In IEEE Conference on Decision and Control, 2010, Atlanta, GA, USA

# Motion Planning with Hybrid Dynamics and Temporal Goals

Amit Bhatia Lydia E. Kavraki Moshe Y. Vardi

***Abstract*— In this paper, we consider the problem of motion planning for mobile robots with nonlinear hybrid dynamics, and high-level temporal goals. We use a multi-layered synergistic framework that has been proposed recently for solving planning problems involving hybrid systems and high-level temporal goals. In that framework, a high-level planner employs a user-defined discrete abstraction of the hybrid system as well as exploration information to suggest high-level plans. A low-level sampling-based planner uses the dynamics of the hybrid system and the suggested high-level plans to explore the state-space for feasible solutions. In previous work, we have proposed a geometry-based approach for the construction of the discrete abstraction for the case when the robot is modeled as a continuous system. Here, we extend the approach for the construction of the discrete abstraction to the case when the robot is modeled as nonlinear hybrid system. To use the resulting abstraction more efficiently, we also propose a lazy-search approach for high-level planning that reduces the size of the search space by reusing previously constructed high-level plans for initializing the search. Our proposed techniques result in computational speedups of close to 10 times over other possible approaches for second-order nonlinear hybrid robot models in challenging workspace environments with obstacles and for a variety of temporal logic specifications.**

## I. Introduction

The problem of constructing motion plans for robots has traditionally involved the task of taking a robot from a given initial state to a state in a goal region while avoiding obstacles. A class of algorithms that have been particularly successful in solving such problems for robot models with differential constraints are the sampling-based algorithms [1], [2]. Such algorithms typically construct a tree of feasible trajectories for the system using an incremental simulator for the dynamics. The geometric constraints arising due to finite geometry of the robot and the obstacles in the workspace, are handled using a collision-detection scheme in the search procedure. The algorithms relax strong completeness guarantees to efficiently solve problem instances involving complex robot dynamics and workspace environments. The algorithms impose very few constraints on the dynamics, and extensions of such algorithms have also been used for solving planning and verification problems involving hybrid systems and reachability-based goals [3]–[9].

The focus of this paper is motion planning problems, involving, mobile robots with nonlinear hybrid dynamics, and high-level temporal goals. The problem instances involve geometric constraints arising due to the finite geometry of the robot and the presence of obstacles in the workspace. The high-level temporal goals are described using a subset of Linear Temporal Logic (LTL) formulas that are known as syntactically co-safe LTL formulas [10]. Researchers have advocated the use of LTL for describing temporal properties [11]. LTL has been widely used in model checking of discrete systems in software and hardware systems [12]. LTL has also been used recently as a framework to describe complex high-level goals in planning problems [13]–[19].

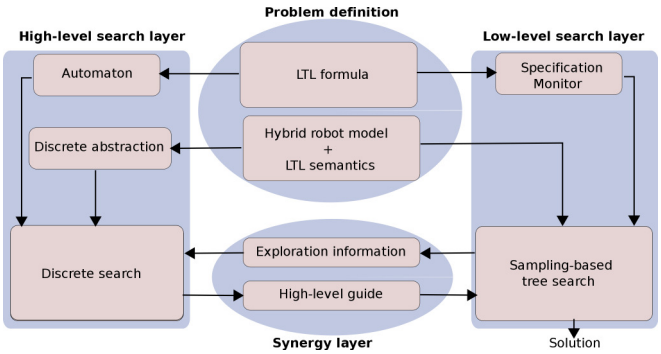



Fig. 1: Multi-layered synergistic approach for planning

The work presented in this paper, as well as [19] use the multi-layered synergistic framework proposed in [16]. Our instantiation of the framework is shown in Figure 1[1] and consists of three main layers: a) The high-level search layer that uses a discrete abstraction for the robot model, the specifications, and the exploration information from the low-level search layer to construct high-level plans, b) The low-level sampling-based search layer that uses the hybrid robot model and the suggested high-level plans to explore the state-space for solution trajectories, and, c) The synergy layer, that facilitates the synergistic interaction between the high-level and the low-level search layers.

The contributions of this paper are twofold. First, we generalize our recently proposed geometry-based approach for the construction of the discrete abstraction [19] to the case when the robot model involves hybrid dynamics. Second, we propose an improved high-level search technique that lets us realize the benefits of using geometry-based discrete abstraction more effectively. As in [19], we follow a more general approach to construct the discrete traces, which includes the class of traces considered in [16] as a subset.

Our proposed approach shows computational speedup of close to 10 times over other possible approaches for challenging benchmark problems involving second-order nonlinear hybrid robot models with finite geometry moving in complex workspace environments with obstacles.

## II. Mathematical Framework

**Robot model** We use the hybrid automata model [20] to describe the hybrid robot dynamics in this paper. A hybrid

The research leading to this work was supported in part by NSF CNS 0615328, NSF EIA-0216467, U.S. ARL W911NF-09-1-0383, and a partnership between Rice University, Sun Microsystems, and Sigma Solutions.

Amit Bhatia, Lydia Kavraki and Moshe Vardi are with the Department of Computer Science, Rice University, Houston, Texas, 77005, {abhatia,kavraki,vardi}@rice.edu

[1]For viewing the figures in this paper, we recommend the online version.

automata is a tuple $H = (S, s_0, \text{INV}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW})$ (we are using notation similar to [16]). $S = Q \times X$ is the product of discrete state-space $Q$ and the set of continuous state-spaces $X = \{X_q \subseteq \mathbb{R}^{n_q} : q \in Q\}$. A pair $s = (q, x) \in S$ denotes a hybrid state of the system and each $q \in Q$ is called a *mode*. $s_0 \in S$ is the initial state. $\text{INV} = \{\text{INV}_q : q \in Q\}$, is the set of invariants, where $\text{INV}_q : X_q \to \{\top, \bot\}$. $E \subseteq Q \times Q$ describes discrete transitions between different discrete modes. $\text{GUARD} = \{\text{GUARD}_{q_i,q_j} : (q_i, q_j) \in E\}$, where $\text{GUARD}_{q_i,q_j} : X_{q_i} \to \{\top, \bot\}$ is the guard function. $\text{JUMP} = \{\text{JUMP}_{q_i,q_j} : (q_i, q_j) \in E\}$, where $\text{JUMP}_{q_i,q_j} : X_{q_i} \to X_{q_j}$ is the jump function. In this paper, the guards and invariants are assumed to be defined as polyhedra (possibly non convex), and the jump functions are assumed to be linear. GUARD is a function of only the discrete transition $e \in E$ and the position of the robot. JUMP keeps the position of the robot the same and resets other components of the state $x \in X$ to 0. $U = \{U_q \subset \mathbb{R}^{m_q} : q \in Q\}$ 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 robot through a set of differential equations. $\text{FLOW}_q(x, u, t)$ gives the continuous state of the robot when the input $u$ is applied for $t$ time units starting from state $x$. A trajectory $\tilde{s}$ of the robot can be described as a concatenation of a finite number of continuous trajectories, interleaved with discrete transitions (cf. [16]). $\Pr(A)$ denotes projection of a Euclidean set $A$ onto $\mathbb{R}^2$.

