Sampling-based Motion Planning with Temporal Goals

Amit Bhatia Lydia E. Kavraki Moshe Y. Vardi

Abstract— **This paper presents a geometry-based, multi-layered synergistic approach to solve motion planning problems for mobile robots involving temporal goals. The temporal goals are described over subsets of the workspace (called propositions) using temporal logic. A multi-layered synergistic framework has been proposed recently for solving planning problems involving significant discrete structure. In this framework, a high-level planner uses a discrete abstraction of the system and the exploration information to suggest feasible high-level plans. A low-level sampling-based planner uses the physical model of the system, and the suggested high-level plans, to explore the state-space for feasible solutions. In this paper, we advocate the use of geometry within the above framework to solve motion planning problems involving temporal goals. We present a technique to construct the discrete abstraction using the geometry of the obstacles and the propositions defined over the workspace. Furthermore, we show through experiments that the use of geometry results in significant computational speedups compared to previous work. Traces corresponding to trajectories of the system are defined employing the sampling interval used by the low-level algorithm. The applicability of the approach is shown for second-order nonlinear robot models in challenging workspace environments with obstacles, and for a variety of temporal logic specifications.**

I. Introduction

Traditional motion planning algorithms have considered the problem of constructing a motion plan for a given robot model, such that the plan takes the robot from a set of initial states to a set of goal states while avoiding obstacles in the workspace [1], [2]. A class of planning approaches have been proposed recently that use a richer framework to specify complex temporal goals like coverage, ordering of events, etc [3]–[8] using formalisms like Linear Temporal Logic (LTL) (cf. [9]). As an example, the temporal goal "*Eventually visit region A followed by a visit to region B or region C*", can be easily expressed using LTL. An approach for motion planning with deterministic μ-calculus specifications has been proposed recently in [10].

To deal with LTL constraints, a hierarchical approach for motion planning for point-mass linear robot models with temporal goals has been proposed recently in [7]. The focus of the approach is on the construction of provably correct motion plans. Currently, the approach can handle second-order linear robot models and more work needs to be done to extend the approach for nonlinear robot models. An approach combing synthesis technique with receding horizon control has also been proposed recently for linear robot models [8].

The focus of this paper is motion planning problems involving, nonlinear robot models with finite geometry, complex workspace environments, and high-level temporal goals. To the best of our knowledge, such a general version of the motion planning problem has not been examined before in the literature. We propose a geometry-based, multi-layered synergistic approach that trades some of the completeness guarantees provided by [7] to efficiently solve the problem. The idea of trading strong completeness guarantees for efficiency and scalability has also been used successfully for solving challenging instances of traditional motion planning problems using sampling-based algorithms [1], [2]. Our approach uses sampling-based algorithms within a multi-layered synergistic framework that has been proposed recently for solving a variety of planning problems [6], [11], [12]. The framework introduces a discrete component to the search procedure by synergistically utilizing the discrete structure present in the problem. The framework consists of following steps: a) Construction of a discrete abstraction for the system, b) High-level planning for the abstraction using the specifications and the exploration information from the low-level planner, c) Low-level sampling-based planning using physical model and the suggested high-level plans. The construction of the discrete abstraction, and the two-way synergisitc interaction between the layers are critical issues, that affect the overall performance of the approach.

In this paper, we suggest an instantiation of the above framework for solving motion planning problems involving complex robot models and high-level temporal goals. The work presented in this paper differs from [6], [11] in the following ways. First, [11] used the geometry of the obstacles but did not address problems involving temporal goals. The work reported in [6] did not consider obstacles and did not address the issue of the construction of the discrete abstraction, but assumed it to be user defined. Benchmark problems with simple specifications that allowed the user to readily provide an abstraction were considered. In this paper, we address the issue of the construction of the discrete abstraction. We propose a geometric approach to construct the discrete abstraction using the geometry of the obstacles as well as the propositions. It is shown through experiments that there is a significant computational advantage in using geometry to construct the abstraction. Second, both our work and [6] use a discrete-time approximation of the continuous-time dynamics to construct discrete traces. However, [6] considers the class of discrete traces where every two neighboring elements in the trace are required to be different. We do not impose this requirement, and as a result our approach to construct discrete traces is more general than [6].

We consider a variety of LTL planning problem instances involving second-order nonlinear robot models, and investigate the effect of vehicle dynamics, the properties of the LTL formula, and the size of the abstraction on the computational performance of the approach. We further show through

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

experiments that the synergistic interaction between different layers is indeed necessary, and that approaches based on one-way interaction (e.g., using an external monitor for the specification) scale poorly with the size of the specification.

II. Mathematical Framework

A. Robot model

We consider the class of robot models R that can be represented as: $\dot{x}(t) = f(x(t), u(t))$, $x(t) \in X \subset \mathbb{R}^n$, $u(t) \in U \subset \mathbb{R}^m$. $\tilde{x}(x_0, \tilde{u})$ denotes the trajectory of the system starting from x_0 and under the effect of input signal $\tilde{u}$. t_f denotes the terminal time for the trajectory which is assumed to be finite. A state on the trajectory $\tilde{x}(x_0, \tilde{u})$ reached at time $t \in [0, t_f]$ is denoted as $x(x_0, \tilde{u}, t)$. $W \subset \mathbb{R}^2$ is the robot's workspace with $W_{\text{free}} = W \setminus W_{\text{obs}}$ denoting collision-free positions for the robot and W_{obs} denoting the set of (polygonal) obstacles in the workspace. $h_R : X \to W$ maps each state $x \in X$ of the robot to the workspace W.

B. Temporal logic

Linear temporal logic is a propositional logic that is used to describe modalities of time along trajectories of a given system (for details we refer the reader to [9]). Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ be a set of boolean atomic propositions. 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 $\sigma_i = \tau_0, \tau_1, \ldots, \tau_{i-1}$. σ_i is a prefix of the trace σ. $\sigma \models \phi$ indicates that σ satisfies the formula ϕ.

Syntactically co-safe LTL formulas: The focus of this paper is motion planning problems over a finite time horizon. The traces generated by trajectories of the system are finite, but, under some restrictions, LTL formulas can be interpreted over finite traces as we describe next. A particular class of LTL formulas are known as co-safety formulas [13]. 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 (cf. [13]). 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 the $\mathcal{X}, \mathcal{F}, \mathcal{U}$ operators, when written in positive normal form (i.e., the negation operator $\neg$ occurs only in front of atomic propositions, see [13] for more details). For this paper, we limit our attention to syntactically co-safe LTL formulas.

