Planning Robot Motions to Satisfy Linear Temporal Logic, Geometric, and Differential Constraints

Erion Plaku
Department of Electrical Engineering and Computer Science
Catholic University of America, Washington DC, 22064

Abstract

This paper shows how to effectively compute collision-free and dynamically-feasible robot motion trajectories that satisfy task specifications given by Linear Temporal Logic (LTL). The proposed approach combines sampling-based motion planning over the continuous state space with discrete search over both the LTL task representation and a workspace decomposition. In distinction from related work, the proposed approach samples the discrete space to shorten the length of the discrete plans and to more effectively guide motion planning in the continuous state space. Experimental results on various scenes, LTL specifications, and a snake-like robot model with nonlinear dynamics and numerous degrees-of-freedom (DOFs) show significant computational speedups over related work.

1 Introduction

Crucial to the goal of enabling robots to complete tasks on their own is their ability to plan at multiple levels of discrete and continuous abstractions. Whether the task is to search, inspect, manipulate objects, or navigate, it generally involves abstractions into discrete actions, which often require substantial continuous motion planning to carry out.

The coupling of the discrete and the continuous, however, poses significant challenges as discrete and continuous planning have generally been treated separately. On the one hand, while discrete planning can take into account sophisticated task specifications, it has generally been limited to discrete worlds (Ghallab, Nau, and Traverso, 2004). On the other hand, while motion planning can take into account obstacles, dynamics, and other continuous aspects of the robot and the world, due to the increased complexity, it has generally been limited to simple tasks, such as planning motions to reach a goal state. (Choset et al., 2005; LaValle, 2006).

To bridge the gap between the discrete and the continuous, researchers are proposing to incorporate task specifications directly into motion planning (Bhatia et al., 2011; Cambon, Alami, and Gravot, 2009; Ding et al., 2011; Fainekos et al., 2009; Hauser and Ng-Thow-Hing, 2011; Kress-Gazit et al., 2007; Plaku and Hager, 2010; Plaku, Kavraki, and Vardi, 2009, 2012; Wolfe, Marthi, and Russell, 2010). More specifically, the problem being studied in this line of research requires generating a collision-free and dynamically-feasible motion trajectory that satisfies a given task specification. In this context, LTL has often been used as the discrete logic in which to express the tasks. LTL makes it possible to express tasks in terms of propositions, logical connectives ($\wedge$ and, $\vee$ or, $\neg$ not), and temporal connectives ($\bigcirc$ next, $\Diamond$ eventually, $\Box$ always, $\cup$ until, $\mathcal{R}$ release). As an illustration, the task of inspecting all the areas of interest can be expressed as

$$\Diamond\pi_{A_1} \wedge \ldots \wedge \Diamond\pi_{A_n}, \tag{1}$$

where π_{A_i} denotes the proposition "robot inspected area A_i." As another example, the task of reaching A_1 before A_2 and A_3 can be expressed as

$$(\neg\pi_{A_2} \wedge \neg\pi_{A_3}) \cup (\pi_{A_1} \wedge \Diamond\pi_{A_2} \wedge \Diamond\pi_{A_3}).$$

The problem of planning motions that satisfy an LTL specification ϕ is often approached by first using model checking to compute a sequence of propositional assignments $\sigma = \tau_1, \tau_2, \ldots$ that satisfies ϕ. Referring again to the LTL specification in Eq. 1, σ could be obtained by setting each $\tau_i = \{\pi_{A_i}\}$, i.e., π_{A_i} is true and every other proposition is false. In a second stage, controllers are used to enable the robot to satisfy the propositions in the order specified by σ. In this context, discrete logic is used in (Arkin, 1990; Payton, Rosenblatt, and Keirsey, 1990; Saffiotti, Konolige, and Ruspini, 1995) to combine a set of behavior schemas which use controllers to link motion to abstract actions. This idea has been revisited more recently in (Kress-Gazit et al., 2011) to synthesize reactive controllers from LTL for car-like systems and in (Ding et al., 2011) to deploy robotic teams in urban environments. The work in (Fainekos et al., 2009) uses LTL to determine the sequence of triangles a point robot needs to visit and relies on a controller to drive the robot between adjacent triangles.

A limitation of these approaches is that it is generally not known in advance which propositional assignments are actually feasible in the continuous world. The LTL specification could present exponentially many alternative discrete solutions, as it is the case in Eq. 1. The underlying assumption in these approaches is that any sequence of propositional assignments that satisfies the LTL specification can be carried out in the continuous world. However, as a result of geometric constraints imposed by obstacle avoidance, the geometric shape of the robot, and the underlying motion dynamics, it may be impossible to carry out certain propositional assignments τ_i. This is in fact one of the main challenges when

incorporating LTL task specifications directly into motion planning, limiting the applicability of these approaches to specific systems and to specific discrete actions for which controllers are available.

In order to be generally applicable, recent work by the author (Plaku and Hager, 2010; Plaku, Kavraki, and Vardi, 2009, 2012) and others (Bhatia et al., 2011) has proposed a two-layered approach that couples the ability of sampling-based motion planning to handle the complexity arising from high-dimensional robotic systems, nonlinear motion dynamics, and collision avoidance with the ability of discrete planning to take into account discrete specifications. While discrete planning guides sampling-based motion planning, the latter feeds back information to further refine the guide and advance the search toward a solution that satisfies the LTL specification. Other approaches that utilize discrete search and sampling-based motion planning have also been developed for multimodal motion planning (Hauser and Ng-Thow-Hing, 2011) and manipulation planning (Nieuwenhuisen, van der Stappen, and Overmars, 2006; Stilman and Kuffner, 2008; Stilman, 2010; Wolfe, Marthi, and Russell, 2010). These other approaches, however, do not take into account LTL specifications.