The first two components of a continuous state $x \in X_q$ denote the position of the robot. $W = \{(q, W_q) : W_q = \Pr(X_q), q \in Q\}$ denotes the workspace for the robot. For a given discrete state $q$, $W_{q,\text{free}} = W_q \setminus W_{q,\text{obs}}$ denotes the set of collision-free positions for the robot and $W_{q,\text{obs}}$ denotes the set of (polygonal) obstacles in the workspace. $W_{\text{free}} = \{(q, W_{q,\text{free}}) : q \in Q\}$ and $W_{\text{obs}} = \{(q, W_{q,\text{obs}}) : q \in Q\}$. $h_H : S \to W$ maps each state $s = (q, x) \in S$ to the the workspace $W$. Given a $w \in W$, we define $h_H^{-1}(w) = \{s \in S : h_H(s) = w\}$. We use the notation $h_H, h_H^{-1}$ for their natural extensions to sets in state-space and workspace as well. For a given discrete state $q$, $G_{q,q'}$ denotes the projection of $\text{GUARD}_{q,q'}^{-1}(\top)$ on $W_q$ and $JG_{q',q}$ denotes the projection of $\text{JUMP}(\text{GUARD}_{q',q}^{-1}(\top))$ on $W_q$.

**Linear temporal logic** Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ be a set of boolean atomic propositions. The semantics are defined over infinite traces of a given system (see Appendix and [12]). Let $\sigma = \{\tau_i\}_{i=0}^{\infty}$, with $\tau_i \in 2^{\Pi}$ and let $\sigma^i = \tau_i, \tau_{i+1}, \ldots$ and $\sigma_i = \tau_0, \tau_1, \ldots, \tau_{i-1}$. $\sigma_i$ is a prefix of the trace $\sigma$. $\sigma \models \phi$ indicates that $\sigma$ satisfies the formula $\phi$. The focus of this paper is motion planning problems over a finite time horizon. Hence, we use syntactically co-safe LTL formulas for describing specifications (see Appendix and [10]).

Given a set of atomic propositions $\Pi$, and a syntactically co-safe LTL formula $\phi$, a Non-deterministic Finite Automaton (NFA) $\mathcal{A}_\phi$ describing all the good prefixes for $\phi$ can be constructed [10]. $A_\phi$ is given by the tuple $\mathcal{A}_\phi = (Z, \Sigma, \delta, z_0, Z_{\text{acc}})$. $Z$ is the set of automaton states, $\Sigma = 2^{\Pi}$ is the input alphabet, $\delta : Z \times \Sigma \to 2^Z$ is the transition relation. $z_0 \in Z$ is the initial state and $Z_{\text{acc}} \subseteq Z$ is the set of accepting final states for the automaton. The set of states on which $\mathcal{A}_\phi$ ends when run on a trace $\sigma = \tau_0, \tau_1, \ldots, \tau_k$, is given by:

$$\mathcal{A}_\phi(\sigma) = \begin{cases} \delta(z_0, \tau_0), & \text{if } k = 0, \\ \cup_{z \in \mathcal{A}_\phi(\sigma_k)} \delta(z, \tau_k), & \text{if } k > 0. \end{cases}$$

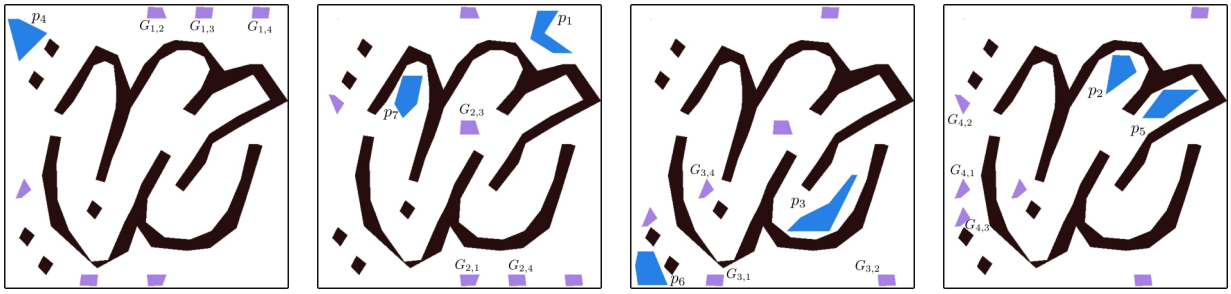
It has been shown recently that using a minimized Deterministic Finite Automaton (DFA) for an NFA can offer significant computational speedups for model checking and falsification of temporal specifications for hybrid systems (cf. [16], [21]). Hence, we use minimized DFA in our work.

**Specifications and simulation issues** Let $\Pi = \{p_0, p_1, p_2, \ldots, p_N\}$ denote the set of boolean atomic propositions. $\Gamma : W_{\text{free}} \to 2^{\Pi}$ maps each point $w \in W_{\text{free}}$ to the set of propositions that hold there and is referred to as the *observation map*. $p_0$ is a proposition that holds true for all $w \in W_{\text{free}} \setminus \bigcup_{i=1,\ldots,N} \Gamma^{-1}(p_i)$. For a given atomic proposition $p \in \Pi$, $\neg p$ holds true for all points $w \in W_{\text{free}} \setminus \Gamma^{-1}(p)$. To interpret an LTL formula over a hybrid-system trajectory, we need an appropriate notion of traces generated by such trajectories. Numerical techniques for incrementally simulating dynamics of a continuous-time system essentially discretize the continuous dynamics (unless arbitrary-precision arithmetic is used, which can be prohibitively expensive) such that the resulting discretization can be implemented efficiently [2]. A practical and universally adopted approach to numerically simulate the dynamics is to consider a discrete-time approximation of the continuous dynamics with respect to a time-step $\Delta t$ and use constant inputs over the time-step $\Delta t$ [2], [4]–[7], [9], [16], [22]. The constraints of invariants and guards are enforced only at sampling instants $c\Delta t, c \in \mathbb{N}$. We use a similar discrete-time approximation, denoted as $H_{\Delta t}$, in our work. The discretization can result in spurious trajectories, but this is unavoidable if numerical techniques are used. In fact, undecidability of the reachability problem [20] shows that this is unavoidable. A practical approach to address this issue is to use additional knowledge of the system for selecting $\Delta t$, and to test the specifications for a set of discrete-time approximations, each corresponding to a different $\Delta t$.

A trace is defined to be a sequence of propositional assignments, where every element of the sequence corresponds to a state of $H_{\Delta t}$. In [16], repeated elements of the sequence are dropped, to obtain a sequence where every two neighboring elements are distinct, whereas in our work we do not impose this requirement. Given a trajectory $\hat{s}$ for $H_{\Delta t}$, starting from $s_0$, under the effect of control sequence $u_1, u_2 \ldots u_k$, the sequence of states $\hat{s} = \hat{s}(0), \hat{s}(1), \hat{s}(2), \ldots \hat{s}(k)$ induces a trace $\sigma(\hat{s}) = \tau_0, \tau_1, \ldots, \tau_k$, $\tau_i = \Gamma(h_H(\hat{s}(i)))$. The LTL semantics is defined with respect to the finite traces generated by $H_{\Delta t}$. These traces are finite, but, under the co-safety restriction, the satisfaction of an LTL formula can be interpreted over finite traces as follows. Given a trajectory $\hat{s}$ of $H_{\Delta t}$, and a co-safe LTL formula $\phi$, we say that $\hat{s}$ satisfies the formula, denoted as $\hat{s} \models_{H_{\Delta t}} \phi$, if $\mathcal{A}_\phi(\sigma(\hat{s}))$ contains an accepting state.

