robot. Let $Pr(A)$ denote projection of a Euclidean set A onto $\mathbb{R}^2$. We define the workspace of the robot as $W = \{(q, W_q) : W_q = Pr(X_q), q \in Q\}$ and denote the set of (polygonal) obstacles in mode q by $W_{q,obs}$. The robot has partial *a priori* knowledge of the obstacles in its workspace. Thus, $W_{q,obs} = W^k_{q,obs} \cup W^u_{q,obs}$ where $W^k_{q,obs}$ and $W^u_{q,obs}$ refer to the sets of known and unknown obstacles in mode q, respectively. We also assume that the robot can detect an unknown obstacle when it comes within some proximity of it. This is represented by SENSE in the hybrid model. We establish the relationship between the hybrid state of the robot $s = (q, x)$ and its workspace by function $h_H : S \to W$. Similarly, given $\mathrm{w} \in W$, we define $h_H^{-1}(\mathrm{w}) = \{s \in S : h_H(s) = \mathrm{w}\}$. We choose Π as the set of atomic propositions for the hybrid system and assign them to the hybrid states according to their positions in the workspace. Hence, the problem is now reduced to designing a motion plan for H that satisfies ϕ.

We employ a multi-layered synergistic framework [4, 27] to solve the motion planning problem by using the initial knowledge of the workspace. The framework consists of three main layers: a high-level search layer, a low-level search layer, and a synergy layer that facilitates the interaction between the high-level and the low-level search layers (see Figure 2). The high-level planner uses an abstraction of the hybrid system and the specification formula ϕ to suggest high level plans. The low-level planner uses the dynamics of the hybrid system and the suggested high-level plans to explore the state-space for feasible solutions. In our work, the low-level layer is a sampling-based planner and does not assume the existence of a controller [26].

To satisfy a specification in an undiscovered environment, an iterative high-level planner is employed. That is, every time an unknown obstacle is encountered, the high-level planner modifies the coarse high-level plan online by accounting for the geometry of the discovered obstacle, the path traveled to that point, and the remaining segment of the specification that is yet to be satisfied. This replanning

is achieved in four steps. First, a "braking" operation is applied to prevent the robot from colliding with the newly discovered obstacle. Simultaneously, a new abstraction of the hybrid system is computed through a new decomposition of the modified environment map (workspace) [3]. Next, the traveled path is mapped on the new abstraction model. Finally, a new satisfying plan is generated as a continuation of the explored portion of the old plan. Thus, the robot does not need to reinitialize (return to its starting point) every time it encounters an unknown environmental feature. Moreover, the robot's progress in satisfying the specification is preserved. This iterative motion-planning framework is discussed in detail in Section 4.

Recall that from ϕ, a DFA can be constructed that accepts all of the formula's satisfying finite traces [21,27]. We use this DFA to design a satisfying high-level plan. We also utilize the DFA to define a metric to measure the "distance-to-satisfaction" of a specification in cases in which the specification is unsatisfiable. This measure is used to produce a high-level plan that satisfies the specification as closely as possible. We formally define this metric in Section 4.3.

In general, a contingency maneuver can be used instead of a "braking" operation as the first step of the approach. Our framework is by no means limited to a stopping maneuver, and the exploration for the "best" contingency plan is left for future work. Moreover, it is important to note that our method of generating a new high-level plan is fast. This is due to the following two reasons: (1) we are not recomputing the DFA, which does not need to change since the specification formula does not change following discovery of an obstacle, and (2) we generate the abstraction of the hybrid system by decomposing the workspace through triangulation, which has been shown to be computationally inexpensive [5]. For instance, the computation time for high-level replanning for the janitor robot example moving in the office environment shown in Figure 1 is in the order of a fraction of a second.

4. PLANNING FRAMEWORK

In this section, we describe our iterative planning framework, which consists of three main layers: a high-level planner, a low-level search layer, and a synergy layer as shown in Figure 2. The high-level planner generates a set of coarse satisfying plans by searching over a structure called a product automaton (Section 4.2). This structure is the product of discrete abstraction $\mathcal{M}$ of the hybrid system (Section 4.1) and automaton $\mathcal{A}_\phi$ corresponding to the formula ϕ. Each of these plans is a sequence of the states of the product automaton which can be mapped back to the states of $\mathcal{M}$. The low-level search layer produces continuous trajectories that follow a satisfying high-level plan. This is achieved by expanding a sampling-based motion tree in the direction of a suggested high-level plan in the hybrid state-space. The synergy layer facilitates the two-way interaction between the high-level and the low-level search layers (Sections 4.2 and 4.3). Algorithm 1 contains the framework pseudocode; it relies on subroutines detailed in Algorithms 2, 3, and 4.