This paper builds upon the success of combining LTL with sampling-based motion planning (Bhatia et al., 2011; Plaku, Kavraki, and Vardi, 2009, 2012). The search for a collision-free and dynamically-feasible trajectory that satisfies the LTL specification is conducted simultaneously in both the continuous and discrete planning layers. Sampling-based motion planning extends a tree in the continuous state space by adding new trajectories as tree branches. Such trajectories are obtained by sampling input controls and propagating forward the motion dynamics of the robot. Discrete planning guides the sampling-based motion planner by searching over both the LTL formula representation and a workspace decomposition to provide discrete plans as intermediate sequences of propositional assignments that should be satisfied. In distinction from related work (Bhatia et al., 2011; Plaku, Kavraki, and Vardi, 2009, 2012), the proposed approach samples the discrete space to shorten the length of the discrete plans and to more effectively guide motion planning in the continuous state space. Moreover, the search is expanded toward new propositions that enable the sampling-based motion planner to make rapid progress toward obtaining a solution. Experimental results on various scenes, LTL specifications, and a snake-like robot with nonlinear dynamics and numerous DOFs show significant computational speedups over related work.

2 LTL Specifications

Let Π denote a set of propositions, where each $\pi_i \in \Pi$ corresponds to a Boolean-valued problem-specific statement, such as "robot is in area A_i." LTL combines propositions with logical connectives ($\neg$ not, $\wedge$ and, $\vee$ or), and temporal connectives ($\bigcirc$ next, $\Diamond$ eventually, $\Box$ always, $\cup$ until, $\mathcal{R}$ release). A discrete state $\tau_i \in 2^{\Pi}$ denotes all the propositions that hold true in the world. As the world changes, e.g., as the result of robot actions, the discrete state could also change. LTL planning consists of finding a sequence of discrete states $\sigma = [\tau_i]_{i=1}^{n}$ that satisfies a given LTL formula, whose syntax and semantics are defined below.

2.1 LTL Syntax and Semantics

Every $\pi \in \Pi$ is a formula. If ϕ and ψ are formulas, then $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi \cup \psi$, $\phi\mathcal{R}\psi$ are also formulas. Let $\sigma = \tau_0, \tau_1, \ldots$, where each $\tau_i \in 2^{\Pi}$. Let $\sigma^i = \tau_i, \tau_{i+1}, \ldots$ denote the i-th postfix of σ. The notation $\sigma \models \phi$ indicates that σ satisfies ϕ and is defined as

- $\sigma \models \pi$ if $\pi \in \Pi$ and $\pi \in \tau_0$;
- $\sigma \models \neg\phi$ if $\sigma \not\models \phi$;
- $\sigma \models \phi \wedge \psi$ if $\sigma \models \phi$ and $\sigma \models \psi$;
- $\sigma \models \bigcirc\phi$ if $\sigma^1 \models \phi$;
- $\sigma \models \phi \cup \psi$ if $\exists\, k \geq 0$ such that $\sigma^k \models \psi$ and $\forall\, 0 \leq i < k : \sigma^i \models \phi$.

The other connectives are defined as $\texttt{false} = \pi \wedge \neg\pi$, $\texttt{true} = \neg\texttt{false}$, $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$, $\Diamond\phi = \texttt{true} \cup \phi$, $\Box\phi = \neg\Diamond\neg\phi$, and $\phi\mathcal{R}\psi \equiv \neg(\neg\phi \cup \neg\psi)$. More details can be found in (Kupferman and Vardi, 2001).

2.2 Co-Safe LTL and Automata Representation

Since LTL planning is PSPACE-complete (Sistla, 1994), as in related work (Bhatia et al., 2011; Plaku, Kavraki, and Vardi, 2009, 2012), this paper considers co-safe LTL. Co-safe LTL formulas are satisfied by finite sequences of discrete states rather than infinite sequences which satisfy general LTL formulas. Most robotic tasks are finite in nature, hence, the use of co-safe LTL does not limit the general applicability of the approaches. Co-safe LTL formulas can be translated into NFAs (Nondeterministic Finite Automata) with at most an exponential increase in size (Kupferman and Vardi, 2001). As recommended in (Armoni et al., 2005; Plaku, Kavraki, and Vardi, 2012), NFAs are converted into DFAs (Deterministic Finite Automata), which are then minimized. A DFA search has a significantly smaller branching factor, since there is exactly one transition that can be followed from each state for each propositional assignment, while when using an NFA there are generally many more.

Formally, a DFA is a tuple $\mathcal{A} = (Z, \Sigma, \delta, z_{init}, \mathit{Accept})$, where Z is a finite set of states, $\Sigma = 2^{\Pi}$ is the input alphabet, $\delta : Z \times \Sigma \to Z$ is the transition function, $z_{init} \in Z$ is the initial state, and $\mathit{Accept} \subseteq Z$ is the set of accepting states. The state obtained by running $\mathcal{A}$ on $\sigma = [\tau_i]_{i=1}^{n}$, $\tau_i \in 2^{\Pi}$, starting from the state z is defined as

$$\mathcal{A}([\tau_i]_{i=1}^{n}, z) = \begin{cases} z, & n = 0 \\ \delta(\mathcal{A}([\tau_i]_{i=1}^{n-1}, z), \tau_n), & n > 0. \end{cases}$$

$\mathcal{A}$ accepts σ iff $\mathcal{A}(\sigma, z_{init}) \in \mathit{Accept}$. As a result, $\sigma \models \phi$ when the equivalent automaton $\mathcal{A}$ accepts σ.

To facilitate presentation, let *Reject* denote all the rejecting states of $\mathcal{A}$, i.e., states that cannot reach an accepting state. Moreover, let $\delta(z)$ denote all the non-rejecting automaton states connected by a single transition from z, i.e.,

$$\delta(z) = \{\delta(z, \tau) : \tau \in 2^{\Pi}\} - \mathit{Reject}.$$

2.3 Interpretation of LTL over Motion Trajectories in the Continuous State Space

The continuous state space $\mathcal{S}$ gives meaning to the propositions in the task specification. As an example, the proposition "robot is in area A_i" holds iff the robot is actually in A_i. The meaning of each proposition $\pi_i \in \Pi$ is defined by a function $\text{HOLDS}_{\pi_i} : \mathcal{S} \rightarrow \{\texttt{true}, \texttt{false}\}$, where $\text{HOLDS}_{\pi_i}(s) = \texttt{true}$ iff π_i holds at the continuous state $s \in \mathcal{S}$. This interpretation provides a mapping $\text{DISCRETESTATE} : \mathcal{S} \rightarrow 2^{\Pi}$ from the continuous state space $\mathcal{S}$ to the discrete space 2^{Π}, i.e.,

$$\text{DISCRETESTATE}(s) = \{\pi_i : \pi_i \in \Pi \wedge \text{HOLDS}_{\pi_i}(s) = \texttt{true}\}.$$