**Motion planning problem** Given a robot model $H$, a sampling interval $\Delta t$, and a syntactically co-safe LTL formula $\phi$, construct a trajectory $\hat{s}$ for the model $H_{\Delta t}$, such that $\hat{s} \models_{H_{\Delta t}} \phi$.

**A hybrid robotic system benchmark** Throughout this paper, we use the following hybrid robotic system as our




(a) Mode 1 (b) Mode 2 (c) Mode 3 (d) Mode 4

Fig. 2: A workspace with 4 modes, 12 guards and 7 propositions. Sets corresponding to discrete transitions are marked in purple, obstacles in brown and propositions in blue color. We refer to Section II for the notation.

test benchmark (shown in Figure 2). The model has four discrete modes and three guard sets in each discrete mode. For each mode, the robot is modeled as either a second-order car, a unicycle or a differential drive with finite geometry. These models are rich enough to capture the key aspects of the dynamics and have been extensively used for benchmarking motion-planning and safety-falsification algorithms (see Appendix and [9], [16], [23]). Furthermore, to make the problem instances even more challenging, we introduce obstacles in the workspace of the robot and model the robot with finite geometry. Such geometric constraints were not present in problem instances considered in [16].

## III. Synergistic motion planning

We now discuss our instantiation of the multi-layered synergistic framework (Figure 1), while addressing the issue of the construction of the discrete abstraction, and suggesting improvements to the high-level search technique. 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 layer.

**High-level search layer** This is the layer that handles most of the discrete nature of the problem. This layer comprises of three key components: a discrete abstraction $D$ of the hybrid robot model $H$, an automaton $\mathcal{A}_\phi$ corresponding to the specification $\phi$, and the high-level search technique.

*a) Discrete abstraction:* One of the main reasons for using a discrete abstract model for solving the problem is to make a quick guess of a possible solution using the discrete abstraction and then explore its feasibility for the hybrid model in the low-level search layer. In [16], the discrete abstraction was treated as a user-supplied input. The benchmark problems considered did not involve geometric constraints, and the problem was set up such that the abstraction was easy to construct. In this paper, we propose an approach to construct the discrete abstraction that uses the geometry of the specifications[2] and the discrete structure of the hybrid dynamics[3]. We utilize the geometry and the discrete structure in a manner such that the overall performance of the multi-layered synergistic framework improves significantly. Note that in our previous work [19], we have explored the issue of the construction of the discrete abstraction for the case when the robot dynamics are continuous. The abstractions used in [17], [24] need to satisfy the bisimilarity property [25], while in the multi-layered synergistic framework (shown in Figure 1), this is not required.

In our work, we use a discrete abstraction that is induced by a decomposition of the workspace of the robot and the discrete structure of hybrid dynamics. A decomposition $D = \cup_{i=0}^{N_D} D_i$ is a partition of workspace $W$ into number of equivalence classes defined by the map $\Upsilon_D : W \to D$. Every equivalence class corresponds to a single discrete mode. For every $d \in D$, let $\Upsilon_D^{-1}(d) = (q \times r)$ (called as the *concretization* of $d$), where $r \subseteq W_q$ and $\Upsilon_D(q \times r) = d$.

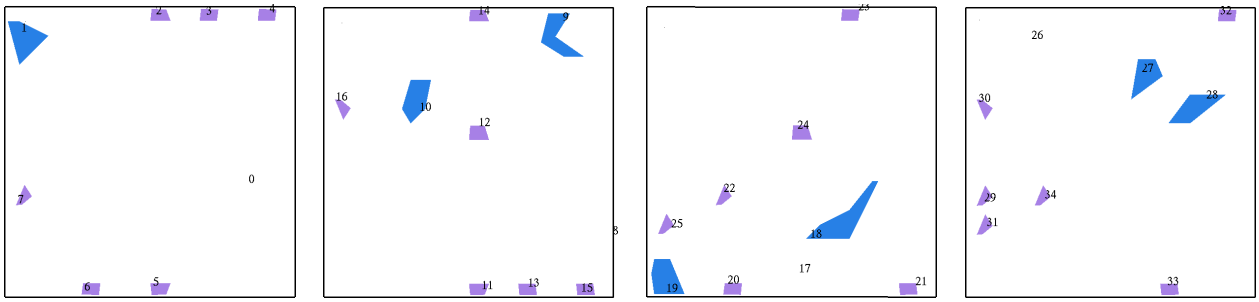
Given the functions GUARD, JUMP and the observation map $\Gamma$, there is more than one way to decompose the workspace $W$. A minimalistic approach towards computing such a decomposition is as follows. We refer to such a decomposition as *(hs) geometry ignoring*[4]. In this decomposition, the set of states corresponding to the sets $G_{q,q'}$ and $JG_{q',q}$ are represented as a single abstract state in the abstraction. Similarly, every nonempty set of states $h_H^{-1}(\Gamma^{-1}(p_i)) \cap \textsc{Inv}_q^{-1}(\top)$ corresponding to a proposition $p_i$, and discrete mode $q$, is represented as a single state in the abstraction. For the robotic benchmark described in Section II, the (hs) geometry-ignoring decomposition is shown in Figure 3 and has 35 elements.

Even though the (hs) geometry-ignoring decomposition is simple to construct, we show through experiments that the performance of the overall approach improves significantly if geometry of the specifications is also used. Such a decomposition is called as *(hs) geometry-using* decomposition. To decompose the workspace using geometry, we triangulate the workspace in each discrete mode. Each $W_q, q \in Q$, is given as a Planar Straight Line Graph (PSLG) to a mesh generation package (we use the Triangle package [26]). A PSLG is made of vertices and segments. Segments are edges whose endpoints are vertices in the PSLG, and whose presence in any mesh generated from the PSLG is enforced. Holes correspond to the regions that cannot be triangulated. The sets describing propositions are given as *segments*. We do not triangulate the sets $G_{q,q'}$ or the sets $JG_{q',q}$ (defined in Section II), but rather specify them as *holes*. Each of these sets is then added later to the decomposition. $W_{\text{q,obs}}$ is given as a set of *holes*. One such decomposition for the example

[2]The geometry of specifications is described by the sets describing the invariants, guards, obstacles, and the propositions.

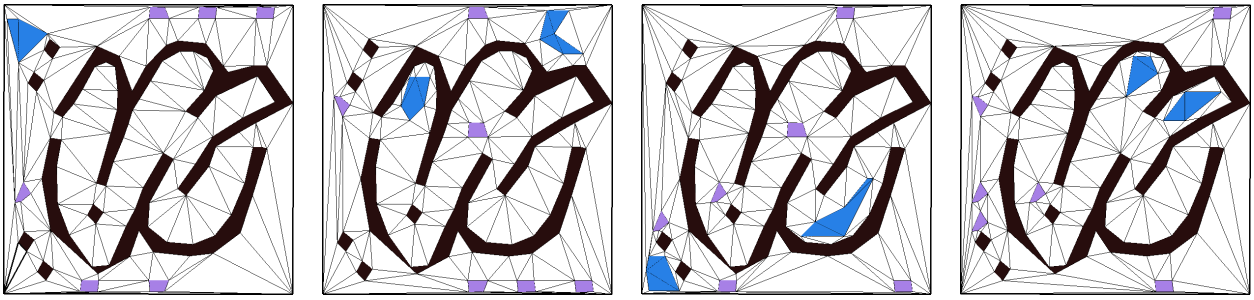
[3]The discrete structure of hybrid dynamics is described by the set of possible discrete transitions (refer to Section II) between different modes.