4.1 Abstraction

To produce a high-level plan, we first abstract the hybrid system H to a discrete model $\mathcal{M} = (D, d_0, \rightarrow_D, \Pi, L_D)$, where D is a set of discrete states, $d_0 \in D$ is the initial state, $\rightarrow_D \subseteq D \times D$ is the transition relation, and $L_D : D \rightarrow 2^\Pi$ is a labeling function. We refer to the model $\mathcal{M}$ as the *abstraction* of the hybrid system. To construct $\mathcal{M}$, we first partition each workspace W_q (for each discrete mode q) into a set of *regions* (i.e. $W_q = \bigcup_{i=1}^{N_r^q} r_i^q$). Specifically, we use a geometry-based conforming Delaunay triangulation of W_q that respects the propositional regions and the boundaries of the obstacles.

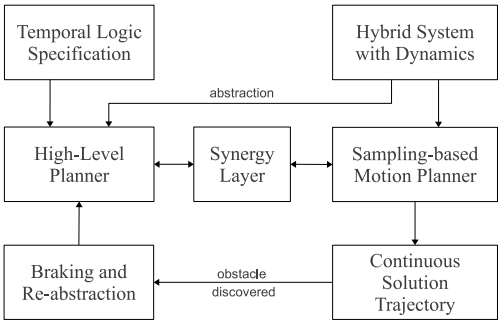


Figure 2: Multi-layered synergistic motion planning framework.

Recall from Section 3 that $W_q = Pr(X_q)$, where X_q is the domain of the continuous states of the hybrid system in mode q. Thus, the decomposition of W_q induces a partition in the hybrid state space. Let S_i^q denote the set of all hybrid states that correspond to the region r_i^q in W_q (i.e. $S_i^q = \{(q,x) \in S \mid x \in Pr^{-1}(r_i^q),\ r_i^q \subseteq W_q,\ q \in Q\}$). Then,

$$S = \bigcup_{q \in Q} \bigcup_{i=1}^{N_r^q} S_i^q.$$

We associate a unique discrete state $d \in D$ to each S_i^q. We model this correspondence with a family of maps $\{\Upsilon_q : S^q \rightarrow D \mid S^q = q \times X_q,\ q \in Q\}$; then the previous sentence can be written as $\Upsilon_q(S_i^q) = d$. Moreover, D can be written as $D = \cup_{q \in Q}\{\Upsilon_q(S_i^q) \mid 1 \leq i \leq N_r^q\}$.

We construct the transition relation $\rightarrow_D$ to include geometric adjacencies between regions of a given workspace as well as adjacencies between discrete modes enabled by GUARD functions of the hybrid system. Specifically, for each pair of geometrically adjacent regions r_j^q and r_k^q in W_q, we add the transition $\left(\Upsilon_q(S_j^q), \Upsilon_q(S_k^q)\right)$ to $\rightarrow_D$; furthermore for each pair of sets S_l^q and $S_m^{q'}$ between which a discrete jump is possible, we add the transition $(\Upsilon_q(S_l^q), \Upsilon_q(S_m^{q'}))$ to $\rightarrow_D$. All the hybrid states in S_i^q have the same labels since the triangulation of each workspace W_q respects the propositional regions. Hence, the labeling function L_D corresponds to the labeling function L from the definition of H; that is, $L_D(\Upsilon_q(S_i^q)) = L(s)$ for every $s \in S_i^q$. For further details, we refer the reader to our previous work [3].

It should be noted that the initial construction of $\mathcal{M}$ is based on the initial knowledge of the environment map. As the robot discovers unknown obstacles, the map is updated and a new abstraction is generated. Given that this method is based on a triangulation of a two-dimensional space, obtaining a new abstraction is fast. Furthermore, we initially assume transitions between all adjacent partitions of the workspace are realizable even though the dynamics of the robot may prevent some transitions. This does not create a

problem in our planning framework because the synergistic framework will bias its discrete search against unrealizable transitions. In fact, one of the advantages of our planning framework is that it does not require a bisimilar abstraction and, hence, allows for inexpensive and fast construction of an approximate abstraction model.

4.2 Initializing the Product Automaton

The structure we use to guide the tree of system trajectories is a product automaton, which is computed as $\mathcal{P} = \mathcal{M} \times \mathcal{A}_\phi$. In line 2 of Algorithm 1, we compute the minimal DFA $\mathcal{A}_\phi$ corresponding to the formula ϕ [21, 24]. Though this translation can require time doubly exponential with respect to the number of propositions in ϕ, we only compute $\mathcal{A}_\phi$ once, so the translation can be seen as an offline step. We refer to elements of $\mathcal{P}$ as *high-level states*. $\mathcal{P}$ is a directed graph in which there exists an edge from high-level state (d_1, z_1) to (d_2, z_2) iff d_1 and d_2 are adjacent in $\mathcal{M}$ and $\mathcal{A}_\phi.\delta(z_1, \mathcal{M}.L_D(d_2)) = z_2$, where $\mathcal{A}_\phi.\delta$ is the deterministic transition function for $\mathcal{A}_\phi$. The latter condition means that there exists a transition in $\mathcal{A}_\phi$ from z_1 to z_2 whose label is satisfied by the set of propositions $\mathcal{M}.L_D(d_2)$ that hold true at d_2. We call a high-level state $(d, z) \in \mathcal{P}$ an *accepting* (or *goal*) state iff z is an accepting state in $\mathcal{A}_\phi$.