Moreover, trajectories over $\mathcal{S}$ give meaning to temporal connectives. A trajectory over $\mathcal{S}$ is a continuous function $\zeta : [0, T] \rightarrow \mathcal{S}$, parametrized by time. As the continuous state changes according to ζ, the discrete state, obtained by the mapping DISCRETESTATE, may also change. In this way, ζ gives rise to a sequence of discrete states, $\text{DISCRETESTATES}(\zeta) \stackrel{def}{=} [\tau_i]_{i=1}^{n}$, where $\tau_i \neq \tau_{i+1}$. As a result of this mapping, ζ is said to satisfy an LTL formula ϕ iff $\text{DISCRETESTATES}(\zeta) \models \phi$.

Note that the underlying motion dynamics of the robot are specified as a set of differential equations $f : \mathcal{S} \times \mathcal{U} \rightarrow \dot{\mathcal{S}}$, where $\mathcal{U}$ is a control space consisting of a finite set of input variables that can be applied to the robot (e.g., a car can be controlled by setting the acceleration and the rotational velocity of the steering wheel). The approach can take into account general nonlinear dynamics, relying only on a simulator, which, when given a state s, an input control u, and a time step dt, computes the new state that results from integrating the underlying motion dynamics.

A dynamically-feasible trajectory $\zeta : [0, T] \rightarrow \mathcal{S}$ starting at $s \in \mathcal{S}$ is obtained by computing a control function $\tilde{u} : [0, T] \rightarrow \mathcal{U}$ and propagating the dynamics forward in time through numerical integration of $\dot{\zeta}(h) = f(\zeta(h), \tilde{u}(h))$, i.e.,

$$\zeta(t) = s + \int_0^t f(\zeta(h), \tilde{u}(h))\, dh.$$

The dynamically-feasible trajectory $\zeta : [0, T] \rightarrow \mathcal{S}$ is considered collision free if each state along the trajectory avoids collisions with the obstacles.

2.4 Problem Statement

Given $\langle \mathcal{S}, \mathcal{U}, f, s_{init}, \Pi, \phi \rangle$ compute a control function $\tilde{u} : [0, T] \rightarrow \mathcal{U}$ such that the resulting dynamically-feasible trajectory $\zeta : [0, T] \rightarrow \mathcal{S}$ where

$$\zeta(t) = s_{init} + \int_0^t f(\zeta(h), \tilde{u}(h))\, dh$$

is collision free and satisfies the LTL specification ϕ, i.e., $\text{DISCRETESTATES}(\zeta) \models \phi$.

2.5 Examples

To facilitate presentation, this section provides examples of the robot model, workspaces, and the LTL specifications, which are used in the experiments.

Snake-Like Robot Model The snake-like robot model, as shown in Fig. 1, consists of several unit links attached to each other. The motion dynamics of the robot are modeled as a car pulling trailers (adapted from (LaValle, 2006, pp. 731)). The continuous state

$$s = (x, y, \theta_0, v, \psi, \theta_1, \theta_2, \ldots, \theta_N),$$

consists of the position $(x, y) \in \mathbb{R}^2$ ($|x| \leq 30, |y| \leq 25$), orientation $\theta_0 \in [-\pi, \pi)$, velocity v ($|v| \leq 2$), and steering-wheel angle ψ ($|\psi| \leq 1rad$) of the head link of the snake-like robot, and the orientation θ_i ($\theta_i \in [-\pi, \pi), 1 \leq i \leq N$) of each trailer link, where N is the number of trailers. The robot is controlled by setting the acceleration a ($|a| \leq 1$) and the rotational velocity ω ($|\omega| \leq 1rad/s$) of the steering-wheel angle. The differential equations of motions are

$$\dot{x} = v\cos(\theta_0) \quad \dot{y} = v\sin(\theta_0) \quad \dot{\theta_0} = v\tan(\psi)$$
$$\dot{v} = a \quad \dot{\psi} = \omega$$
$$\dot{\theta_i} = \tfrac{v}{d}\left(\prod_{j=1}^{i-1}\cos(\theta_{j-1} - \theta_j)\right)(\sin(\theta_{i-1}) - \sin(\theta))$$

where $1 \leq i \leq N$ and $d = 0.05$ is the hitch length. A continuous state s is considered valid iff the robot does not collide with the obstacles and the robot does not self intersect, i.e., non-consecutive links do not collide with each other.

While related work in LTL motion planning (Bhatia et al., 2011; Ding et al., 2011; Fainekos et al., 2009; Kress-Gazit et al., 2007) has generally focused on low-dimensional robotic systems, by increasing the number of trailer links, the snake-like robot provides challenging test cases for high-dimensional motion-planning problems with dynamics, as noted in (Laumond, 1993). In the experiments in this paper, the number of trailer links is varied as $0, 5, 10, 15$ yielding problems with $5, 10, 15, 20$ DOFs.

Workspaces The workspaces in which the robot operates, as in the related LTL motion-planning work (Bhatia et al., 2011; Ding et al., 2011; Fainekos et al., 2009; Kress-Gazit et al., 2007; Plaku, Kavraki, and Vardi, 2009, 2012), are populated with polygonal obstacles and propositions. In this way, a proposition π_i is associated with a polygon p_i, and the function $\text{HOLDS}_{\pi_i}(s)$ is true iff the position component of s is inside p_i. In addition, as in the related LTL motion-planning work, each workspace is triangulated (using the Triangle package (Shewchuk, 2002)). The triangulation conforms to the vertices of the polygons associated with the obstacles and the propositions, i.e., triangulation treats the polygonal obstacles and propositions as holes. The polygonal propositions $p_1, \ldots, p_k$ and the triangles $t_1, \ldots, t_m$ give rise to an adjacency region graph $G = (R, E)$, where

$R = \{p_1, \ldots, p_k, t_1, \ldots, t_m\}$ and

$E = \{(r_i, r_j) : r_i, r_j \in R \text{ are physically adjacent}\}$.