[4](HS) is used to indicate that the discrete structure of hybrid dynamics is also being used.



(a) Mode 1 (b) Mode 2 (c) Mode 3 (d) Mode 4

Fig. 3: (HS) Geometry-ignoring decomposition of workspace shown in Figure 2. The decomposition has 35 elements.



(a) Mode 1 (b) Mode 2 (c) Mode 3 (d) Mode 4

Fig. 4: (HS) Geometry-using decomposition of workspace shown in Figure 2. The decomposition has 706 elements.

considered in Figure 2 is shown in Figure 4 and has 706 elements. Note that the geoemtry-based approach proposed here is one of several alternatives to construct the discrete abstraction. For example, it is also possible to use grid-based decompositions as in [23].

Given a hybrid system $H$, $M = (D, d_0, \rightarrow_D, h_D)$ denotes a discrete abstraction of the system[5]. Here $D$ is a set of states, $d_0 \in D$ is the initial state of the abstraction, and $\rightarrow_D \subseteq D \times D$ is the transition relation. Each element $d \in D$ of the decomposition is used as a state for the abstraction. We add a transition between $d_i, d_j$ if there is a discrete transition between their concretizations, or if their concretizations share an edge in the low-dimensional triangulation. $h_D : D \rightarrow \Pi$ is the observation map for the abstraction that identifies the set of propositions from $\Pi$ that is true in a given abstract state $d \in D$ and is defined as $h_D(d) = \Gamma(\Upsilon_D^{-1}(d))$[6].

We wish to remark here that the idea of using a triangulation-based decomposition has been used before for continuous robot models (cf. [17], [24]). However, the abstractions used in [17], [24] need to satisfy the bisimilarity property [25], while in our work, this is not required.

*b) Discrete search:* To construct high-level plans, the product of $\mathcal{A}_\phi$ and $M$ is checked for feasible runs. Existence of a feasible run on $\mathcal{A}_\phi \times M$ implies that the abstraction $M$ can satisfy the specification $\phi$ [27]. The idea of using the product $\mathcal{A}_\phi.Z \times M.D$ for discrete planning has also been used before [14], [17], [28]. An important difference in our approach is that the vertices and edges in the graph representation of $\mathcal{A}_\phi.Z \times M.D$ are assigned weights. These are used to synergistically convey the low-level exploration information to the high-level layer.

A pair $(z, d) \in \mathcal{A}_\phi.Z \times M.D$ is called a high-level state. A high-level plan $\zeta$ is a sequence of high-level states $(z_i, d_i)_{i=1}^k$ such that $d_i \rightarrow_D d_{i+1}, \forall i \in [1, k-1]$, and $z_i \in \delta(z_{i-1}, h_D(d_i))$ and $z_k$ satisfies the acceptance condition of the automaton $\mathcal{A}_\phi$. Every high-level state $(z, d)$ is assigned a weight $\rho(z, d)$ (explained as part of the synergy layer). An edge $e((z_i, d_i), (z_j, d_j))$ connecting high-level states $(z_i, d_i)$ and $(z_j, d_j)$ is assigned a weight $(\rho(z_i, d_i) * \rho(z_j, d_j))^{-1}$. The high-level layer constructs a high-level plan as the shortest path from a given abstract state to the set of (abstract) accepting states using Dijkstra's algorithm. To account for the fact that the weights are an estimate of feasibility, the planner also computes a random path occasionally, which need not be the shortest one.

The high-level search technique proposed in [16] (referred to as *reinitialized-search* technique) always starts the search from $(z_0, d_0)$. This can be expensive for the cases when the size of search space $(\mathcal{A}_\phi.Z \times M.D)$ is big. In this paper, we use a *lazy-search* technique that starts the search from $(z_0, d_0)$ only when other candidate high-level states that have been used previously do not look promising. Instead of reinitializing the search from $(z_0, d_0)$ every time a new high-level plan is being constructed, the lazy-search initializes search from previously explored high-level states. This effectively means that portions of previously explored plans are reused. The selection of a high-level state $(z, d)$ from the high-level states of a previously explored high-level plan $\zeta$ is made using the weight function $\rho$. Further details are discussed as part of the algorithm description.

**Low-level search layer** A high-level plan $\zeta$ is checked for feasibility incrementally at the low-level search layer, by exploring the hybrid state-space $S = Q \times X$. The suggested high-level plan $\zeta$ is used to bias the search such that the

[5]The abstraction is called as (hs) geometry ignoring or (hs) geometry using, depending on the kind of decomposition used.

[6]With slight abuse of notation, we define $\Gamma(\Upsilon_D^{-1}(d)) = \Gamma(w)$, where $d = \Upsilon(w), w \in W$.

resulting exploration of $S$ improves the chances of finding a feasible trajectory satisfying the specification $\phi$. The search is done for a predetermined exploration time $t_{explore}$, using highly successful sampling-based algorithms [2] that build an exploration tree $\mathcal{T}$ in $S$ while keeping estimates for the coverage of $S$. Note that we use the sampling-based algorithm used in [16], but others [4]–[6], [8], [22] could also be used. The low-level search layer passes the exploration information to the synergy layer. The low-level layer also uses the automaton $\mathcal{A}_\phi$ as a specification monitor to identify when a trajectory satisfying $\phi$ has been found. The low-level layer differs from [16] in the fact that we are using a different approach to generate traces (see Sections I, II), and we also need to take into account the finite geometry of the robot, and the presence of obstacles in the workspace, while constructing feasible trajectories for $H_{\Delta t}$.

**Synergy layer** A high-level plan is based on the formula $\phi$ and the abstraction $M$. A synergistic interaction between the high-level and the low-level layers is facilitated by using a feasibility estimate associated with each high-level state [16]. The feasibility estimates help convey the information about hybrid dynamics and the low-level exploration systematically from the low-level layer to the high-level layer. Given a high-level state $(z, d)$, the feasibility estimate associated with it is given by weight $\rho(z, d)$. The formula for $\rho(z, d)$ has been derived experimentally and is similar to one used in [16]. $\rho(z, d)$ is defined as $\rho(z,d) = \frac{(\mathrm{cov}\,(z,d)+1)*\mathrm{vol}\,(h_H(\Upsilon_D^{-1}(d)))}{\rho_{\mathcal{A}}^3(z)*(\mathrm{nrsel}\,(z,d)+1)^2}$, where $\mathrm{cov}\,(z, d)$ estimates the coverage of the set $h_H(\Upsilon_D^{-1}(d))$ from the tree vertices associated with $(z, d)$, $\mathrm{vol}\,(h_H(\Upsilon_D^{-1}(d)))$ is the volume of the projection of the set $r$, with $(q \times r) = \Upsilon_D^{-1}(d)$ and $\mathrm{nrsel}\,(z, d)$ is the number of times the high level state $(z, d)$ has been selected in the past for further exploration. $\rho_{\mathcal{A}}(z)$ computes the shortest distance of the state $z$ from the set of accepting states $Z_{\mathrm{acc}}$ (in terms of number of transitions required) for the automaton $\mathcal{A}_\phi$.

If a solution is found, the search stops and the solution is returned, else the low-level layer continues the exploration with a new high-level plan suggested by the high-level layer, based on updates to feasibility estimates from last iteration. The overall search is conducted for a predetermined exploration time $t_{max}$.