Model checking LTL specifications: One of the most widely used methods to check whether a given system M satisfies an LTL formula ϕ is through an automaton construction [9]. A Büchi automaton $A_{\neg\phi}$ is first constructed whose language is exactly the set of runs that violate ϕ. The product of $A_{\neg\phi}$ and M is then checked for feasible runs. A feasible run on $A_{\neg\phi} \times M$ gives a feasible trace for M that violates ϕ. For constructing feasible motion plans satisfying the specification, we instead consider the automaton A_{ϕ}.

NFA for syntactically co-safe LTL: Given a set of atomic propositions Π, and a syntactically co-safe LTL formula ϕ, it has been shown that an NFA can be constructed (with at most exponential blowup), that describes all the finite prefixes satisfying the formula ϕ [13]. The NFA 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 $Z_{\text{acc}} \subset Z$ is the set of accepting final states. The set of states on which $\mathcal{A}_{\phi}$ ends when run on a trace $\sigma = \tau_0, \tau_1, \ldots, \tau_k$, is:

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

For the formula $\phi = F(p_1 \wedge F(p_2 \vee p_3))$, the corresponding NFA is shown in Figure 1[1]. 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. [6], [14]). In light of this result, we use minimized DFA in the work presented in this paper.

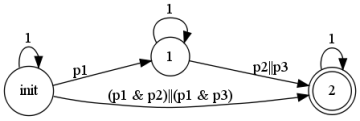
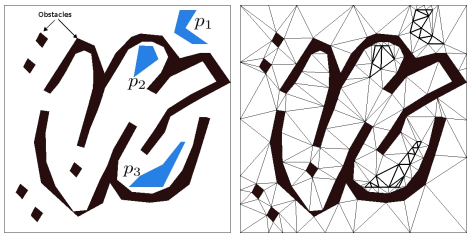


Fig. 1. NFA $\mathcal{A}_{\phi}$ describing all the finite good prefixes for the syntactically co-safe LTL formula $\phi = F(p_1 \wedge F(p_2 \vee p_3))$.



(a) Workspace (b) Decomposition

Fig. 2. (a) A workspace with obstacles and three propositions p_1, p_2, p_3. (b) A triangulation-based decomposition of the workspace.

C. Specifications for robots

The specifications considered in this paper are expressed as co-safe LTL formulas using finite number of atomic propositions. Let $\Pi = \{p_0, p_1, p_2, \ldots, p_N\}$ denote the set of boolean atomic propositions. Each proposition denotes a region of interest in the workspace for the robot. $\Gamma : W_{\text{free}} \to 2^{\Pi}$ maps each point $w \in W_{\text{free}}$ to the set of propositions that hold there. 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 $w \in W_{\text{free}} \setminus \Gamma^{-1}(p)$[2]. In Figure 2(a), we show an example with three propositions in the workspace together with the obstacles.

Note that we do not describe obstacles as propositions in our work for the following reason. The geometric constraints arising due to obstacles, and the finite geometry of the robot, are checked using collision-detection scheme by the low-level planner. The satisfaction of a proposition on the other hand, is evaluated by treating the robot as a point mass.

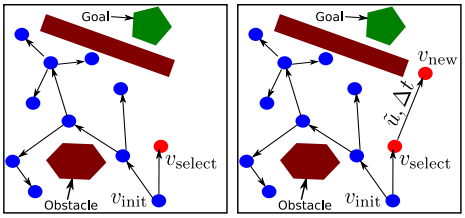
Trajectory Traces: A trajectory of the robot $\tilde{x}(x_0, \tilde{u})$ is defined over time horizon $[0, t_f] \subset \mathbb{R}^+$, and is constructed using a sampling-based algorithm in the proposed approach.

[1]We recommend the online version of the paper for viewing figures.

[2]If the set $W_{\text{free}} \setminus \Gamma^{-1}(p)$ is non-convex, it can still be represented as union of finite number of convex sets.

To interpret an LTL formula, we need an appropriate notion of traces generated by such trajectories.

For systems with dynamics, a sampling-based motion planning algorithm typically generates a tree of feasible trajectories $G_R = (V_R, E_R)$ where each vertex $v \in V_R$ represents a feasible state $v.x$ for the robot and each edge $e \in E_R$ connecting a pair of vertices (v, v') represents a control $\tilde{u}$ and a time step Δt, such that $v'.x = x(v.x, \tilde{u}, \Delta t)$. Many instantiations of such algorithms exist [1], [2], [15]–[19]. In Figure 3, we show an iteration of one such algorithm. v_{init} denotes the root of the tree at initial state of the robot. A node v_{select} is selected for expansion. A new vertex v_{new} and the resulting state $v_{\text{new}}.x = x(v_{\text{select}}.x, \tilde{u}, \Delta t)$ is generated using the robot dynamics, together with a suitably chosen time-step Δt and a control $\tilde{u}$ (cf. [2]). Typically, $\tilde{u}$ is a constant input $u \in U$, applied for time step Δt.



(a) Node selection (b) Tree expansion

Fig. 3. An exploration step in a general, sampling-based algorithm

Since the search tree G_R stores feasible states at sampling instants, and the proof of correctness for our approach (discussed in Section III-D) uses the search tree, a suitable framework is needed to describe the sampling nature of our approach. We do so by defining the notion of sampled discretization as follows.

Definition 1 (Sampled discretization): Given a trajectory $\tilde{x}(x_0, \tilde{u})$, and a sampling interval $\Delta t \in \mathbb{R}_+$, with $t_f = i_f \Delta t$ for some $i_f \in \mathbb{N}$, the sequence $\tilde{x}^{\Delta t}(x_0, \tilde{u}) = \{x(x_0, \tilde{u}, i\Delta t)\}_{i=0}^{i_f}$ is a sampled discretization of $\tilde{x}(x_0, \tilde{u})$.

Given a sampled discretization of a trajectory $\tilde{x}^{\Delta t}(x_0, \tilde{u})$ we can associate a *trace* as follows. The trace denotes the sequence of sets of atomic propositions that hold true at sampling instants on a given trajectory.

Definition 2 (Trace): Given a sampled discretization $\tilde{x}^{\Delta t}(x_0, \tilde{u})$ of robot trajectory $\tilde{x}(x_0, \tilde{u})$ with $t_f = i_f \Delta t$, the trace generated by the discretization is defined as $\sigma(\tilde{x}^{\Delta t}(x_0, \tilde{u})) = \tau_0, \tau_1, \ldots, \tau_{i_f}$, with $\tau_i = \Gamma(h_R(x(x_0, \tilde{u}, i\Delta t))$.