Each region $r \in R$ is labeled by the corresponding discrete state, i.e.,

$$\text{DISCRETESTATE}(r) = \begin{cases} \{\pi_\ell\}, & \text{if } r = p_\ell \text{ for some } 1 \leq \ell \leq k, \\ \emptyset, & \text{otherwise} \end{cases}$$

An illustration of the workspaces, propositions, and triangulations is provided in Fig. 1.

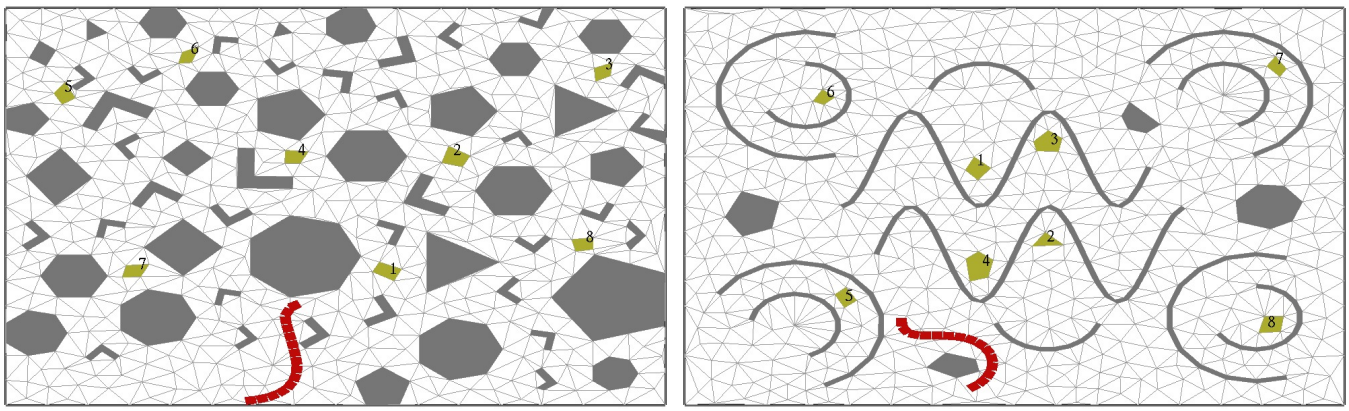

Figure 1: Workspaces used in the experiments. Obstacles are in gray. Propositions are in a golden color and are numbered $1, 2, \ldots, 8$. The robot is in red and is shown in some initial state. The workspace triangulations are also shown. These workspaces provide challenging environments, as the snake-like robot has to wiggle its way through various narrow passages.

LTL Tasks LTL tasks used in the experiments are defined over 8 propositions, as shown in Fig. 1.

The first task is to compute a collision-free and dynamically-feasible trajectory ζ which satisfies the propositions $\pi_1, \pi_2, \ldots, \pi_8$ in succession, i.e.,

$$\phi_1 = \beta \cup (\pi_1 \wedge ((\pi_1 \vee \beta) \cup (\pi_2 \wedge (\ldots (\pi_7 \vee \beta) \cup \pi_8)))),$$

where $\beta = \wedge_{i=1}^{8} \neg \pi_i$.

The second task is to compute a collision-free and dynamically-feasible trajectory which eventually satisfies $\pi_i, \pi_j, \pi_k, \pi_i, \pi_j$ with different i, j, k, i.e.,

$$\phi_2 = \bigvee_{1 \leq i,j,k \leq 8, i \neq j, j \neq k, i \neq k} \Diamond \pi_i \wedge (\Diamond \pi_j \wedge (\Diamond \pi_k \wedge (\Diamond \pi_i \wedge (\Diamond \pi_j)))).$$

This task presents polynomially different possibilities about which propositions to satisfy.

The third task is to compute a collision-free and dynamically-feasible trajectory ζ that eventually satisfies each proposition, i.e.,

$$\phi_3 = \bigwedge_{i=1}^{8} \Diamond \pi_i.$$

This task presents combinatorially many different possibilities regarding the order in which to satisfy the propositions.

3 Method

Let $s_{init} \in \mathcal{S}$ denote the initial state of the robot. Let ϕ denote the LTL specification. In order to compute a collision-free and dynamically-feasible trajectory ζ that satisfies ϕ, i.e., $\textsc{DiscreteStates}(\zeta) \models \phi$, as mentioned earlier, the approach couples sampling-based motion planning in the continuous state space $\mathcal{S}$ with discrete search over both the automaton $\mathcal{A}$ representing LTL and the region graph $G = (R, E)$. To facilitate presentation, the general idea is described first followed by details of the approach.

3.1 Coupling Sampling-based Motion Planning and Discrete Search

Sampling-based motion planning uses a tree data structure $\mathcal{T}$ as the basis for conducting the search in the continuous space $\mathcal{S}$. Each vertex $v_i \in \mathcal{T}$ is associated with a collision-free continuous state, denoted as $\mathit{state}(v_i)$. The vertex v_i also keeps track of its parent in $\mathcal{T}$, denoted as $\mathit{parent}(v_i)$, and the collision-free and dynamically-feasible trajectory that connects its parent state to $\mathit{state}(v_i)$, denoted as $\mathit{ptraj}(v_i)$. Initially, $\mathcal{T}$ contains only its root vertex v_1, where $\mathit{state}(v_1)$ corresponds to the initial continuous state s_{init}. As the search progresses, $\mathcal{T}$ is extended by adding new vertices v_j and new collision-free and dynamically-feasible trajectories $\mathit{ptraj}(v_j)$ as branches of $\mathcal{T}$. As described in Section 3.3, these collision-free and dynamically-feasible trajectories are obtained by sampling controls and propagating forward for several steps the motion dynamics of the robot, and stopping the propagation if a collision is found.