**Algorithm** The main algorithm (called as `ML-LTL-H`) is shown in Figure 5 and the low-level sampling-based search algorithm (called as `EXPLORE-H`) is shown in Figure 6. The `ML-LTL-H` algorithm has two key differences from the one used in [16]. First, the discrete abstraction is not treated as a user-defined input, but instead is constructed as part of the algorithm (*Line:4,5*, Figure 5). Second, the high-level search uses the lazy-search technique (*Line:12-17*, Figure 5). We next describe the input, the output and the data-structures used in the `ML-LTL-H` algorithm.

*Input:* The algorithm takes as an input the hybrid model $H$, a syntactically co-safe LTL formula $\phi$, a time-step $\Delta t$, the bound on simulation time $t_{max}$ and the bound on exploration time $t_{explore}$ for each call to the low-level search algorithm.

*Output:* The algorithm returns a boolean variable $soln$ that is true if a solution is found. If a solution is found, then $\hat{s}$ is a trajectory of $H_{\Delta t}$ whose trace satisfies $\phi$.

*Data structures:* The search tree $\mathcal{T}$ is stored as a directed

```
ML-LTL-H(H,φ,Δt, t_explore, t_max)
1  A_φ ← COMPUTE_AUTOMATON(φ) {Compute automaton}
2  H_Δt ← COMPUTE_DISCRETIZATION(H, Δt) {Compute discrete-time approximation}
3  T ← INIT(H_Δt.s_0, A_φ) {Initialize search tree}
4  D ← DECOMPOSE(H_Δt, Υ_D) {Compute decomposition}
5  M = G(V, E) ← COMPUTE_ABSTRACTION(D) {Construct abstraction}
6  (ρ, ϵ) ← INITIALIZE_ESTIMATES(A_φ, M) {Initialize feasibility estimates}
7  (soln, s̄) ← (⊥, ∅)
8  (z_prev, d_prev) ← (∅, ∅)
9  σ_avail ← INITIALIZE_AVAIL_HIGHLEVEL_STATES(A_φ, M)
10 clck ← 0 { Initialize timer}
11 while (clck ≤ t_max ∧ ¬soln) do {Search for solution}
12     (z_init, d_init) ← SELECT_HIGHLEVEL_STATE(ρ, σ_avial) {Select a high-level state}
13     if ((z_init, d_init) = (z_prev, d_prev) ∨ ρ(z_init, d_init) < ϵ) then
14         (z_init, d_init) ← (A_φ.z_0, d_0)
15         (z_prev, d_prev) ← (∅, ∅)
16     else
17         (z_prev, d_prev) ← (z_init, d_init)
18     ζ ← DISCRETE_SEARCH(M, A_φ, ρ, (z_init, d_init)) {Construct high-level plan}
19     (T, ρ, σ_avail, soln, v_acc) ← EXPLORE-H(A_φ,T,M,ρ,ζ,t_explore,H_Δt) {Low-level search}
20     if (soln) then
21         (s̄) ← CONSTRUCT_TRAJECTORY(T, v_acc) {Construct solution trajectory}
22 return (T, soln, s̄)
```

Fig. 5: Multi-layered synergistic planning algorithm

graph $\mathcal{T} = G_{\mathcal{T}}(V_{\mathcal{T}}, E_{\mathcal{T}})$. Each vertex $v' \in V_{\mathcal{T}}$ of the tree $\mathcal{T}$ stores a feasible state $v'.s$ of the hybrid model $H_{\Delta t}$ and an edge $e(v, v') \in E_{\mathcal{T}}$ connecting it to its parent vertex $v$. An edge $e(v, v') \in E_{\mathcal{T}}$ stores an input $u$, and time duration $t = \Delta t$ such that $v'.s$ is the resulting state of $H_{\Delta t}$. $v$ is called the *parent* of the vertex $v'$ and $v'$ a *child* of the vertex $v$. $D$ is the computed decomposition and $M$ is the abstraction. $\zeta$ denotes a high-level plan. $(z_{init}, d_{init})$ and $(z_{prev}, d_{prev})$ store respectively the high-level state used as initial state in current iteration and in the last iteration for constructing the high-level plan. For a given high-level state $(z, d)$, let $(z, d).vertices = \{v \in V_{\mathcal{T}} : z \in v.\alpha, v.s \in \Upsilon_D^{-1}(d)\}$. For a given high-level plan $\zeta$, $\sigma_{avail}$ stores a subset of the high-level states $(z_i, d_i)$, such that $(z_i, d_i).vertices \neq \emptyset$. The high-level states with a higher index $i$ are favored more for addition to $\sigma_{avail}$. With slight abuse of notation, we use $\rho$ (previously used to denote the feasibility estimate function) to also denote the data structure holding the feasibility estimates for each of the high-level states. $\mathcal{T}(v_{\mathrm{init}}, v)$ denotes the sequence of vertices connecting the root $v_{\mathrm{init}}$ of the tree to the vertex $v$. Let $\hat{s}(\mathcal{T}(v_{\mathrm{init}}, v))$ denote the corresponding sequence of states. $v.\alpha = \mathcal{A}_\phi(\sigma)$, stores the state of the automaton $\mathcal{A}_\phi$ when run on the trace $\sigma = \Gamma(\hat{s}(\mathcal{T}(v_{\mathrm{init}}, v)))$. $v_{\mathrm{acc}}$ is a tree vertex such that $v_{\mathrm{acc}}.\alpha \cap \mathcal{A}_\phi.Z_{\mathrm{acc}} \neq \emptyset$.

The lazy-search technique is implemented in *Line:12-17*, of the `ML-LTL-H` algorithm (Figure 5). SELECT_HIGHLEVEL_STATE$(\rho, \sigma_{avial})$ selects a high-level state $(z_{init}, d_{init})$ from the set of available high-level states $\sigma_{avail}$ as the starting high-level state for a new high-level plan. A state $(z, d)$ is selected with probability $\frac{\rho(z,d)}{\Sigma_{(z,d)\in\sigma_{avail}}\rho(z,d)}$. If the selected high-level state $(z_{init}, d_{init})$ is the same as the one last selected during previous iteration, or if the feasibility estimate of $(z_{init}, d_{init}) < \epsilon$, then the high-level planning is reinitialized to start from the high-level state $(z_0, d_0)$. If not, then $z_{prev}, d_{prev}$ is updated to $(z_{init}, d_{init})$. Note that the reinitialized search technique of [16] is equivalent to setting $\epsilon = \infty$ on *Line 6* (Figure 5).

The low-level sampling-based search algorithm `EXPLORE-H` is shown in Figure 6. The algorithm is different from the exploration algorithm proposed in [16] in two major ways. First, we are using a different approach to generate discrete traces compared to [16] (*Lines:11,15*,

Figure 6; see Sections I, II). Second, our problem instances involve finite geometry of the robot and the presence of obstacles in workspace. Hence, to guarantee feasibility of system states, we also need to include collision detection in the low-level search (*Line:11*, Figure 6). We next describe the input, the output and the additional data-structures used in the EXPLORE−H algorithm.