For each high-level state $(d, z) \in \mathcal{P}$, we assign a weight defined by

$$w(d, z) = \frac{(\textsc{cov}(d, z) + 1) \cdot \textsc{vol}(d)}{\textsc{DistFromAcc}(z) \cdot (\textsc{numsel}(d, z) + 1)^2} \qquad (1)$$

where $\textsc{cov}(d, z)$ is the number of tree vertices (generated by the low-level planner) associated with (d, z) (an estimate of coverage), $\textsc{vol}(d)$ is the area of the workspace corresponding to the abstraction state d, $\textsc{DistFromAcc}(z)$ is the minimum distance from automaton state z to an accepting state in the automaton, and $\textsc{numsel}(d, z)$ is the number of times (d, z) has been selected for tree expansion in line 2 of Algorithm 4. Then, to each directed edge $e = ((d_1, z_1), (d_2, z_2))$ in $\mathcal{P}$, we assign the weight

$$w(e) = \frac{1}{w(d_1, z_1) \cdot w(d_2, z_2)}. \qquad (2)$$

The values in (1) and (2) are continually updated as the planning framework progresses. For example, whenever the low-level planner creates a tree vertex associated with a high-level state (d, z), the value of $\textsc{cov}(d, z)$ is incremented by one. The estimates in (1) and (2) have been shown to work well in previous work [4]. In general, a weighing scheme that incorporates more than just number-of-edge distance is useful to promote expansion in unexplored areas (i.e., where cov and numsel are both small) and to discourage expansion in areas where attempts at exploration have repeatedly failed (i.e., where numsel $\gg$ cov).

4.3 Planning

Once the product automaton has been computed, line 4 of Algorithm 1 computes a trajectory for the system that satisfies the formula ϕ as closely as possible. The details of this approach are given in Algorithm 2. Many details are similar to the framework discussed in past works [3–5]. We differ from them by allowing for online replanning in light of newly discovered obstacles in Algorithm 1, and by partially satisfying an unsatisfiable specification when computing a lead in Algorithm 3.

The core loop of our planning algorithm is shown in lines 4, 5, and 6 of Algorithm 2. The subroutine ComputeLead in Algorithm 3 creates leads that reach as close as possible to an accepting state. Each lead computed in line 4 is a suggested sequence of contiguous high-level states through which Explore attempts to guide the tree of motions.

Measure of Satisfiability.

We present a measure of satisfiability that uses the graph-based distance to an accepting state in the DFA. Each high-level state (d, z) is annotated with the graph-based distance value $\textsc{DistFromAcc}(z)$ corresponding to the automaton state z. Our framework computes trajectories that end in a high-level state (d_g, z_g) such that $\textsc{DistFromAcc}(z_g)$ is minimized. The function DistFromAcc is an intuitive measure on the automaton that translates to a reasonable high-level plan for many formulas that we have encountered, such as the example specification in Section 1. For such a specification, a trajectory that minimizes DistFromAcc takes the robot to all reachable regions of interest, while a non-optimal trajectory with respect to DistFromAcc would miss some reachable regions. The topic of "approximating" temporal properties is a subject of ongoing research. Generally, it requires making the satisfaction relation quantitative rather than qualitative. For example, the satisfaction value can be an arbitrary lattice element rather than a Boolean value; cf. [20]. In addition, the authors in [31] describe a synthesis algorithm to minimize quantitative satisfaction error given a set of contradictory specifications.

Algorithm 1 Framework for planning for hybrid systems with LTL specifications in a partially unknown environment

Input: A robot model described by a hybrid system $H = (S, s_0, \textsc{inv}, \textsc{sense}, E, \textsc{guard}, \textsc{jump}, U, \textsc{flow}, \Pi, L)$, a bounded workspace $W \subset \mathbb{R}^2$, a set of initially known obstacles $O \subset W$, a co-safe LTL formula ϕ defined over $H.\Pi$, and a time bound $t_{\max}$.