Let $\mathit{traj}(\mathcal{T}, v_j)$ denote the trajectory from $\mathit{state}(v_1)$ to $\mathit{state}(v_j)$, which is obtained by concatenating the collision-free and dynamically-feasible trajectories associated with the tree edges from v_1 to v_j. Then, $\mathit{traj}(\mathcal{T}, v_j)$ satisfies the LTL formula ϕ iff $\textsc{DiscreteStates}(\mathit{traj}(\mathcal{T}, v_j)) \models \phi$. Moreover, recall that $\textsc{DiscreteStates}(\mathit{traj}(\mathcal{T}, v_j)) \models \phi$ iff the sequence of discrete states ends up on an accepting state when run on the automaton $\mathcal{A}$ representing ϕ. In order to make this computation efficient, each vertex v_j is associated with the automaton state, denoted as $z(v_j)$, obtained by running $\textsc{DiscreteStates}(\mathit{traj}(\mathcal{T}, v_j))$ on $\mathcal{A}$. The computation of $z(v_j)$ is done incrementally when checking $\mathit{ptraj}(v_j)$ for collisions, as described in Section 3.3. As it will be explained in the next paragraph, v_j is also associated with $\mathit{region}(v_j)$, which denotes the workspace region that contains the position component of $\mathit{state}(v_j)$.

The objective of the discrete layer is to guide sampling-based motion planning as it expands $\mathcal{T}$ in the continuous space $\mathcal{S}$. To do so effectively, the discrete layer selects at each iteration an automaton state z_{from} and a workspace region r_{from} from which to expand the search and an automa-

Algorithm 1 LTLMOTIONPLANNING($\Pi, \phi, G, s_{init}, f, t_{max}$)

Input
Π: propositions
ϕ: LTL formula
$G = (R, E)$: workspace region graph
s_{init}: initial state of the robot
$f : \mathcal{S} \times \mathcal{U} \rightarrow \dot{\mathcal{S}}$: differential equations of motion
t_{max}: upper bound on time
Output: If successful, method computes a collision-free and dynamically-feasible trajectory ζ that satisfies ϕ

$\mathcal{A} \leftarrow$ AUTOMATON(ϕ); $\mathcal{T} \leftarrow$ CREATETREE(s_{init})
while TIME() $< t_{max}$ **and** SOLVED() = `false` **do**
  $\diamondsuit$ *discrete layer*
  $\langle z_{from}, r_{from}, z_{to} \rangle \leftarrow$ DISCRETETARGET($\mathcal{T}, \mathcal{A}, G$)
  $\sigma \leftarrow$ DISCRETEPLAN($\mathcal{A}, G, z_{from}, r_{from}, z_{to}$)
  $\sigma_{active} \leftarrow \emptyset$
  for $i = |\sigma|$ down to 1 **do**
    $\langle z, r \rangle \leftarrow \sigma(i)$
    if $|vertices(\mathcal{T}, z, r)| > 0$ **then**
      $\sigma_{active}.pushback(\langle z, r \rangle)$
  $\diamondsuit$ *continuous layer*
  for several times **do**
    $\langle z, r \rangle \leftarrow$ SELECTATRANDOM(σ_{active})
    $v \leftarrow$ SELECTATRANDOM($vertices(\mathcal{T}, z, r)$)
    $u \leftarrow$ SAMPLECONTROLINPUT()
    for several times **do** $\diamondsuit$ *extend* $\mathcal{T}$ *from* v
      $dt \leftarrow$ GETINTEGRATIONSTEP($state(v), u$)
      $s_{new} \leftarrow$ INTEGRATEMOTIONEQS($f, state(v), u, dt$)
      $r_{new} \leftarrow$ LOCATEREGION(s_{new})
      $z_{new} \leftarrow \delta(z(v),$ DISCRETESTATE(s_{new}))
      if COLLISION(s_{new}) = `true` **or** (z_{new}) $\in$ *Reject* **then**
        break for loop of extend $\mathcal{T}$
      $v_{new} \leftarrow$ ADDNEWVERTEX($\mathcal{T}, s_{new}, u, dt, r_{new}, z_{new}, v$)
      if $z_{new} \in$ *Accept* **then return** $traj(\mathcal{T}, v_{new})$
      if $\langle z_{new}, r_{new} \rangle \notin \sigma_{active}$ **then**
        $\sigma_{active}.pushback(\langle z_{new}, r_{new} \rangle)$
      $vertices(\mathcal{T}, z_{new}).pushback(v_{new})$
      $vertices(\mathcal{T}, z_{new}, r_{new}).pushback(v_{new})$
      $v \leftarrow v_{new}$

ton state z_{to} toward which to expand the search. The automaton state z_{from} and the workspace region r_{from} are selected from those already associated with the vertices in $\mathcal{T}$. The automaton state z_{to} is selected from those connected by a single automaton transition from z_{from}, i.e., $z_{to} \in \delta(z_{from})$. An additional criterion is that z_{to} should not be associated with the tree vertices, i.e., $\forall v_j \in \mathcal{T} : z(v_j) \neq z_{to}$, so that the search can be expanded toward new automaton states.

The discrete and the continuous layers work in tandem to effectively compute a collision-free and dynamically feasible trajectory that satisfies the LTL task specification ϕ. The search proceeds incrementally until a solution is obtained or an upper bound on computational time is exceeded. Each iteration consists of invoking the discrete layer to select $z_{from}, r_{from}, z_{to}$ and then compute a discrete plan, i.e., a sequence of automaton states and workspace regions that connects $\langle z_{from}, r_{from} \rangle$ to z_{to}. Sampling-based motion planning is invoked next which aims to expand $\mathcal{T}$ from vertices associated with $\langle z_{from}, r_{from} \rangle$ toward z_{to} while using the discrete plan as a guide. As the motion planner expands $\mathcal{T}$, new automaton states and workspace regions could be reached by the vertices and trajectories added to $\mathcal{T}$. As a result, the discrete layer could suggest to the continuous layer a different discrete plan for expanding $\mathcal{T}$ in the next iteration. This coupling of discrete planning in the discrete layer and sampling-based motion planning in the continuous layer, as evidenced by experimental results, allows the approach to efficiently compute a collision-free and dynamically-feasible motion trajectory that satisfies the LTL task specification. Pseudocode is given in Algo 1. More detailed descriptions of the main steps of the approach follow.