Note that our approach to construct traces is more general than the one used in [6]. In [6], every two neighboring elements τ_i, τ_{i+1} of the trace are required to be different.

Semantics: The LTL semantics here is defined with respect to the finite traces generated by sampled discretization of the trajectories. These traces are finite, but, under the co-safety restriction, LTL formulas can be interpreted over finite traces.

Definition 3 (LTL semantics): Given a trajectory $\tilde{x}(x_0, \tilde{u})$, a sampling interval Δt, and an LTL formula ϕ, we say that the trajectory satisfies the formula, denoted as $\tilde{x}(x_0, \tilde{u}) \models_{R,\Delta t} \phi$, if $\mathcal{A}_\phi(\sigma(\tilde{x}^{\Delta t}(x_0, \tilde{u})))$ contains an accepting state.

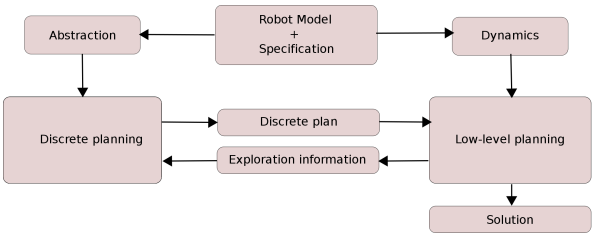


Fig. 4. Multi-layered synergistic approach for planning

D. Problem definition

Definition 4 (LTL motion planning problem $\mathcal{P}$): Given a robot model R, a sampling interval Δt, and a syntactically co-safe LTL formula ϕ, construct a robot trajectory $\tilde{x}(x_0, \tilde{u})$, such that $\tilde{x}(x_0, \tilde{u}) \models_{R,\Delta t} \phi$.

III. Multi-layered Synergistic Planning

Sampling-based motion planning approaches such as PRM [20], RRT [15], [16], EST [17], [18] and PDST [19], have been widely used in motion planning problems involving complex kinematic and differential constraints. An iteration of one such algorithm (for systems with dynamics) was briefly discussed in Section II (shown in Figure 3). A key feature of these approaches is that they are single-layered, i.e., exploration happens directly in the state-space of the system.

The multi-layered framework discussed in Section I, is inspired by earlier works [21], [22] that introduced a discrete search component for solving planning problems. The multi-layered framework is different from the earlier works [21], [22] and the LTL planning approaches [4], [7], [23] in that there is a two-way, synergistic interaction between different layers of planning. Our implementation of the framework is shown in Figure 4, and consists of four steps. 1) Construction of discrete abstraction for the robot motion, 2) Discrete planning for the abstraction using temporal logic specification and low-level exploration information, 3) Exploration of robot's state-space using high-level discrete plan as a guide, 4) Termination if solution found (or computation time exceeds the limit) in step 3, or else feedback of exploration information from step 3 to step 2, and repeat from step 2.

In the context of temporal logic specifications, previous work [6] did not address the issue of the construction of the discrete abstraction, but instead assumed it to be defined by the user, and presented simple benchmarks where the abstraction can be easily defined. In our work, we automatically construct the discrete abstraction and advocate the use of the geometry of the specifications for its construction.

A. Construction of discrete abstractions

As mentioned in Section I, the construction of the discrete abstraction is a critical step in the approach of Figure 4, and it significantly affects the performance of the approach. A way to construct such abstractions is by using the geometric structure present in the planning problem. For traditional motion planning problems, this geometric structure is induced by obstacles in the workspace and for problems involving temporal goals the geometric structure is also induced by the observation map Γ. In this paper, we employ discrete

abstractions that use a triangulation-based decomposition of the workspace. We wish to remark here that the idea of using a triangulation-based decomposition of the workspace has been used before (cf. [7], [24]). The abstractions used in [7], [24] need to satisfy the bisimilarity property [25], while in our work, this is not required.

Definition 5 (Workspace decomposition): Given a robot workspace $W \subset \mathbb{R}^2$, a decomposition $D = \cup_{i=1}^{N_D} D_i$ is a partition of workspace into number of equivalence classes defined by the map $\Upsilon_D : W \rightarrow D$.

A well-defined decomposition should also satisfy the following property. This is a standard requirement that ensures that the observation map Γ is also well defined for the abstraction (also discussed in [7]).

Definition 6 (Proposition-preserving decomposition): A decomposition D of the workspace W is proposition preserving if for all $w_1, w_2 \in W$, $\Upsilon_D(w_1) = \Upsilon_D(w_2) \Rightarrow \Gamma(w_1) = \Gamma(w_2)$.

The simplest proposition-preserving decomposition is the one that is induced by Γ and ignores the geometry of the specifications. We call such a decomposition as *geometry ignoring*. Every element of 2^{Π} is represented by at most one element of the geometry-ignoring decomposition. However, as we will show through experiments, there is a significant computational advantage in using decompositions that use the geometry of the specifications. To compute such decompositions, we use a triangulation of the workspace. The workspace is given as a Planar Straight Line Graph (PSLG) to a mesh generation package (we use the Triangle package [26]). To ensure that the resulting decomposition is proposition-preserving, the sets describing propositions are given as *segments*. The obstacles are given as *holes*. One such decomposition for the example considered in Figure 2(a) is shown in Figure 2(b). Such decompositions will be referred as *geometry-using* decompositions. Given a decomposition of the the workspace, an abstraction of robot model can be constructed as follows. With slight abuse of notation, we define $\Gamma(\Upsilon_D^{-1}(d)) = \Gamma(w)$, where $d = \Upsilon_D(w), w \in W$.

Definition 7 (Robot model abstraction): Given a robot workspace W and a decomposition D of the workspace, $M = (D, d_0, \rightarrow_D, h_D)$ is an abstraction of the robot model. D is the set of states, $d_0 \in D$ is the initial state of the abstraction, satisfying the property $\Upsilon_D(h_R(x_0)) = d_0$. $\rightarrow_D \subset D \times D$ is the transition relation. $d_i \rightarrow_D d_j$ iff $\Upsilon_D^{-1}(d_i)$ and $\Upsilon_D^{-1}(d_j)$ share an edge in the workspace W. $h_D : D \rightarrow \Pi$ is the observation map, defined as $h_D(d) = \Gamma(\Upsilon_D^{-1}(d))$.