EXPLORE-H($\mathcal{A}_\phi$,$\mathcal{T}$,$M$,$\rho$,$\zeta$,$t_{explore}$,$H_{\Delta t}$)

1 $(soln, v_{\rm acc}, clck) \leftarrow (\bot, \emptyset, 0)$ {Initialize search}
2 $\sigma_{avail} \leftarrow$SELECT_FEASIBLE_HIGHLEVEL_STATES($\zeta$, $\mathcal{T}$) {Select a few feasible high-level states}
3 **while** $(clck < t_{explore} \wedge \neg soln)$ **do**
4 $(z_{sel}, d_{sel}) \leftarrow$ SELECT_HIGHLEVEL_STATE($\rho$, $\sigma_{avail}$) {Select high-level state}
5 $v_{sel} \leftarrow$ SELECT_VERTEX ($\rho$, $(z_{sel}, d_{sel})$) {Select tree vertex for expansion}
6 $s_1 \leftarrow v_{sel}.s$
7 $nsteps \leftarrow$ CHOOSE_STEPS($H_{\Delta t}$) {Set number of iterations}
8 **for** $i = 1, \ldots, nsteps$ **do**
9 $u \leftarrow$ CHOOSE_INPUT ($H_{\Delta t}.U$, $\mathcal{T}$, $v_{sel}$) {Select an input}
10 $(s_{i+1}, v_{i+1}) \leftarrow$SIMULATE_DYNAMICS ($H_{\Delta t}$, $s_i$, $u$) {Simulate discrete-time dynamics}
11 **if** CHECK_FEASIBILITY($H_{\Delta t}$, $\mathcal{A}_\phi$, $v_{i+1}$) **then**
12 $\mathcal{T} \leftarrow$UPDATE_TREE($\mathcal{T}$, $v_{i+1}$, $u$, $\Delta t$) {Tree update}
13 $\sigma_{avail} \leftarrow$UPDATE_FEASIBLE_HIGHLEVEL_STATES($\zeta$, $v_{i+1}$) {Update available states}
14 $\rho \leftarrow$ UPDATE_FEASBILITY_ESTIMATES($\rho$, $v_{i+1}$) {Update feasibility estimates}
15 **if** $(v_{i+1}.\alpha \cap \mathcal{A}_\phi.Z_{\rm acc} \neq \emptyset)$ **then** {Check for acceptance condition}
16 $(soln, v_{\rm acc}) \leftarrow (\top, v_{i+1})$
17 break
18 **else**
19 break
20 return $(\mathcal{T}, \rho, soln, v_{\rm acc}, \sigma_{avail})$

Fig. 6: Low-level sampling-based search algorithm

*Input:* The algorithm takes as an input the automaton $\mathcal{A}_\phi$, search tree $\mathcal{T}$, abstraction $M$, feasibility estimates $\rho$, suggested high-level plan $\zeta$, bound on exploration time $t_{explore}$ and sampling interval $\Delta t$.

*Output:* The algorithm returns the boolean variable $soln$, the updated feasibility estimates $\rho$, updated search tree $\mathcal{T}$. Additionally if $solution = \top$, then $v_{\rm acc}$ is the vertex marked with an accepting state of the automaton.

*Data structures:* The following additional data structures are used by the low-level algorithm. For a given vertex $v$, $v.nsel$ stores the number of times the vertex has been selected for expansion before. Similarly for a given high-level state $(z, d)$, $(z, d).nsel$ stores the number of times the state has been selected before for exploration.

## IV. Experimental results

We now discuss the experimental results obtained using the proposed approach. We also do a comparative analysis with some of the other possible approaches to highlight the computational speedups obtained using ideas proposed in this paper. The focus of comparison is two-fold. First, to evaluate the advantage of using the geometry of invariants, guards and propositions while constructing the abstraction. Second, to evaluate the advantage of doing a lazy search for high-level plans, which saves the computational effort spent in high-level search layer by starting the search adaptively from a previously explored high-level state that need not be $(z_0, d_0)$.

**Implementation and hardware** The code developed for simulations presented in this paper is based on the OOPSMP library [29] and builds on top of the code developed for [16], [19]. For computing DFAs for syntactically co-safe LTL formulas, we have used the tool scheck [30]. For computing triangulations, we have used the package Triangle [26]. All the simulations were run on Rice SUG@R cluster. Each processor used from the cluster is an Intel Xeon processor running at 2.83 GHz, and can access up to 16 GB RAM. Each simulation run was run on a single processor with no parallelism. The time allocated for each simulation $t_{max}$ was set to 1200 seconds and the exploration time $t_{explore}$ (for single call to EXPLORE−H) was set to 0.75 seconds. The time-step $\Delta t$ was set to 0.25 seconds.

*LTL formulas* We have considered the following LTL formulas for the experiments. $\phi_1^{n_{\rm op}} = \wedge_{i=1}^{n_{\rm op}}(Fp_i)$, $\phi_2^{n_{\rm op}} = F(p_1 \wedge F(p_2 \wedge F(p_3 \ldots F(p_{n_{\rm op}}))))$, $\phi_3^{n_{\rm op}} = F(p1 \wedge ((p_0 \vee p_1)\mathcal{U}(p_2 \wedge ((p_0 \vee p_2)\mathcal{U} \ldots (p_{n_{\rm op}})))))$, where $n_{\rm op} \in [1, 7]$ is the number of temporal operators in the formula. $\phi_1^{n_{\rm op}}$ are usually referred to as coverage formulas, since they can be used to describe coverage specifications over the workspace of the robot. $\phi_2^{n_{\rm op}}$ are referred to as the sequencing formulas and these are used to describe sequencing requirements in the specifications. $\phi_3^{n_{\rm op}}$ are the strict sequencing formulas. They are different from sequencing formulas in the fact that the order of visits is strict. In Figure 7, the size of minimized DFA for different kinds of specifications is shown.

| $n_{\rm op}$ | # of states / # of transitions | | |
|---|---|---|---|
| | Coverage | Sequencing | Strict sequencing |
| 1 | 2 / 2 | 2 / 2 | 2 / 2 |
| 2 | 4 / 8 | 3 / 5 | 3 / 6 |
| 3 | 8 / 26 | 4 / 9 | 4 / 12 |
| 4 | 16 / 80 | 5 / 14 | 6 / 28 |
| 5 | 32 / 242 | 6 / 20 | 10 / 76 |
| 6 | 64 / 728 | 7 / 27 | 17 / 209 |
| 7 | 128 / 2186 | 8 / 35 | 29 / 569 |

Fig. 7: Size of minimized DFA for different specifications

*Test cases:* We have used constrained Delaunay triangulation for computing the geometry-based decomposition. To avoid generating problem instances where the robot is in collision with obstacles in initial state, the initial state of the robot was fixed at $s_0 = (0, (-0.43, 0.45, 0, 0, 0)^T)$, which is a collision-free state for every robot model. The dynamics in each discrete mode (including the bounds on input) were chosen using uniform distributions. All the computation times are reported as mean over 40 test runs. To account for the number of timeouts in finding a solution, we have used the value of timeout (1200 seconds) for the test runs when no solution was found. For every experiment involving 40 test runs, less than $10\%$ of the runs reported a timeout.

**Comparative analysis** We now discuss the benefits of using a (hs) geometry-using abstraction and lazy-search technique in high-level search layer by examining what happens if one or both of these key ideas are not used. We refer to Section III for the discussion on different approaches.

*Reinitialized search, (hs) geometry ignoring* (Figure 8a) We start by evaluating the performance obtained using (hs) geometry-ignoring decomposition for the abstraction, and reinitialized search technique for the high-level search.

*Reinitialized search, (hs) geometry using* (Figure 8b) We now consider the improvements obtained by using a (hs) geometry-using decomposition, but with reinitialized search for high-level search. The results shown in Figure 8b indicate an improvement of up to 3-5 times with the coverage formulas taking the longest. This indicates that an efficient high-level search strategy is also a key parameter affecting the overall performance of the approach.