3.2 Discrete Layer

DISCRETETARGET($\mathcal{T}, \mathcal{A}, G$) selects an automaton state z_{from} and a region r_{from} from which to expand the search and an automaton state z_{to} toward which to expand the search (Algo 1:3). To facilitate the selection, tree vertices are grouped according to their associated automaton states, i.e.,

$$vertices(\mathcal{T}, z) = \{v : v \in \mathcal{T} \wedge z(v) = z\}.$$

These vertices are further grouped according to their associated regions, i.e.,

$$vertices(\mathcal{T}, z, r) = \{v : v \in \mathcal{T} \wedge z(v) = z \wedge region(v) = r\}.$$

Let Γ denote the set of all automaton states that are already associated with the vertices in $\mathcal{T}$, i.e.,

$$\Gamma = \{z : z \in Z \wedge |vertices(\mathcal{T}, z)| > 0\}.$$

The automaton state z_{from} is selected from Γ according to the probability distribution

$$prob(z) = 2^{1/d(z)} / \sum_{z' \in \Gamma} 2^{1/d(z')},$$

where $d(z)$ denotes the minimum number of automaton transitions to reach an accepting state in $\mathcal{A}$ from z. Since the overall objective is to effectively compute a trajectory that satisfies the LTL specification, the selection of z_{from} is biased toward automaton states that are close to accepting states.

The region r_{from} is selected uniformly at random from those regions associated with $vertices(\mathcal{T}, z_{from})$, i.e.,

$$\{r : r \in R \wedge |vertices(\mathcal{T}, z_{from}, r)| > 0\}.$$

The random selection is commonly advocated in motion-planning literature as it allows the sampling-based motion planner to expand $\mathcal{T}$ from different regions.

To expand the search toward new automaton states, z_{to} is selected among those automaton states not yet associated with the tree vertices, i.e., $z_{to} \notin \Gamma$. To ensure that the discrete plans are not too long, another criterion is that z_{to} should be a non-rejecting automaton state connected by a single transition from z_{from}, i.e., $z_{to} \in \delta(z_{from})$. Taking these criteria into account, z_{to} is selected uniformly at random from the set

$$\delta(z_{from}) - \Gamma.$$

In this way, the selection of $z_{from}, r_{from}, z_{to}$ aims to expand the search from tree vertices associated with automaton states that are close to accepting states and toward new automaton states.

DISCRETEPLAN$(\mathcal{A}, G, z_{from}, r_{from}, z_{to})$ computes a sequence of automaton states and regions that connects $\langle z_{from}, r_{from} \rangle$ to z_{to} (Algo 1:4). The discrete search is conducted over an abstract graph, which is obtained by implicitly combining the automaton $\mathcal{A}$ and the region graph $G = (R, E)$. More specifically, the edges coming out of a vertex $\langle z, r \rangle$ in the abstract graph are computed as

$$\text{EDGES}(\langle z, r \rangle) = \{\langle \delta(z, \text{DISCRETESTATE}(r')), r' \rangle : (r, r') \in E\}.$$

The cost of an edge $(\langle z', r' \rangle, \langle z'', r'' \rangle)$ is defined as the distance from region r' to r''. A vertex $\langle z, r \rangle$ in the abstract graph is considered as a goal vertex iff $z = z_{to}$.

The discrete plan σ from $\langle z_{from}, r_{from} \rangle$ to z_{to} is computed as the shortest path with probability p and as a random path with probability $1 - p$. While shortest paths provide greedy guides to the sampling-based motion planner, random paths allow for exploration of new regions, which provide alternative routes to prevent the sampling-based motion planner from getting stuck. A randomized version of depth-first-search, which visits the out-going vertices in a random order is used for the computation of random paths. Shortest paths are computed using Dijkstra's algorithm.

Note that sampling-based motion planning can expand $\mathcal{T}$ only from those $\langle z, r \rangle \in \sigma$ for which $|\textit{vertices}(\mathcal{T}, z, r)| > 0$. For this reason, σ is scanned backwards and $\langle z, r \rangle$ is added to $\sigma_{\textit{active}}$ if $|\textit{vertices}(\mathcal{T}, z, r)| > 0$. (Algo 1:6-9).

3.3 Continuous Layer

The objective of sampling-based motion planning is to expand $\mathcal{T}$ by adding several collision-free and dynamically-feasible trajectories using $\sigma_{\textit{active}}$ as a guide. Each trajectory is generated by first selecting $\langle z, r \rangle$ from $\sigma_{\textit{active}}$ uniformly at random (Algo 1:11). A vertex v is then selected uniformly at random from $\textit{vertices}(\mathcal{T}, z, r)$ (Algo 1:12) and a control input u is sampled uniformly at random (Algo 1:13). Note that other selection and sampling strategies are possible, as discussed in motion-planning books (Choset et al., 2005; LaValle, 2006). Random selections and sampling are commonly used in motion planning and are also shown to work well for the problems studied in this work.

A trajectory ζ is then obtained by integrating for several steps the motion dynamics of the robot when applying the control input u to $\textit{state}(v)$. To ensure accuracy, as advocated in the literature, this paper uses Runge-Kutta methods with an adaptive integration step (Algo 1:15-16).

Intermediate states s_{new} along ζ are added as new vertices to $\mathcal{T}$. Each new vertex v_{new} is associated with the corresponding region r_{new} and automaton state z_{new} (Algo 1:17-18). Recall that r_{new} is computed as the region that contains the position component of s_{new}. The automaton state z_{new} is computed as $\delta(z(v), \text{DISCRETESTATE}(s_{new}))$, where v is the parent of v_{new}. The integration of ζ stops if s_{new} is in collision or z_{new} is a rejecting automaton state (Algo 1:19-20). Otherwise, if z_{new} is an accepting automaton state, then $\textit{traj}(\mathcal{T}, v_{new})$ constitutes a collision-free and dynamically-feasible trajectory that satisfies the LTL task specification. In such case, the search terminates successfully.

The sampling-based motion planner may augment $\sigma_{\textit{active}}$. In fact, $\langle z_{new}, r_{new} \rangle$ is added to $\sigma_{\textit{active}}$ if not already there (Algo 1:23-24). Such additions enable the sampling-based motion planner to expand the search toward new automaton states and new regions.