The abstraction will be called as *geometry ignoring* or *geometry using*, depending on the kind of decomposition used. We again wish to emphasize that unlike [7], [24], the abstractions used in our work do not need to satisfy the bisimilarity property [25].

B. Discrete planning

Given a discrete abstraction M for a robot model R, and an LTL formula ϕ defined over the set of propositions Π, the (discrete) high-level planner searches for promising high-level plans (also called as guides) over the product $\mathcal{A}_\phi.Z \times D$. The idea of using the product $\mathcal{A}_\phi.Z \times D$ for discrete planning has also been used before (cf. [4], [7], [23]). An important difference in our approach is that the edges in the graph representation of $\mathcal{A}_\phi.Z \times D$ are assigned weights (called as *feasibility estimates*). These are used to synergistically convey the low-level exploration information to the high-level layer. We now introduce the notion of a high-level state, and a high-level plan.

Definition 8 (High-level state): For a given LTL formula ϕ defined over set of propositions Π, a pair $(z, d) \in \mathcal{A}_\phi.Z \times D$ is a high-level state.

A high-level state (z, d) will be called as accepting if z is an accepting state of the automaton. A feasible high-level plan ζ is defined over the set of high-level states as follows.

Definition 9 (Feasible high-level plan): A high-level plan ζ is a sequence $(z_i, d_i)_{i=1}^{k}$, such that each (z_i, d_i) is a high-level state, $d_i \rightarrow_D d_{i+1}, \forall i \in [1, k-1]$, $d_1 = d_0$, and $z_i \in \delta(z_{i-1}, h_D(d_i))$ and $z_k \in \mathcal{A}_\phi.Z_{\text{acc}}$.

A high-level plan is based on the formula ϕ and the abstraction M. It does not take into account the dynamics of the underlying robot model. Moreover, there can be potentially many candidate feasible high-level plans and in some cases none. The high-level planner attempts to take the dynamics of the underlying robot model into account through the feasibility estimate defined for each high-level state. This also helps the high-level planner to suggest more promising plans amongst possibly many candidate plans. The following notion is adapted from [6].

Definition 10 (Feasibility estimate): Given a high-level state (z, d), the feasibility estimate associated with it is given by weight $w(z, d)$ defined as:

$w(z,d) = \frac{(\text{cov}\,(z,d)+1) * \text{vol}\,(\Upsilon_D^{-1}(d))}{\text{w}\,(z) * (\text{nrsel}\,(z,d)+1)^2}$, where,

$\text{cov}\,(z, d)$ estimates the coverage of the region $\Upsilon_D^{-1}(d)$ from the tree vertices associated with (z, d), $\text{vol}\,(\Upsilon_D^{-1}(d))$ is the volume of the region $\Upsilon_D^{-1}(d)$ and $\text{nrsel}\,(z, d)$ is the number of times the high level state (z, d) has been selected in the past for further exploration. $\text{w}\,(z)$ computes the shortest distance of the state z from the set of accepting states Z_{acc}. Thus, the discrete planner associates a high weight to a pair (z, d) if the region $\Upsilon_D^{-1}(d)$ has a larger volume, the state z is closer to the set of accepting states Z_{acc}, and the low-level sampling-based motion planner makes quick progress in covering the region $\Upsilon_D^{-1}(d)$ with tree vertices.

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 $(w(z_i, d_i) * w(z_j, d_j))^{-1}$. The discrete planner computes the shortest path from an (abstract) initial 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.

C. Algorithm

We now discuss the main algorithm (called as `ML-LTL-MP`, and shown in Figure 5), and later discuss details of the low-level planner. The `ML-LTL-MP` algorithm is different from the algorithm proposed in [6] in that the construction of the discrete abstraction is also part of our algorithm. (*Lines: 3,4*, Figure 5) and not an input.

Input: The algorithm takes as an input the problem $\mathcal{P}$, the decomposition map Υ_D, a time-step Δt, the bound on simulation time t_{max}, and the bound on exploration time $t_{explore}$ for each call to the low-level planning algorithm.

ML-LTL-MP($\mathcal{P}$, Υ_D, Δt, $t_{explore}$, t_{max})

1 $\mathcal{A}_\phi \leftarrow$ COMPUTE_AUTOMATON($\mathcal{P}.\phi$) {Compute automaton}
2 $\mathcal{T} \leftarrow$ INIT($\mathcal{P}.R.x_0, \mathcal{A}_\phi$) {Initialize the search tree}
3 $D \leftarrow$ DECOMPOSE($\mathcal{P}.R, \Upsilon_D$) {Compute decomposition}
4 $M = G(V, E) \leftarrow$ COMPUTE_ABSTRACTION(D)
5 $w \leftarrow$ INITIALIZE_ESTIMATES($\mathcal{A}_\phi, M$)
6 $clck \leftarrow 0$ { Initialize timer}
7 $(solution, \tilde{x}, \tilde{u}) \leftarrow (\bot, \emptyset, \emptyset)$
8 **while** $(clck \leq t_{max} \wedge \neg solution)$ **do**
9 $\quad\zeta \leftarrow$ DISCRETE_PLANNER($M, \mathcal{A}_\phi, w$)
10 $\quad(\mathcal{T}, w, solution, v_{\text{acc}}) \leftarrow$ EXPLORE($\mathcal{P}$,$\mathcal{T}$,M,w,ζ,$t_{explore}$,Δt)
11 $\quad$**if** $(solution)$ **then**
12 $\quad\quad(\tilde{x}^{\Delta t}(\tilde{u}, x_0), \tilde{u}) \leftarrow$ CONSTRUCT_DISCRETIZATION($\mathcal{T}, v_{\text{acc}}$)
13 return $(\mathcal{T}, solution, \tilde{x}^{\Delta t}(\tilde{u}, x_0), \tilde{u})$

Fig. 5. Multi-layered synergistic interaction-based LTL Planning algorithm

Output: The algorithm returns a boolean variable *solution* that is true if a solution is found. If a solution is found, then $\tilde{x}^{\Delta t}(\tilde{u}, x_0)$ is the solution with the corresponding control $\tilde{u}$.

Data structures: The search tree $\mathcal{T}$ is stored as a directed 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 $v'.x$ of the robot R 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 a control $\tilde{u}$, and time duration $t = \Delta t$ such that $v'.x = x(v.x, \tilde{u}, t)$. 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. ζ denotes a high-level plan. With slight abuse of notation, we use w to also denote the data-structure holding the feasibility estimates for each of the high-level states.