Output: Returns `true` if successful in moving the robot through the workspace to satisfy ϕ; returns `false` otherwise.

```
1: M ← ComputeAbstraction(W, O, H.Π, H.L)
2: A_φ ← ComputeMinDFA(φ, W, H.L)
3: P ← ComputeProduct(M, A_φ, H.Π, H.L)
4: {x_i}_{i≥0} ← Plan(H, W, O, H.Π, H.L, P, t_max)
5: j ← 1
6: while j < |{x_i}| do
7:    Move system from state x_{j−1}.s to state x_j.s
8:    if H.sense(x_j.s) = ⊤ then
9:       Apply braking operation to reach stopped robot state s′
10:      H.s_0 ← s′
11:      Add discovered obstacle to O
12:      M ← ComputeAbstraction(W, O, H.Π, H.L)
13:      P ← ComputeProduct(M, A_φ, H.Π, H.L)
14:      {x_i}_{i≥0} ← Plan(H, W, O, H.Π, H.L, P, t_max)
15:      if Plan was unsuccessful then
16:         return false
17:      j ← 1
18:   j ← j + 1
19: return true
```

ComputeLead computes a lead that ends in a high-level state (d, z) such that the number of transitions from z to an accepting state in the automaton, given by $\textsc{DistFromAcc}(z)$, is minimized. If the specification ϕ is satisfiable in the current environment, then $\textsc{DistFromAcc}(z) = 0$, i.e., (d, z) is an accepting state. On the other hand, if the specification ϕ is unsatisfiable, then z is as close as possible to an accepting state in the automaton. In many cases, there are multiple candidate high-level states that tie under the DistFromAcc

metric. To break ties, we choose the high-level state with minimal edge-weight distance from the starting high-level state, using the edge-weight function defined in (2).

The subroutine ComputeAvailableCells in line 5 of Algorithm 2 computes the set of high-level states that exist in the current lead and are nonempty. A high-level state (d, z) is *nonempty* if there exists at least one tree vertex associated with it, i.e., if $\text{COV}(d, z) > 0$. In the first few iterations of Plan, the only nonempty high-level state will be (d_0, z_0), where d_0 is the abstraction region containing the initial system state $H.s_0$, and z_0 is the initial state of the DFA. As the algorithm progresses, the tree planner reaches more high-level states, and the set C of available cells grows larger. To promote progress, we favor high-level states that are closest to the end of the lead. Specifically, moving backwards along the lead, for each nonempty high-level state (d, z) we encounter, we add (d, z) to the set C of available cells and then quit early with probability 0.5.

The subroutine Explore, given in Algorithm 4, corresponds to the low-level search layer of our framework. This function promotes tree expansion in high-level states from the set C. In line 2 of Explore, a high-level state (d, z) is sampled from C with probability $w(d, z)/\sum_{(d',z')\in C} w(d', z')$. Then, in line 3, we perform one iteration of the low-level tree planner to promote expansion from the set of tree vertices associated with the high-level state (d, z) and obtain a new tree vertex v. Any tree-based motion planner can be used in this step; in our approach, we are using an EST-like approach [13]. If z is an accepting state of the automaton, then v is returned as the endpoint of a solution trajectory. This trajectory is reconstructed by Plan in line 8 by following parent vertices back to the root of the tree. Otherwise, if the new vertex v corresponds to a newly reached high-level state that is in the current lead, then the high-level state is added to the set of available cells in line 8 of Explore to be considered in the future.

4.4 Discovering an Obstacle and Replanning

Once a system trajectory that satisfies ϕ is computed, we begin moving the robot along the trajectory. At each state in the trajectory, we query the robot's range sensor in line 8 of Algorithm 1. We assume that the robot's range sensor checks for obstacles within radius ρ of the center of the robot and reports a polygonal model of any previously unknown obstacle that it finds. If no new obstacles are discovered along the trajectory, then the robot reaches the final state of the planned trajectory and stops, having completed its mission. If an obstacle is discovered by the range sensor from some state s along the trajectory, then we apply a braking operation to the robot to reach some stopped state s'. The braking operation should respect the dynamics of the system. In the general case, the robot should perform a contingency maneuver to avoid the newly discovered obstacle [1, 11]. The radius ρ of the range sensor is assumed to be large enough for the braking or contingency maneauver to safely be performed. Once the braking maneuver is complete, we recompute the discrete abstraction $\mathcal{M}$ to ignore the new obstacle, recompute the product automaton $\mathcal{P}$, and replan a trajectory from s', following the same planning approach described in Section 4.3. Once a new trajectory is found by the planner, we resume moving the robot from s' along the new trajectory. It is important to note that because we recompute the entire discrete abstraction from scratch, all of the previous edge weights in $\mathcal{P}$ are lost and recomputed in the next planning iteration.

Algorithm 2 Plan: Temporal planning algorithm for hybrid systems

Input: A robot model described by a hybrid system $H = (S, s_0, \text{INV}, \text{SENSE}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW}, \Pi, L)$,
a bounded workspace $W \subset \mathbb{R}^2$,
a set of known obstacles $O \subset W$,
a product automaton $\mathcal{P}$,
and a time bound $t_{\max}$.