*Lazy search, (hs) geometry ignoring* (Figure 8c) We now consider the improvements obtained by using only the lazy-search technique in high-level search layer, but with

| $n_{\text{op}}$ | Mean computation times (seconds) | | |
|---|---|---|---|
| | Coverage | Sequencing | Strict sequencing |
| 1 | 0.4 | 0.4 | 0.4 |
| 2 | 28.7 | 17.8 | 33.1 |
| 3 | 80.6 | 80.3 | 97.9 |
| 4 | 135.2 | 92.3 | 131.5 |
| 5 | 219.9 | 110.4 | 161.1 |
| 6 | 489.0 | 173.2 | 319.3 |
| 7 | 644.9 | 186.7 | 358.1 |

(a) Reinitialized search, (hs) geometry ignoring

| $n_{\text{op}}$ | Mean computation times (seconds) | | |
|---|---|---|---|
| | Coverage | Sequencing | Strict sequencing |
| 1 | 0.6 | 0.6 | 0.6 |
| 2 | 10.9 | 6.3 | 7.5 |
| 3 | 42.1 | 31.1 | 25.7 |
| 4 | 74.1 | 29.0 | 39.9 |
| 5 | 115.4 | 35.7 | 49.4 |
| 6 | 265.9 | 53.8 | 69.8 |
| 7 | 519.5 | 62.5 | 117.0 |

(b) Reinitialized search, (hs) geometry using

| $n_{\text{op}}$ | Mean computation times (seconds) | | |
|---|---|---|---|
| | Coverage | Sequencing | Strict sequencing |
| 1 | 0.5 | 0.5 | 0.5 |
| 2 | 23.0 | 14.7 | 28.8 |
| 3 | 58.8 | 112.7 | 101.2 |
| 4 | 92.6 | 138.6 | 141.4 |
| 5 | 105.9 | 147.2 | 223.1 |
| 6 | 246.9 | 212.0 | 331.1 |
| 7 | 237.1 | 247.2 | 445.2 |

(c) Lazy search, (hs) geometry ignoring

| $n_{\text{op}}$ | Mean computation times (seconds) | | |
|---|---|---|---|
| | Coverage | Sequencing | Strict sequencing |
| 1 | 0.5 | 0.6 | 0.6 |
| 2 | 6.4 | 5.4 | 6.3 |
| 3 | 24.0 | 25.9 | 28.5 |
| 4 | 25.1 | 24.9 | 31.7 |
| 5 | 32.5 | 28.0 | 40.1 |
| 6 | 61.8 | 44.6 | 69.6 |
| 7 | 77.4 | 53.7 | 81.8 |

(d) Lazy search, (hs) geometry using

Fig. 8: Performance comparison of different possible approaches

(hs) geometry-ignoring decomposition of the workspace for constructing the abstraction. The results indicate that for coverage formulas, the performance improves significantly. However, for sequencing and strict-sequencing formulas, the performance degrades. This indicates that the lazy search is most useful only when used with meaningful abstractions.

*Lazy search, (hs) geometry using* (Figure 8d) Finally, we show the mean computation times when using the approach proposed in the paper in Figure 8d. Note that our approach of using (hs) geometry-using abstractions and lazy search for high-level plans consistently outperforms other possible approaches considered here by significant margin. In particular, this approach shows a speedup of 6-8 times when compared with the approach of using (hs) geometry-ignoring abstraction and reinitialized search for coverage formulas involving 5-7 temporal operators.

The results of comparative analysis lead to two important conclusions. First, systematic decomposition of the state-space using the geometry of the specifications, and the discrete structure of hybrid dynamics has a profound effect on computational efficiency of the approach. For example, when using reinitialized search, the (hs) geometry-using abstraction yields a speedup of up to 3-4 times for sequencing and strict sequencing formulas when compared with (hs) geometry-ignoring abstraction (Figures 8a, 8b).

Second, geometry-based decompositions result in abstractions of larger size (Figures 3, 4). Hence, to realize the benefits of such decompositions fully, an efficient high-level search technique is also required. In fact, the results from Figure 8 indicate that the best performance is obtained when geometry-based abstractions are used in combination with the proposed lazy-search technique. For the case of coverage formula with 6-7 temporal operators, combining the (hs) geometry-using abstraction and the lazy-search technique yields a performance improvement of close to 10 times (Figures 8a, 8d).

## V. Conclusions

In this paper, we have considered motion planning problems involving mobile robots with nonlinear hybrid dynamics and finite geometry, obstacles in the workspace, and high-level temporal goals. We have proposed an instantiation of the multi-layered synergistic framework proposed in [16] while addressing two key issues: the construction of the discrete abstraction for the case of hybrid dynamics, and an improved high-level search technique to use the discrete abstraction more efficiently. For the construction of the discrete abstraction, we have proposed a geometry-based approach that extends our previous work [19] to the case when the robot is modeled as a hybrid system. To utilize such abstractions more efficiently, we have also proposed a lazy-search technique that reuses portions of previously explored high-level plans for constructing new plans. We have shown through examples that the abstractions constructed using the geometry of the specifications, and the discrete structure of hybrid dynamics, improve the computational performance of the approach significantly. The best performance is obtained when such abstractions are used together with the lazy-search technique in the high-level search layer. In the future, we plan to generalize the framework to a broader class of hybrid systems, and consider problem instances with larger number of discrete modes.

## VI. Acknowledgments

We would like to thank Y. Lustig and E. Plaku for many useful comments and suggestions.

## References

[1] H. Choset, K. M Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. 2005.

[2] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 1st edition, 2006.

[3] E. Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001.

[4] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *HSCC'04*, volume 2993 of *LNCS*, pages 142–156. Springer, 2004.

[5] J. Kim, J. M. Esposito, and V. Kumar. An RRT-based algorithm for testing and validating multi-robot controllers. In *RSS'05*, 2005.

[6] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. Sampling-based planning, control and verification of hybrid systems. *Control Theory and Applications, IEEE Proceedings* -, 153(5):575–590, 2006.

[7] A. Bhatia and E. Frazzoli. Sampling-based resolution-complete algorithms for safety falsification of linear systems. In *HSCC'08*, volume 4981 of *LNCS*, pages 606–609. Springer, 2008.

[8] T. Dang and T. Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, 2009.

[9] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design*, 34(2):157–182, 2009.