Let $\mathcal{T}(v_{\text{init}}, v)$ denote the sequence of vertices connecting the root of the tree v_{init} to the vertex v. Let $\hat{x}(\mathcal{T}(v_{\text{init}}, v))$ denote the corresponding sequence of robot states. $v.\alpha = \mathcal{A}_\phi(\sigma)$, stores the state of the automaton $\mathcal{A}_\phi$ when run on the trace $\sigma = \Gamma(h_R(\hat{x}(\mathcal{T}(v_{\text{init}}, v))))$. v_{acc} is a tree vertex such that $v_{\text{acc}}.\alpha \cap \mathcal{A}_\phi.Z_{\text{acc}} \neq \emptyset$.

We now discuss each line of shown in Figure 5. *Line 1:* COMPUTE_AUTOMATON($\mathcal{P}.\phi$) computes the minimized DFA $\mathcal{A}_\phi$ corresponding to the co-safe LTL specification $\mathcal{P}.\phi$. *Line 2:* INIT($\mathcal{P}.R.x_0, \mathcal{A}_\phi$) initializes the search tree $\mathcal{T}$ with a vertex v_{init} corresponding to the initial state $\mathcal{P}.R.x_0$ and the corresponding automaton state $v_{\text{init}}.\alpha$. *Line 3:* DECOMPOSE($\mathcal{P}.R, \Upsilon_D$) computes the decomposition D using the map Υ_D (see Section III-A). *Line 4:* COMPUTE_ABSTRACTION(D) computes the abstraction M corresponding to the decomposition D (see Section III-A). *Line 5:* INITIALIZE_ESTIMATES($\mathcal{A}_\phi, M$) initializes the feasibility estimates $w(z, d)$ for each high-level state (z, d). *Line 6:* The timer is initialized by setting $clck$ to 0. *Line 7:* The boolean variable $solution$ and the data structures $\tilde{x}, \tilde{u}$ for storing a solution are initialized. *Line 8:* The search for a feasible solution begins. *Line 9:* DISCRETE_PLANNER($M, \mathcal{A}_\phi, w$) computes a high-level plan ζ (see Section III-B). *Line 10:* EXPLORE($\mathcal{P}$,$\mathcal{T}$,M,w,ζ,$t_{explore}$,Δt) invokes the low-level planner to search for a feasible solution guided by the high-level plan ζ for a maximum of $t_{explore}$ time. EXPLORE is described later and shown in Figure 6. EXPLORE returns with variable $solution$ (*true* if solution is found), the updated tree $\mathcal{T}$ and the updated feasibility estimates w. Additionally, if a solution is found, then v_{acc} is marked with an accepting state of the automaton. *Line 11:* Check if a solution was found by the low-level planner. If $solution$ is true, then Line 12 is executed, else the procedure is repeated from *Line 8*. *Line 12:* CONSTRUCT_DISCRETIZATION($\mathcal{T}, v_{\text{acc}}$) constructs a sampled discretization $(\tilde{x}^{\Delta t}(\tilde{u}, x_0), \tilde{u})$ using the set of states $\hat{x}(\mathcal{T}(v_{\text{init}}, v_{\text{acc}}))$.

We now explain the low-level sampling-based exploration algorithm EXPLORE, shown in Figure 6. The algorithm differs from the one used in [6] in two major ways. First, our approach for generating discrete traces is more general compared to [6] (*Lines:11,15*, Figure 6; see Sections I, II). Second, our problem setup involves finite geometry of the robot and the presence of obstacles in the workspace. Hence, we also need to include collision detection in the low-level exploration (*Line:11*, Figure 6).

EXPLORE($\mathcal{P}$,$\mathcal{T}$,M,w,ζ,$t_{explore}$,Δt)

1 $(solution, v_{\text{acc}}, clck) \leftarrow (\bot, \emptyset, 0)$
2 $\sigma_{avail} \leftarrow$FEASIBLE_HIGHLEVEL_STATES($\zeta, \mathcal{T}$)
3 **while** $(clck < t_{explore} \wedge \neg solution)$ **do**
4 $\quad(z_{select}, d_{select}) \leftarrow$ SELECT_HIGHLEVEL_STATE(w, σ_{avail})
5 $\quad v_{select} \leftarrow$ SELECT_VERTEX $(w, (z_{select}, d_{select}))$
6 $\quad x_0 \leftarrow v_{select}.x$
7 $\quad u \leftarrow$ CHOOSE_INPUT $(\mathcal{P}.R.U, \mathcal{T}, v_{select})$
8 $\quad nsteps \leftarrow$ CHOOSE_STEPS($\mathcal{P}.R$)
9 $\quad$**for** $i = 0, 1, \ldots, nsteps$ **do**
10 $\quad\quad(x_{i+1}, v_{i+1}) \leftarrow$SIMULATE_DYNAMICS $(\mathcal{P}.R, x_i, u, \Delta t)$
11 $\quad\quad$**if** CHECK_FEASIBILITY($\mathcal{P}.R, \mathcal{A}_\phi, v_{i+1}$) **then**
12 $\quad\quad\quad\mathcal{T} \leftarrow$UPDATE_TREE($\mathcal{T}, v_{i+1}, u, \Delta t$)
13 $\quad\quad\quad\sigma_{avail} \leftarrow$UPDATE_FEASIBLE_HIGHLEVEL_STATES(ζ, v_{i+1})
14 $\quad\quad\quad w \leftarrow$ UPDATE_FEASBILITY_ESTIMATES(w, v_{i+1})
15 $\quad\quad\quad$**if** $(v_{i+1}.\alpha \cap P.\mathcal{A}_\phi.Z_{\text{acc}} \neq \emptyset)$ **then**
16 $\quad\quad\quad\quad(solution, v_{\text{acc}}) \leftarrow (\top, v_{i+1})$
17 $\quad\quad\quad\quad$break
18 $\quad\quad$**else**
19 $\quad\quad\quad$break
20 return $(\mathcal{T}, w, solution, v_{\text{acc}})$

Fig. 6. Low-level sampling-based exploration algorithm

Input: The algorithm takes as an input the problem $\mathcal{P}$ the search tree $\mathcal{T}$, abstraction M, feasibility estimates w, suggested high-level plan ζ, bound on exploration time $t_{explore}$ and time-step used for simulation of dynamics Δt.

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

Data structures: In addition to the previously defined data structures, the following are the ones used by the low-level algorithm. For a given high-level state (z, d), let $(z, d).vertices = \{v \in V_{\mathcal{T}} : z \in v.\alpha, h_R(v.x) \in \Upsilon_D^{-1}(d)\}$. 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.