4 Experiments and Results

Experimental validation is provided by using several scenes, LTL specifications, and a snake-like robot model with numerous DOFs, as described in Section 2.5. By increasing the number of links, the robot model provides challenging test cases for high-dimensional problems. In the experiments in this paper, the number of links is varied as $0, 5, 10, 15$ yielding problems with $5, 10, 15, 20$ DOFs. The running time for each problem instance is obtained as the average of thirty different runs. Experiments are run on an Intel Core 2 Duo machine (CPU: P8600 at 2.40GHz, RAM: 8GB) using Ubuntu 11.10. Code is compiled with GNU g++-4.6.1

Fig. 2 provides a summary of the results. Comparisons to related work (Bhatia et al., 2011; Plaku, Kavraki, and Vardi, 2009) show significant computational speedups. The speedups are more pronounced in the case of tasks 2 and 3. Recall that task 1 presents only one possible order in which to satisfy the propositions, while tasks 2 and 3 represent polynomially and exponentially many different possibilities. As a result, while the automaton for task 1 has linear size, the automata for tasks 2 and 3 have polynomial and exponential size, respectively. Since related work guides motion planning by using complete discrete plans, i.e., from $\langle z(v_{\textit{init}}), r(v_{\textit{init}}) \rangle$ to some $z \in \textit{Accept}$, as the size of the automaton increases, so does the length of the discrete plan and the computational cost to obtain such discrete plans. In distinction, the proposed approach uses short discrete plans from $\langle z_{from}, r_{from} \rangle$ to z_{to}, where z_{to} is connected by only one automaton transition from z_{from}. Moreover, while all the discrete plans in related work start from $\langle z(v_{\textit{init}}), r(v_{\textit{init}}) \rangle$, the proposed approach is biased to start discrete plans from automaton states that are close to accepting states.

As expected, the running time increases with the number of DOFs. As more and more links are added to the robot, it becomes increasingly difficult for the robot to move along the narrow passages of the workspaces and satisfy the LTL specifications. Nevertheless, the proposed approach is able to efficiently solve all problem instances, even as the number of DOFs is increased to 20, while related work struggles to find solutions.

In all the above experiments, the number of expansion iterations per discrete target is set to 200 (Algo. 1:10). Fig. 3 summarizes the results of the proposed approach when varying the number of expansion iterations. Significantly large numbers of iterations result in wasted computational time particularly when it is difficult to expand $\mathcal{T}$ to reach the discrete target. A small number of iterations may not give the sampling-based motion planner enough time to make progress in expanding $\mathcal{T}$ to reach the discrete target. As the results indicate, however, the approach works well for a wide

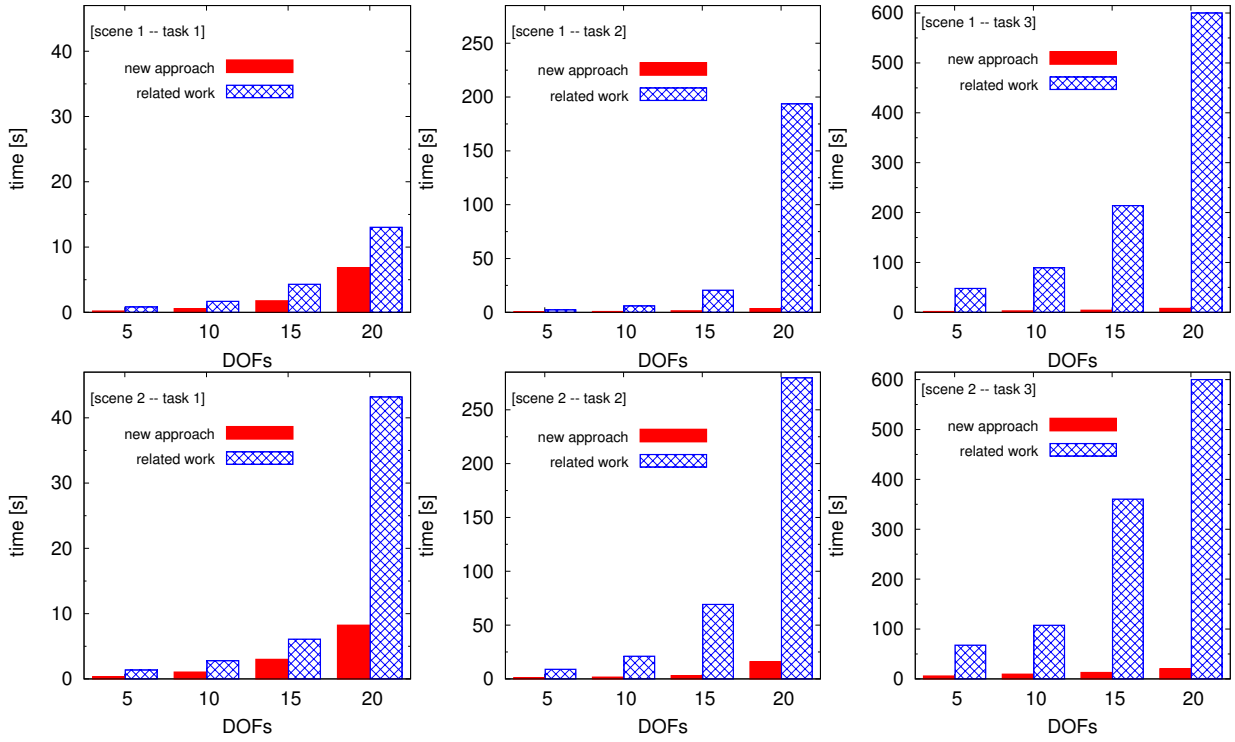

Figure 2: Comparison of the proposed LTL motion-planning approach to related work (Bhatia et al., 2011; Plaku, Kavraki, and Vardi, 2009). Results are shown over the two scenes (Fig. 1) and the three LTL specifications (Section 2.5) when varying the number of DOFs of the snake-like robot model from 5 to 20.