Output: Returns a sequence of triplets, each containing hybrid system state, control, and corresponding high-level state, representing a system trajectory that satisfies the specification. Reports an error and aborts if no such trajectory could be found within time $t_{\max}$.

1: $\mathcal{T} \leftarrow$ InitializeTree(s_0)
2: `solved` $\leftarrow$ `false`
3: **while** Time Elapsed $< t_{\max}$ **do**
4: $K = ((d_1, z_1), \ldots, (d_k, z_k)) \leftarrow$ ComputeLead($\mathcal{P}, H.s_0$)
5: $C \leftarrow$ ComputeAvailableCells(K)
6: $v \leftarrow$ Explore($H, W, O, \mathcal{T}, C, K, \mathcal{P}, \Delta t$)
7: **if** $v \neq$ `NULL` **then**
8: Follow v.`parent` to construct trajectory $\{x_i\}_i$
9: **return** $\{x_i\}_i$
10: Report unsuccessful and exit

Algorithm 3 ComputeLead: Subroutine to compute high-level guides

Input: A product automaton $\mathcal{P}$ (product of DFA and workspace decomposition) and a starting high-level state (d_0, z_0).

Output: Returns a lead, which is a sequence of high-level states beginning with the given start (d_0, z_0) and ending as close as possible to an accepting state.

1: $F \leftarrow \arg\min_{(d,z)\in\mathcal{P}}\{\text{DistFromAcc}(z)\}$
2: Run Dijkstra's all-pairs shortest-path algorithm on $\mathcal{P}$ with source (d_0, z_0); store parent map `parent` and weight map `weight`
3: $(d_g, z_g) \leftarrow \arg\min_{(d,z)\in F}\{\texttt{weight}[(d, z)]\}$
4: Construct lead $K = ((d_0, z_0), \ldots, (d_g, z_g))$ using `parent` map
5: **return** K

5. EXPERIMENTS

To test our approach, we have created an experiment for a second-order car with hybrid dynamics to explore an office-like environment. The full map of the office is shown in Figure 3(a). Geometrically, the robot is modeled as a rectangle with length $l = 0.2$ and width $w = 0.1$. The robot state has a continuous component (x, y, θ, v, ψ), which includes the planar position $(x, y) \in [0, 10]^2$, heading $\theta \in [-\pi, \pi]$, forward velocity $v \in [-1/6, 1]$, and steering angle $\psi \in [-\pi/6, \pi/6]$. The robot state also includes a discrete component $g \in \{1, 2, 3\}$, corresponding to the gear of the car. The car is controlled with the input pair $u = (u_0, u_1)$, where u_0 is the forward acceleration and $u_1 \in [-\pi/18, \pi/18]$ is the steering angle velocity. Given the current gear g, we bound the acceleration input so that $u_0 \in [-1/6, g/6]$. The dynamics of the car are given by $\dot{x} = v\cos(\theta)$, $\dot{y} = v\sin(\theta)$, $\dot{\theta} = v\tan(\psi)/l$, $\dot{v} = u_0$, and $\dot{\psi} = u_1$. We model each gear as a separate discrete mode of the hybrid system. We define guards and jumps on the dynamics of the robot so that the robot switches gears as follows. If, when in gear $g < 3$, the car achieves velocity $v > g/6$, then the car switches to gear $g + 1$. If, when in gear $g > 1$, the car achieves velocity

Algorithm 4 EXPLORE: Tree-exploration subroutine

Input: A robot model described by a hybrid system $H = (S, s_0, \text{INV}, \text{SENSE}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW}, \Pi, L)$,
a bounded workspace $W \subset \mathbb{R}^2$,
a set of known obstacles $O \subset W$,
a tree of motions $\mathcal{T}$,
a set of available high-level states C,
a lead K,
a product automaton $\mathcal{P}$,
and an exploration time Δt.

Output: Returns a tree vertex that reaches the goal high-level state if one was found; returns `NULL` otherwise.

```
1: while TIME ELAPSED < Δt do
2:    (d, z) ← C.sample()
3:    v ← SELECTANDEXTEND(T, H, (d, z), W, O, P, PROP)
4:    if v.z ≠ ∅ then
5:       if v.z.isAccepting() then
6:          return v
7:       if (v.d, v.z) ∉ C ∧ (v.d, v.z) ∈ L then
8:          C ← C ∪ {(v.d, v.z)}
9: return NULL
```

$v < (g-1)/6$, then the car switches to gear $g-1$. Immediately following a gear switch, the acceleration input bounds are updated accordingly. Furthermore, we define guards in the office so that in small rooms (rooms containing the red and orange propositions), the car is restricted to first gear. In larger rooms (rooms containing the yellow, purple, and green propositions), the car is restricted to first and second gears. Specifically, the guards in the rooms prevent the robot from switching outside of the allowable gears by restricting the robot's velocity, so that the guard conditions to switch gears are never satisfied. The car is given a sensing radius of 1. If, when executing a solution trajectory, it discovers a new obstacle within its sensing radius, the car switches to an "emergency" mode in which it applies an emergency deceleration sufficient to reduce its velocity to $\epsilon > 0$ before colliding with the obstacle. It then rebuilds the abstraction, recomputes the product automaton, switches out of the emergency mode into the mode corresponding to gear $g = 1$, and computes a new trajectory to follow.