[10] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19:291 – 314, 2001.
[11] A. Pnueli. The temporal logic of programs. In *SFCS '77*, pages 46–57. IEEE Computer Society, 1977.
[12] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. 2000.
[13] S. G. Loizou and K. J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on LTL specifications. In *IEEE Conference on Decision and Control*, volume 1, pages 153–158 Vol.1, 2004.
[14] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.
[15] S. Karaman and E. Frazzoli. Complex mission optimization for multiple-UAVs using linear temporal logic. In *American Control Conference*, pages 2003–2009, 2008.
[16] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Falsification of LTL safety properties in hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *LNCS*, pages 368–382. Springer-Verlag, 2009.
[17] G. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45:343–352, 2009.
[18] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning for dynamical systems. In *IEEE Conference on Decision and Control*, 2009.
[19] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation*, pages 2689–2696, 2010.
[20] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
[21] R. Armoni, S. Egorov, R. Fraer, D. Korchemny, and M. Y. Vardi. Efficient LTL compilation for SAT-based model checking. In *Proceedings of the IEEE/ACM International Conference on Computer-aided design*, pages 877–884, Washington, DC, USA, 2005. IEEE Computer Society.
[22] T. Dang, A. Donze, O. Maler, and N. Shalev. Sensitive state-space exploration. In *IEEE CDC'08*, pages 4049–4054. IEEE, Dec. 2008.
[23] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Impact of workspace decompositions on discrete search leading continuous exploration (DSLX) motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3751–3756. IEEE, 2008.
[24] C. Belta, V. Isler, and G. J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21, 2005.
[25] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88:971 – 984, 2000.
[26] J. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, volume 1148 of *LNCS*, chapter 23, pages 203 – 222. Springer-Verlag, 1996.
[27] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *the VIII Banff Higher order workshop conference on Logics for concurrency : structure versus automata*, pages 238–266. Springer-Verlag New York, Inc., 1996.
[28] P. Tabuada and G. J. Pappas. Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.
[29] E. Plaku, K. E. Bekris, and L. E. Kavraki. OOPS for motion planning: An online, open-source, programming system. In *IEEE International Conference on Robotics and Automation*, pages 3711–3716, 2007. Available for download at `http://www.kavrakilab.org/OOPSMP`.
[30] T. Latvala. Efficient model checking of safety properties. In *Model Checking Software*, volume 2648 of *LNCS*, pages 74–88. Springer, 2003.

## APPENDIX

*a) Additional details on hybrid robotic benchmark:* We discuss some of the additional details of the robotic hybrid benchmark introduced in Section II. We use a normalized length scale defined according to the relation $1\text{m} = 0.05$ non-dimensional units. The workspace is chosen as a square of size 1 for each discrete mode.

The dynamics for second-order car are as follows. The state is given by $x = (x_1, x_2, \theta, v, \phi)$, with $\dot{x_1} = v\cos(\theta)$, $\dot{x_2} = v\sin(\theta)$, $\dot{\theta} = v\tan(\phi)/L$, $\dot{v} = u_1$, $\dot{\phi} = u_2$. $L$ is the distance between front and rear axles. The translational velocity $v$ is bounded in magnitude by $v_{max} \in [1\text{m/s}, 3\text{m/s}]$ and the steering angle $\phi$ is bounded in magnitude by $\phi \in [30^\circ, 70^\circ]$. The linear acceleration is bounded in magnitude by $u_{1,max} \in [0.2\text{m/s}^2, 1.0\text{m/s}^2]$ and the steering velocity is bounded in magnitude by $u_{2,max} \in [5^\circ/s, 15^\circ/s]$. We have used $L = 1\text{m}$ and the width of the car as $0.5\text{m}$.

The dynamics for second-order unicycle are as follows. The state is given by $x = (x_1, x_2, \theta, v, \omega)$, with $\dot{x_1} = v\cos(\theta)$, $\dot{x_2} = v\sin(\theta)$, $\dot{\theta} = \omega$, $\dot{v} = u_1$, $\dot{\omega} = u_2$. The translations and rotational speeds are bounded by $v_{max} \in [3\text{m/s}, 10\text{m/s}]$ and $\omega_{max} \in [30^\circ/\text{s}, 70^\circ/\text{s}]$ respectively. The translational and rotational accelerations are bounded in magnitude by $u_{1,max} \in [0.2\text{m/s}^2, 1.0\text{m/s}^2]$ and $u_{2,max} \in [5^\circ/\text{s}, 15^\circ/\text{s}]$ respectively. The geometric length and width of the model are fixed as $1, 0.5\text{m}$ respectively.

The dynamics for second-order differential drive are as follows. The state is given by $x = (x_1, x_2, \theta, \omega_l, \omega_r)$, with $\dot{x_1} = 0.5r(\omega_l + \omega_r)\cos(\theta)$, $\dot{x_2} = 0.5r(\omega_l + \omega_r)\sin(\theta)$, $\dot{\theta} = r/L(\omega_r - \omega_l)$, $\dot{\omega_l} = u_1$, $\dot{\omega_r} = u_2$. The rotational velocities $w_l, w_r$ are bounded in magnitude by $w_{l,max}, w_{r,max} \in [70^\circ/\text{s}, 90^\circ/\text{s}]$. The wheel radius $r$ is set to $0.25\text{m}$ and the distance $L$ between front and rear axles is set to $1\text{m}$. The rotational accelerations for left and right wheel $u_1, u_2$ are bounded in magnitude by $u_{1,max}, u_{2,max} \in [15^\circ/\text{s}^2, 30^\circ/\text{s}^2]$.

*b) Linear Temporal Logic:* Linear temporal logic is a propositional logic that is used to describe modalities of time along trajectories of a given system. Given a set of boolean atomic propositions, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$, the syntax is defined according to the following grammar: $\phi := \pi | \neg\phi | \phi \vee \phi | \mathcal{X}\phi | \phi\mathcal{U}\phi$. $\mathcal{X}$ and $\mathcal{U}$ are temporal operators and $\neg, \vee$ are boolean operators. Additionally, using the operators specified above, it is possible to define $\mathcal{R}$ (release), $\mathcal{F}$ (future) and $\mathcal{G}$ (globally, always) temporal operators as well. The semantics of LTL are defined over infinite traces of a given system. Let $\sigma = \{\tau_i\}_{i=0}^{\infty}$, with $\tau_i \in 2^\Pi$ and let $\sigma^i = \tau_i, \tau_{i+1}, \ldots$ and $\sigma_i = \tau_0, \tau_1, \ldots, \tau_{i-1}$. $\sigma_i$ is a prefix of the trace $\sigma$. For more details, we refer to [12]. $\sigma \models \phi$ indicates that $\sigma$ satisfies the formula $\phi$. $\sigma \models \top$, $\sigma \not\models \bot$, $\sigma \models \pi$ if $\pi \in \tau_0$, $\sigma \models \phi \vee \psi$ if $\sigma \models \phi$ or $\sigma \models \psi$. $\sigma \models \neg\phi$ if $\sigma \not\models \phi$. $\sigma \models \mathcal{X}\phi$ if $\sigma^1 \models \phi$; $\sigma \models \phi\mathcal{U}\psi$, if $\exists k \geq 0$, s.t., $\sigma^k \models \psi$, and $\forall i \in [0, k)$, $\sigma^i \models \phi$. $\mathcal{F}\phi = \top\mathcal{U}\phi$, $G\phi = \neg\mathcal{F}\neg\phi$, $\phi\mathcal{R}\psi = \neg(\phi\mathcal{U}\neg\psi)$.

*c) Co-safe LTL formulas:* A particular class of LTL formulas called as *co-safe* LTL formulas can be used to describe finite horizon specifications of the system [10]. Informally, these are the LTL formulas such that any good trace satisfying the formula has a finite good prefix. A finite good prefix for a formula is a finite prefix such that all its trace extensions satisfy the formula [10]. A class of co-safety formulas that are easy to characterize are the syntactically co-safe LTL formulas. Syntactically co-safe LTL formulas are the LTL formulas that contain only $\mathcal{X}, \mathcal{F}$, and $\mathcal{U}$ as the temporal operators, when written in positive normal form (i.e, the negation operator $\neg$ occurs only in front of atomic propositions, see [10] for more details). For the case of syntactically co-safe LTL formulas, it has been shown that an NFA can be constructed (with at most exponential blowup) that describes all the finite good prefixes satisfying a syntactically co-safe LTL formula [10].