Line 1: The variables $solution$ and v_{acc} are initialized and the local timer $clck$ is initialized. *Line 2:* FEASIBLE_HIGHLEVEL_STATES($\zeta, \mathcal{T}$) identifies the high-level states (z_i, d_i), such that $(z_i, d_i).vertices \neq \emptyset$. All such high-level states are stored in σ_{avail}. *Line 3:* The low-level planning algorithm starts exploration of the state-space for a feasible solution. *Line 4:* SELECT_HIGHLEVEL_STATE(w, σ_{avail}) selects a high-level state (z_{select}, d_{select}) for further exploration (and $(z_{select}, d_{select}).nsel$ is incremented by 1). A high-level state (z, d) is selected with probability $w(z, d)/\Sigma_{(z,d)\in\sigma_{avail}} w(z, d)$. *Line 5:* SELECT_VERTEX $(w, (z_{select}, d_{select}))$ selects a vertex v_{select} from $(z_{select}, d_{select}).vertices$ favoring the vertices that have been selected fewer times so far in the exploration. *Line 6:* The local variable x_0 is initialized to the robot state $v_{select}.x$.

Line 7: CHOOSE_INPUT $(\mathcal{P}.R.U, \mathcal{T}, v_{select})$ chooses an input u from the input set $\mathcal{P}.R.U$. *Line 8:* CHOOSE_STEPS($\mathcal{P}.R$) chooses the number of time steps $nsteps$ for which the robot dynamics are to be incrementally simulated with constant input u. For our work, we have kept this variable to a fixed value given by the user. *Line 9:* Start the incremental simulation of dynamics. *Line 10:* SIMULATE_DYNAMICS $(\mathcal{P}.R, x_i, u, \Delta t)$ computes the state of the robot at time Δt starting from x_i and using the input u. *Line 11:* CHECK_FEASIBILITY($\mathcal{P}.R, \mathcal{A}_\phi, v_{i+1}$) returns true if $v_{i+1}.x$ satisfies the geometric constraints, and the set of automaton states $v_{i+1}.\alpha$ is non-empty. *Line 12:* UPDATE_TREE($\mathcal{T}, v_{i+1}, u, \Delta t$) updates the tree with the new vertex and the corresponding time-step and input. $v_i.nsel$ is incremented by 1. *Line 13:* UPDATE_FEASIBLE_HIGHLEVEL_STATES(ζ, v_{i+1}) updates the data-structure σ_{avail} to account for the fact that the vertex v_{i+1} is also available for expansion in future. *Line 14:* UPDATE_FEASBILITY_ESTIMATES(w, v_{i+1}) updates the feasibility estimates for the high-level states taking into account the addition of vertex v_{i+1} to the tree. *Line 15-16:* If $v_{i+1}.\alpha \cap \mathcal{A}_\phi.Z_{\text{acc}} \neq \emptyset$, then $solution$, v_{acc} are set to *true* and v_{i+1} respectively.

D. Correctness of the algorithm

The changes introduced in the construction of the discrete traces, and the semantics used in our approach (see Section II), allow us to prove correctness of the proposed algorithm (in contrast to [6]) as follows.

Theorem 1 (Correctness of algorithm): Given an LTL motion planning problem $\mathcal{P}$, if the ML-LTL-MP algorithm returns with $solution = \top$ and a control $\tilde{u}$, then $\tilde{x}(x_0, \tilde{u})$ is a solution to the problem.

Proof: Consider the sampled discretization $\tilde{x}^{\Delta t}(\tilde{u}, x_0)$ constructed by the algorithm. Let $t_f = i_f \Delta t$ be the time horizon of the trajectory $\tilde{x}(\tilde{u}, x_0)$. Let $\mathcal{T}(v_{\text{init}}, v_{\text{acc}})$ be the sequence of vertices from which the sampled discretization is constructed. Consider a vertex $v' \in \mathcal{T}(v_{\text{init}}, v_{\text{acc}}) \setminus v_{init}$ and let v be its parent. $v'.x = x(v.x, u, \Delta t), u \in U$ and $v'.x$ is collision-free and $v'.x \in X$ (line 11, Figure 6). Hence, $v'.x$ is a feasible state for the robot. Moreover, $v.\alpha \neq \emptyset$ for all $v \in \mathcal{T}.V$ (*Line:11*, Figure 6) and $v_{\text{acc}}.\alpha \cap \mathcal{P}.\mathcal{A}_\phi \neq \emptyset$.

Hence it can be concluded that the trace $\sigma = \Gamma(h_R(\hat{x}(\mathcal{T}(v_{\text{init}}, v_{\text{acc}}))))$ satisfies the formula ϕ and that the corresponding trajectory $\tilde{x}(x_0, \tilde{u})$ is a feasible collision-free trajectory for the robot. Hence, $\tilde{x}(x_0, \tilde{u}) \models_{R,\Delta t} \phi$. ■

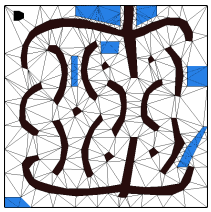
IV. Experimental results

We now discuss the results obtained using the proposed approach, and also do a comparative analysis with some other possible approaches, for a variety of problem instances.

A. Implementation and hardware

The code developed for the experiments presented in this paper is based on the OOPSMP library [27], and builds on top of the code written for [6]. For computing DFAs for syntactically co-safe LTL formulas, we have used the tool scheck2 [28]. For computing triangulations, we have used the package Triangle [26]. All the experiments 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 test run was executed on a single processor with no parallelism.

(a) Workspace



(b) Decomposition

n_{op}	Number of states / Number of transitions		
	Coverage ($\phi_1^{n_{\text{op}}}$)	Sequencing ($\phi_2^{n_{\text{op}}}$)	Strict sequencing ($\phi_3^{n_{\text{op}}}$)
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

(c) Size of minimized DFA $\mathcal{A}_\phi$ for different specifications

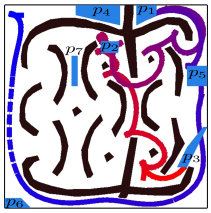
Fig. 7. (a) Workspace with seven propositions used in experiments. The sets shown in blue were labeled with propositions for each run using a random permutation. (b) Conforming Delaunay triangulation-based decomposition of the workspace. See Section IV-B. (c) Size of minimized DFA $\mathcal{A}_\phi$ for different specifications.