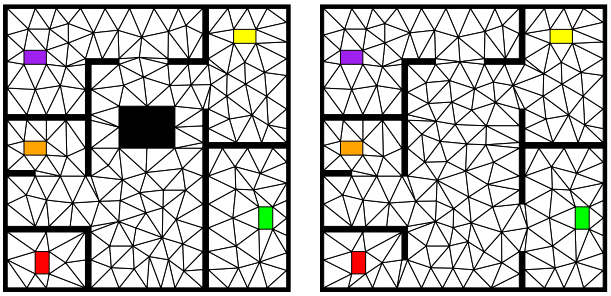
In this experiment, the robot is asked to visit N randomly chosen regions of interest in any order, where $N \in \{1, \ldots, 5\}$. Formally, the robot is given the co-safe LTL specification

$$\phi_N = \bigwedge_{i=0}^{N-1} \mathcal{F} p_i, \tag{3}$$

where each p_i corresponds to a propositional region in the office environment.

The robot's initial map includes the walls of the office. However, the robot does not know the current status of the doors into each room, nor does it know whether there are any obstacles in the central lobby. Specifically, the robot is unaware that the doors to two of the rooms are closed (we model this as rectangular obstacles filling the doorways), and there is a large rectangular obstacle in the center of the lobby. Figure 3(a) contains the actual map of the office, and Figure 3(b) contains the robot's initial map. We include the triangulations in the maps in Figure 3 to demonstrate granularity. A triangulation always respects the currently known obstacles and the geometry of the propositional regions.

We have implemented our framework and experiments in C++ using the Open Motion Planning Library (OMPL) [8]. For the co-safe LTL formulas considered in our experiments, we have converted them to minimal DFA's by using scheck [24]. To triangulate environments, we use Triangle [30]. All experiments were run on the Shared University Computing Grid at Rice. Each experiment used a 2.83 GHz Intel Xeon processor with 16 GB RAM. For each set of input parameters, we average our timing measurements over 50 independent runs.



(a) (b)

Figure 3: (a) an office-like environment with propositional regions of interest; (b) the robot's initial map, in which 3 obstacles are unknown.

Table 1: Experimental data for office experiment with a full initial map and a partial initial map

Initial Map	N	Solution Time	Time Computing Product $\mathcal{P} = \mathcal{M} \times \mathcal{A}_\phi$
Full	1	1.36	0.003
	2	4.05	0.003
	3	11.19	0.004
	4	21.49	0.006
	5	34.85	0.008
Partial	1	3.78	0.009
	2	19.32	0.037
	3	54.83	0.088
	4	257.82	0.206
	5	549.86	0.415

Table 1 contains experimental data for satisfying the coverage formula ϕ_N in the office environment, comparing the full initial map (Figure 3(a)) to a partial initial map (Figure 3(b)). With a fully accurate initial map, the robot does not encounter any unanticipated obstacles, and so our method behaves equivalently to the past method presented in [3–5]. We are including data for the full initial map for comparison. For the partial map, planning times increase significantly with the number of regions of interest in the coverage formula. Visiting more regions causes the robot

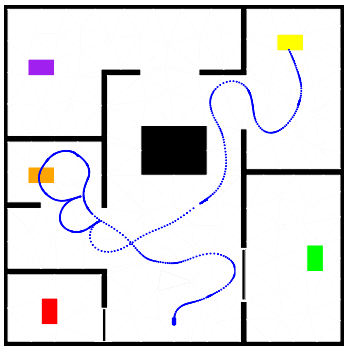
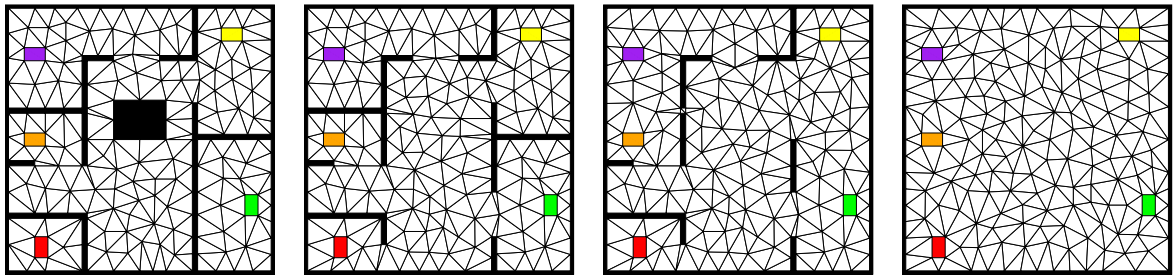


Figure 4: A sample trajectory that satisfies the specification "Visit the green, orange, and yellow regions in any order" as closely as possible.