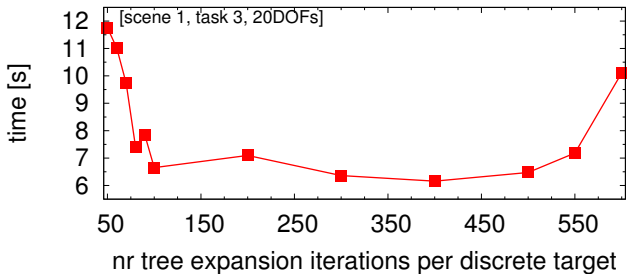

Figure 3: Impact of the number of expansion iterations per discrete target (Algo. 1:10) on the computational efficiency of the approach.

range of values. Similar trends to Fig. 3 which shows the result for scene 1, task 3, and the robot model with 20 DOFs, were observed for the other scenes, tasks, and robot DOFs, but are not shown here due to space constraints.

As in related work, the focus of this paper was on improving the computational time to solve LTL motion-planning problems. The solution trajectories, even though not optimized, are generally short. The table below summarizes the average path length computed as the distance traveled by the reference point of the head link of the snake-like robot. These results are presented as ratios over the lower bound on the solution obtained by the shortest-collision free path for a point robot with no differential constraints.

[scene 1, task 1]

DOFs	5	10	15	20
length ratio	1.11	1.22	1.31	1.34

As the results indicate, the solutions produced by the proposed approach come close to the lower bound. Similar results were obtained for the other scenes and tasks but are not shown here due to space constraints.

5 Discussion

Toward the goal of bridging the gap between the discrete and the continuous, this paper showed how to effectively compute collision-free and dynamically-feasible motion trajectories that satisfy task specifications given by LTL. LTL makes it possible to express sophisticated tasks in terms of propositions, logical connectives, and temporal connectives. The approach combined sampling-based motion planning over the continuous state space with discrete search over both the LTL task representation and a workspace region graph. The discrete search guides the sampling-based motion planner to expand the search from tree vertices associated with automaton states that are close to accepting states and toward new automaton states. In distinction from related work, the proposed approach samples the discrete space to shorten the length of the discrete plans and to more effectively guide motion planning in the continuous state space. Experimental results show significant computational speedups over related work.

Even though the focus of this paper, as in related work,

was on improving the computational time to solve LTL motion-planning problems, the proposed approach generally produces short trajectories. The quality of the trajectories can be further improved by using common postprocessing techniques (Choset et al., 2005; LaValle, 2006). In addition, A* criteria could be incorporated into the approach to select automaton states and workspace regions that promote the generation of shorter trajectories. Recent results in sampling-based motion planning (Karaman and Frazzoli, 2011), which show optimality in the case of point-to-point planning, could provide a basis on how to obtain optimality even when considering LTL motion-planning tasks.

References

Arkin, R. C. 1990. Integrating behavioral, perceptual and world knowledge in reactive navigation. *Robotics and Autonomous Systems* 6:105–122.

Armoni, R.; Egorov, S.; Fraer, R.; Korchemny, D.; and Vardi, M. 2005. Efficient LTL compilation for SAT-based model checking. In *Intl Conf on Computer-Aided Design*, 877–884.

Bhatia, A.; Maly, M.; Kavraki, L.; and Vardi, M. 2011. Motion planning with complex goals. *IEEE Robotics Automation Magazine* 18:55–64.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* (1):104–126.

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

Ding, X. C.; Kloetzer, M.; Chen, Y.; and Belta, C. 2011. Formal methods for automatic deployment of robotic teams. *IEEE Robotics and Automation Magazine* 18(3):75–86.

Fainekos, G. E.; Girard, A.; Kress-Gazit, H.; and Pappas, G. J. 2009. Temporal logic motion planning for dynamic mobile robots. *Automatica* 45(2):343–352.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Morgan Kaufmann.

Hauser, K., and Ng-Thow-Hing, V. 2011. Randomized multi-modal motion planning for a humanoid robot manipulation task. *International Journal of Robotics Research* 30(6):678–698.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7):846–894.

Kress-Gazit, H.; Conner, D. C.; Choset, H.; Rizzi, A.; and Pappas, G. J. 2007. Courteous cars: Decentralized multi-agent traffic coordination. *Special Issue of the IEEE Robotics and Automation Magazine on Multi-Agent Robotics*.

Kress-Gazit, H.; Wongpiromsarn, T.; ; and Topcu, U. 2011. Correct, reactive robot control from abstraction and temporal logic specifications. *IEEE Robotics and Automation Magazine on Formal Methods for Robotics and Automation* 18(3):65–74.

Kupferman, O., and Vardi, M. 2001. Model checking of safety properties. *Formal methods in System Design* 19(3):291–314.

Laumond, J. 1993. Controllability of a multibody mobile robot. *IEEE Transactions on Robotics and Automation* 9(6):755–763.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge, MA: Cambridge University Press.

Nieuwenhuisen, D.; van der Stappen, A.; and Overmars, M. 2006. An effective framework for path planning amidst movable obstacles. In *International Workshop on Algorithmic Foundations of Robotics*, volume 47 of *Springer Tracts in Advanced Robotics*. 87–102.

Payton, D. W.; Rosenblatt, J. K.; and Keirsey, D. M. 1990. Plan guided reaction. *IEEE Trans. on Systems, Man, and Cybernetics* 20(6):1370–1372.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *IEEE International Conference on Robotics and Automation*, 5002–5008.

Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2009. 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 *Lecture Notes in Computer Science*. York, UK: Springer. 368–382.

Plaku, E.; Kavraki, L. E.; and Vardi, M. Y. 2012. Falsification of LTL safety properties in hybrid systems. *International Journal on Software Tools and Technology Transfer*. invited, in print.

Saffiotti, A.; Konolige, K.; and Ruspini, E. H. 1995. A multivalued logic approach to integrating planning and control. *Artificial Intelligence* 76(1-2):481–526.

Shewchuk, J. R. 2002. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications* 22(1-3):21–74.

Sistla, A. 1994. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing* 6:495–511.

Stilman, M., and Kuffner, J. 2008. Planning among movable obstacles with artificial constraints. *International Journal of Robotics Research* 27(12):1295–1307.

Stilman, M. 2010. Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics* 26(3):576–584.

Wolfe, J.; Marthi, B.; and Russell, S. 2010. Combined task and motion planning for mobile manipulation. In *Intl. Conf. on Automated Planning and Scheduling*, 254–258.