B. Experimental setup

Robot models: We have used second-order models of a car, a unicycle, and a differential drive, as robot models for experiments. These models are rich enough to capture the key aspects of the dynamics and have been extensively used for benchmarking planning algorithms for mobile robots. We refer to [6], [11] for more details.

LTL formulas: We have considered the following LTL formulas for the experiments. $\phi_1^{n_{\text{op}}} = \wedge_{i=1}^{n_{\text{op}}}(Fp_i)$, $\phi_2^{n_{\text{op}}} = F(p_1 \wedge F(p_2 \wedge F(p_3 \ldots F(p_{n_{\text{op}}}))))$, $\phi_3^{n_{\text{op}}} = F(p1 \wedge ((p_0 \vee p_1)\mathcal{U}(p_2 \wedge ((p_0 \vee p_2)\mathcal{U} \ldots (p_{n_{\text{op}}}))))$ where $n_{\text{op}} \in [1, 7]$ is the number of temporal operators in the formula. $\phi_1^{n_{\text{prop}}}$ are usually referred to as coverage formulas since they can be used to describe coverage specifications over the workspace of the robot. $\phi_2^{n_{\text{prop}}}$ are referred to as the sequencing formulas and these are used to describe sequencing requirements in the specifications. $\phi_3^{n_{\text{prop}}}$ are the strict sequencing formulas. They are different from sequencing formulas in the fact that the order of visits is strict. Additionally, we have also used the formula $\phi_4 = F(p_1 \wedge F(p_2 \vee p_3))$ in some of the experiments. This is the same formula that was used to describe some of the ideas in Section II. In Figure 7(c), the size of minimized DFA for different kinds of specifications is shown.

Test cases: The workspaces used for the set of experiments are shown in Figure 2(a) and Figure 7. The time-step Δt for constructing sampled discretization was set to 0.25 seconds. We have used conforming Delaunay triangulation for computing the abstraction. The labels for each of the regions corresponding to the propositions were permuted randomly in each of the test run. The robot model (including the bounds on input) were also chosen using uniform distributions. To ensure that the initial state of the robot is always feasible, the



(a) A simulation run for ϕ_3^3

n_{op}	Unicycle			Car			Differential Drive		
	$\phi_1^{n_{\text{op}}}$	$\phi_2^{n_{\text{op}}}$	$\phi_3^{n_{\text{op}}}$	$\phi_1^{n_{\text{op}}}$	$\phi_2^{n_{\text{op}}}$	$\phi_3^{n_{\text{op}}}$	$\phi_1^{n_{\text{op}}}$	$\phi_2^{n_{\text{op}}}$	$\phi_3^{n_{\text{op}}}$
1	1.3	1.8	1.2	2.0	1.4	2.0	7.0	10.8	11.4
2	3.9	3.0	3.9	5.0	10.2	36.3	14.4	14.3	21.0
3	5.6	5.9	12.5	10.9	10.2	78.4	32.7	27.6	32.5
4	14.0	7.5	27.3	22.4	21.1	112.3	78.5	56.5	46.9
5	24.9	11.0	18.1	39.6	33.6	146.2	116.6	51.5	80.9
6	57.2	17.3	34.1	77.3	33.2	196.9	215.3	78.9	94.7
7	109.8	19.2	41.4	153.6	29.7	211.5	429.2.0	97.9	159.3

(b) Performance results (in seconds) for the approach

Fig. 8. Experimental results of the approach for different robot models and specifications. The workspace and the decomposition used for the experiments are shown in Figure 7. For the simulation run, shades closer to blue indicate earlier snapshots of the robot, and those closer to red, the later ones.

initial position of the robot was fixed at $(-0.43, 0.45)$. The remaining components of the initial state of the robot were set to 0. All the computation times are reported in seconds as average over 40 test runs. The maximum time allocated for each simulation t_{max} was set to 900 seconds and the exploration time $t_{explore}$ (for single call to EXPLORE) was set to 0.75 seconds. For the test runs when no solution was found, we have used the upper bound on simulation time t_{max}. For every experiment involving 40 test runs, less than 15% of the runs reported a timeout in finding a solution.

C. Results

We now discuss the performance results of our approach. We have carried out detailed analysis based on three parameters: type of robot model, the properties of the co-safe LTL formula, and the resolution of abstraction.

i) Type of robot model: We have analyzed separately, the performance of the approach for each type of robot model. We wish to remark here that the robot model involves not only the dynamics, but also the geometry of the robot. The workspace used for these set of experiments is shown in Figure 7(a). The experimental results for different types of robot models are shown in Figure 8. For the simulation run, shades closer to blue indicate earlier snapshots of the robot, and closer to red the later ones. The results indicate that the mean computation times for the case of differential-drive are the longest for coverage ($\phi_1^{n_{\text{op}}}$) and sequencing formulas ($\phi_2^{n_{\text{op}}}$). For strict sequencing formulas ($\phi_3^{n_{\text{op}}}$), the mean computation times for the case of car were the longest. The size of the automaton $\mathcal{A}_\phi$ (shown in Figure 7(c)), support the performance trends observed for coverage ($\phi_1^{n_{\text{op}}}$) and sequencing formulas ($\phi_2^{n_{\text{op}}}$). However, note that the computation times for low-level exploration are affected by not only the dynamics, but also the geometry of the robot. This helps to explain the performance trends for the strict sequencing formulas ($\phi_3^{n_{\text{op}}}$).

ii) Properties of the specifications: The performance results shown in Figure 8 (and later in Figure 9) indicate that the performance of the approach is affected by not only the length of the formula but also the type of temporal operators in the formula. Problems involving coverage formulas ($\phi_1^{n_{\text{op}}}$) with about 6-7 temporal operators take the longest to solve. We refer to Figure 7(c), and the above discussion on the role of robot model on overall performance of the approach.

iii) Resolution of abstraction: We have also done a preliminary investigation of the role of triangulation parameters on performance of the approach. The workspace shown in Figure 2 was used, with the formula ϕ_4. Different kinds of triangulations were considered, like those constructed using the area constraints on the triangles as well as the conforming Delaunay triangulation. The upper bound on area was varied between 0.5 and 0.0005. The average time taken to find a solution varied between 3.5 and 4.4 seconds. Even though this suggests that the approach is not very sensitive to the exact choice of the triangulation parameters, further work is required to develop a principled approach for choosing the triangulation parameters.

D. Comparative analysis

We now discuss the benefits of using synergy, and geometry in the overall framework, by examining the effect of removing these ideas from the overall framework.