(a) (b) (c) (d)

Figure 5: Office-like environments in which (a) all obstacles are known; (b) 3 obstacles are unknown; (c) 6 obstacles are unknown; (d) all 16 obstacles are unknown.

Table 2: Experimental data for office experiment with formula ϕ_5 when varying the initial map

Number of Initially Unknown Obstacles	Solution Time	Time Computing Product $\mathcal{P} = \mathcal{M} \times \mathcal{A}_\phi$
0	34.85	0.008
3	594.86	0.415
6	1430.71	0.952
16	3001.96	1.966

to discover more unknown obstacles, each of which requires the robot to brake. Every time the robot comes to a stop near a newly discovered obstacle, planning a solution trajectory from that stopped point is often time-consuming for the low-level motion-planning layer. This is due to the close proximity of the robot and the obstacle. With longer-range sensors, this problem can be alleviated. For all experiments, times spent recomputing the product automaton $\mathcal{P}$ remain very small (see Tables 1 and 2). It should be noted that the times spent recomputing $\mathcal{P}$ in Tables 1 and 2 also include the time to regenerate the abstraction $\mathcal{M}$. Figure 4 contains an example trajectory for the robot, given a specification to visit three regions of interest (green, orange, and yellow), one of which is unreachable (green) due to a closed door. First, the robot drives toward the room containing the green region. When it encounters the door, it brakes and recomputes the abstraction and the product automaton. The planning framework uses our measure of satisfiability to generate another trajectory that satisfies the specification as closely as possible, which is to visit the two remaining regions. A similar replanning step occurs when the robot encounters the large obstacle in the lobby. When the robot is in the room with the orange region, it stays only in first gear as required.

To test the importance of the initial map, we have also run experiments with initial maps of varying accuracy, ranging from a completely known environment to a completely unknown environment in which no obstacles or walls are initially known by the robot (except for the bounding box of the environment). The four types of initial maps are shown in Figure 5. As before, all propositional regions are initially known. Table 2 contains data for this set of experiments. We focus on solving ϕ_5, the most difficult of the formulas to consider. As seen in Table 2, the time spent building and recomputing $\mathcal{P}$ and $\mathcal{M}$ is negligible compared to the time spent planning solution trajectories.

6. CONCLUSION

In this paper, we have presented an iterative motion planning framework for a hybrid system with complex and possibly nonlinear dynamics given a temporal logic specification and a partially unknown environment. We have also presented a metric of satisfiability which we can optimize in cases where obstacles in the environment prevent full satisfaction of the given temporal logic specification; in such cases, the robotic system satisfies the specification as closely as possible.

For future work, we plan to change the replanning step to retriangulate only the part of the workspace that has changed upon discovery of a new obstacle, instead of retriangulating the entire workspace. This would allow the framework to keep many of the edge weights in the product automaton, so that not all information from the synergy layer is lost. In addition, we plan to add support for obstacles to disappear from the robot's initial map (the current framework only supports obstacles appearing). We could also assume a probabilistic distribution on where and when obstacles will appear, and then generate trajectories that maximize probability of successful satisfaction of the specification. Finally, we would like to consider a "greedy" temporal motion planning approach that begins executing a partial trajectory along a lead in the product automaton. This is to prevent the framework from wasting time generating an entire solution trajectory for a large specification, only to discover an obstacle early in that trajectory, stop, and recompute another solution trajectory. As seen in our experiments, when the robot's initial map is sufficiently inaccurate, this wasted planning time can add up.

7. ACKNOWLEDGMENTS

The authors would like to acknowledge Devin Grady, Ryan Luna, and Mark Moll of Rice University for their helpful feedback and suggestions. Work on this project by Maly, Kavraki, Kress-Gazit, and Vardi has been supported in part by NSF Expeditions 1139011. Maly, Lahijanian, Kavraki, and Vardi have also been supported in part by NSF CCF 1018798. Kavraki has been supported in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under grant number W911NF-09-1-0383. This work was also supported in part by the Shared University Grid at Rice funded by NSF under Grant EIA-0216467 and a partnership between Rice University, Sun Microsystems, and Sigma Solutions, Inc.

8. REFERENCES

[1] K. E. Bekris, D. K. Grady, M. Moll, and L. E. Kavraki. Safe distributed motion coordination for second-order systems with different planning cycles. *Intl. J. of Robotics Research*, 31(2):129–149, Feb. 2012.

[2] K. E. Bekris and L. E. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *IEEE Intl. Conf. on Robotics and Automation*, pages 704–710, 2007.