Single-layered, monitor-based (Figure 9, left): The performance obtained using a single-layered, sampling-based approach that relies on an external monitor is shown in Figure 9, at the left. The external monitor keeps track of the state of the automaton on every vertex of the search tree $\mathcal{T}$ and raises a flag if an accepting state of the automaton is reached during exploration (indicating that a solution was found). We skip the details and refer the reader to [6] for the details and the comparison when doing falsification of LTL safety properties for hybrid systems.

Synergistic, geometry-ignoring abstraction (Figure 9, center): The performance results obtained using the geometry-ignoring abstraction (with 8 states) are shown in Figure 9 at the center.

Synergistic, geometry-using abstraction (Figure 9, right): Finally, the performance results obtained with the geometry-using abstraction (with 618 states) are shown in Figure 9 at the right.

Analysis: The results indicate the following. First, a two-way, synergistic interaction between different layers of planning is indeed necessary for scalability. Even for traditional planning problems ($n_{\text{op}} = 1$), the single-layered, monitor-based approach performs 25 times slower than the geometry-based, multi-layered synergistic approach proposed in this paper (cf. [11]). For $n_{\text{op}} \geq 2$, the performance of the single-layered, monitor-based approach degrades significantly. In fact, for $n_{\text{op}} \geq 4$, the single-layered, monitor-based approach does not work.

Second, even though the discrete abstraction can be constructed by ignoring the geometry of the specifications, there is a significant improvement in performance if the abstraction is constructed using the geometry of the specifications. For the case of sequencing ($\phi_2^{n_{\text{op}}}$) and strict sequencing formulas ($\phi_3^{n_{\text{op}}}$), geometry-based abstractions result in speedup of up

Mean computation time (seconds) / Number of successful runs									
n_{op}	Single-layered, monitor-based			Synergistic, geometry-ignoring abstraction			Synergistic, geometry-using abstraction		
	$\phi_1^{n_{\text{op}}}$	$\phi_2^{n_{\text{op}}}$	$\phi_3^{n_{\text{op}}}$	$\phi_1^{n_{\text{op}}}$	$\phi_2^{n_{\text{op}}}$	$\phi_3^{n_{\text{op}}}$	$\phi_1^{n_{\text{op}}}$	$\phi_2^{n_{\text{op}}}$	$\phi_3^{n_{\text{op}}}$
1	51.7 / 40	50.2 / 40	50.2 / 40	4.4 / 40	4.1 / 40	3.8 / 40	2.5 / 40	2.6 / 40	2.8 / 40
2	55.5 / 11	63.4 / 10	63.2 / 9	24.2 / 40	14.1 / 40	29.5 / 40	9.0 / 40	7.5 /40	12.0 / 40
3	63.4 / 5	33.0 / 1	33.1 / 1	24.2 / 40	25.8 / 40	70.8 / 39	18.3 / 40	13.0 / 40	44.2 / 39
4	37.9 / 1	X / 0	X / 0	45.0 / 40	54.9 / 40	114.4 / 39	33.9 / 40	21.9 / 40	41.5 / 40
5	35.7 / 1	X / 0	X / 0	113.1 / 40	115.8 / 38	259.4 / 35	64.3 / 40	24.5 / 40	81.8 / 39
6	X / 0	X / 0	X / 0	213.5 / 39	172.7 / 37	265.7 / 37	119.6 / 40	40.1 / 40	78.0 / 39
7	X / 0	X / 0	X / 0	362.7 / 36	194.8 / 38	322.1 / 36	200.2 / 40	58.5 / 40	155.1 / 38

Fig. 9. Performance results for different approaches (see Section IV-D); (left) single-layered, monitor-based approach using the RRT algorithm; (center) synergistic approach with geometry-ignoring abstractions; (right) synergistic approach with geometry-using abstractions proposed by us. The workspace used for the experiments is shown in Figure 7(a). The robot model was chosen at random, and the labels for propositions were permuted randomly, in each run. The geometry-ignoring abstraction (center) has 8 states and the geometry-using abstraction (right) has 618 states. For the case of single-layered, monitor-based approach (left), the mean computation times are based on successful runs only.

to 3-4 times. For the challenging case of coverage specifications ($\phi_1^{n_{\text{op}}}$) with many temporal operators, the speedup obtained is about 50%. The results indicate that geometry-based abstractions certainly result in significant improvement in computational efficiency. However, more efficient high-level search techniques are needed to take full advantage of such abstractions that are larger in size compared to those constructed without using the geometry of the specifications.

V. Conclusions

In this paper, we have considered motion planning problems involving temporal goals. We have considered a general version of the problem that involves differential constraints arising from the robot dynamics, the temporal logic constraints arising from the specification, and the geometric constraints arising from the obstacles and the propositions (in the workspace). We have proposed a geometry-based multi-layered synergistic approach for solving the problem. A critical step in the approach is the construction of the discrete abstraction, which is the main focus of the paper. We have proposed a way to construct the discrete abstraction using the geometry of the obstacles and the propositions. The framework used to construct discrete traces from continuous-time robot trajectories uses the time-step used by the low-level sampling-based algorithm. The propositions considered here are defined on the workspace. We plan to consider a more general version of the problem involving propositions defined on the state-space of the robot in future.

VI. Acknowledgments

We would like to thank E. Plaku, Y. Lustig, and the reviewers 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] 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.

[4] 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.

[5] S. Karaman and E. Frazzoli. Complex mission optimization for multiple-UAVs using linear temporal logic. In *American Control Conference*, pages 2003–2009, 2008.

[6] 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.

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

[8] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning for dynamical systems. In *IEEE Conference on Decision and Control*, 2009.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. 2000.

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

[11] 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.

[12] 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.

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

[14] 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.

[15] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, 1998.

[16] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[17] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2719–2726, 1997.

[18] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21(3):233–255, 2002.

[19] A. M. Ladd and L. E. Kavraki. Motion planning in the presence of drift, underactuation and discrete system changes. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, 2005.

[20] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics*, 12:566 – 580, 1996.

[21] R. Alami, J. P. Laumond, and T. Siméon. Two manipulation planning algorithms. In *Algorithmic Foundations of Robotics*, pages 109–125, 1995.

[22] C. L. Nielsen and L. E. Kavraki. A two level fuzzy prm for manipulation planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1716–1721, 2000.

[23] 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.

[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] Available at `http://www.kavrakilab.org/OOPSMP`.

[28] T. Latvala. Efficient model checking of safety properties. In *Model Checking Software*, volume 2648 of *LNCS*, pages 74–88. Springer, 2003.