[3] A. Bhatia, L. Kavraki, and M. Vardi. Motion planning with hybrid dynamics and temporal goals. In *Decision and Control, IEEE Conf. on*, pages 1108–1115, 2010.

[4] A. Bhatia, L. Kavraki, and M. Vardi. Sampling-based motion planning with temporal goals. In *Robotics and Automation, IEEE Int. Conf. on*, pages 2689–2696, May 2010.

[5] A. Bhatia, M. Maly, L. Kavraki, and M. Vardi. Motion planning with complex goals. *Robotics Automation Magazine, IEEE*, 18(3):55 –64, Sep. 2011.

[6] R. Bloem, K. Greimel, T. Henzinger, and B. Jobstmann. Synthesizing robust systems. In *Formal Methods in Computer-Aided Design*, pages 85–92, 2009.

[7] Y. Chen, J. Tumova, and C. Belta. LTL robot motion control based on automata learning of environmental dynamics. In *Robotics and Automation, IEEE Int. Conf. on*, pages 5177 –5182, May 2012.

[8] I. A. Şucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19:72–82, December 2012.

[9] X. Ding, S. Smith, C. Belta, and D. Rus. MDP optimal control under temporal logic constraints. In *Decision and Control and European Control Conf. (CDC-ECC), IEEE Conf. on*, pages 532–538, 2011.

[10] G. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45:343–352, 2009.

[11] T. Fraichard. A short paper about motion safety. In *Proc. 2007 IEEE Intl. Conf. on Robotics and Automation*, pages 1140–1145, Apr. 2007.

[12] D. K. Grady, M. Moll, C. Hegde, A. C. Sankaranarayanan, R. G. Baraniuk, and L. E. Kavraki. Multi-objective sensor-based replanning for a car-like robot. In *IEEE Intl. Symp. on Safety, Security, and Rescue Robotics*, 2012.

[13] D. Hsu, J. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Intl. J. of Computational Geometry and Applications*, 9(4-5):495–512, 1999.

[14] Karaman and Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conf. on Decision and Control*, 2009.

[15] K. Kim and G. Fainekos. Approximate solutions for the minimal revision problem of specification automata. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*, pages 265–271, 2012.

[16] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 53(1):287–297, 2008.

[17] H. Kress-Gazit, G. Fainekos, and G. Pappas. Where's waldo? Sensor-based temporal logic motion planning. In *Robotics and Automation, 2007 IEEE Int. Conf. on*, pages 3116 –3121, Apr. 2007.

[18] H. Kress-Gazit, G. Fainekos, and G. Pappas. Temporal-logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 25(6):1370–1381, Dec. 2009.

[19] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu. Correct, reactive, high-level robot control. *Robotics Automation Magazine, IEEE*, 18(3):65 –74, Sep. 2011.

[20] O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213, 2007.

[21] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19:291 – 314, 2001.

[22] M. Lahijanian, S. B. Andersson, and C. Belta. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409, Apr. 2012.

[23] M. Lahijanian, J. Wasniewski, S. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees,. In *IEEE Int. Conf. on Robotics and Automation*, pages 3227–3232, Alaska, 2010.

[24] T. Latvala. Efficient model checking of safety properties. In *Model Checking Software*, pages 74–88. Springer, 2003.

[25] S. C. Livingston, R. M. Murray, and J. W. Burdick. Backtracking temporal logic synthesis for uncertain environments. In *IEEE Intl. Conf. on Robotics and Automation*, pages 5163–5170, 2012.

[26] E. Plaku, L. Kavraki, and M. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Trans. on Robotics*, 26(3):469–482, Jun. 2010.

[27] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Falsification of LTL safety properties in hybrid systems. In *Proc. of the Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, York, UK, 2009.

[28] V. Raman and H. Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP. In *Proc. of the 23rd Int. Conf. on Computer Cided Verification*, CAV'11, pages 663–668, Berlin, Heidelberg, 2011. Springer-Verlag.

[29] S. Sarid, B. Xu, and H. Kress-Gazit. Guaranteeing high-level behaviors while exploring partially known maps. In *Proc. of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[30] J. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, chapter 23, pages 203 – 222. Springer-Verlag, Berlin/Heidelberg, 1996.

[31] P. Černý, S. Gopi, T. A. Henzinger, A. Radhakrishna, and N. Totla. Synthesis from incompatible specifications. In *Proc. of the tenth ACM Int. Conf. on Embedded software*, EMSOFT '12, pages 53–62, New York, NY, USA, 2012. ACM.

[32] T. Wongpiromsarn, U. Topcu, and R. Murray. Receding horizon control for temporal logic specifications. In *Proc. of the 13th ACM Int. Conf. on Hybrid Systems: Computation and Control*, pages 101–110. ACM, 2010.