Fast Motion Planning in Dynamic Environments With Extended Predicate-Based Temporal Logic

Ziyang Chen, Mingyu Cai, Zhangli Zhou, *Graduate Student Member, IEEE*, Lin Li, *Graduate Student Member, IEEE*, and Zhen Kan, *Senior Member, IEEE*

***Abstract*— Formal languages effectively outline robots' task specifications, yet current temporal logic struggles to balance semantic expression with solution speed. To address this challenge, we propose extended predicate-based temporal logic (E-pTL), augmenting conventional linear temporal logic with more expressive atomic predicates to reflect the time, space, and order attributes of task and represent complex tasks with dynamic propositions, time windows, and even relative time window. This approach, blending automata-based task abstraction and extended predicates, offers enhanced expressiveness and conciseness for intricate specifications. To cope with E-pTL, we introduce a novel planning framework, the Planning Decision Tree (PDT). PDT incrementally builds a tree through automata and system state searches, recording potential task plans. The proposed pruning method can reduce the exploration space. This method swiftly handles complex temporal tasks defined by E-pTL. Rigorous analysis confirms PDT-based planning's feasibility (ensuring satisfactory planning aligned with task specifications) and completeness (guaranteeing a feasible solution if available). Moreover, PDT-based planning proves efficient, with solution times approximately linearly proportional to automaton states squared. Extensive simulations and experiments validate its effectiveness and efficiency.**

***Note to Practitioners*—Temporal logic, as a formal language, has been widely applied to describe the task of robotic systems. However, linear temporal logic (LTL) struggles with the explicit time constraints, while other temporal logics, e.g., signal temporal logic (STL), cannot compactly represent task progress via an automaton. Consequently, current solutions for complex task with temporal logic specification heavily lean on optimization-based approaches, which are time-consuming and impractical for real-time systems. This work presents a novel approach for fast motion planning in dynamic environments with extended predicate-based temporal logic (E-pTL). The proposed E-pTL not only enhances semantic expression but also can be checked by an automaton, simplifying progress checks. However, existing automata-based methods encounter limitations in dynamic tasks. Graph search methods require replanning at each time step and lack completeness guarantees, while sampling methods can't efficiently handle the feasibility of each atomic proposition. In contrast, the developed planning decision tree (PDT)-based framework incrementally searches for automata and validates propositions without a transition system, significantly reducing the search space. The suggested pruning method further trims the search space without compromising feasibility. Overall, PDT enables rapid, reliable planning, outperforming optimization-based methods like ILP/MILP and proving well-suited for real-time applications.**

***Index Terms*— Formal methods for robotics and automation, extended predicate-based temporal logic, planning decision tree.**

Manuscript received 10 April 2024; accepted 10 June 2024. Date of publication 26 June 2024; date of current version 28 February 2025. This article was recommended for publication by Associate Editor L. Cheng and Editor Z. Li upon evaluation of the reviewers' comments. This work was supported in part by the National Natural Science Foundation of China under Grant U2013601 and Grant 62173314. *(Corresponding author: Zhen Kan.)*

Ziyang Chen, Zhangli Zhou, Lin Li, and Zhen Kan are with the Department of Automation, University of Science and Technology of China, Hefei, Anhui 230026, China (e-mail: zkan@ustc.edu.cn).

Mingyu Cai is with the Department of Mechanical Engineering, University of California at Riverside, Riverside, CA 92521 USA.

Digital Object Identifier 10.1109/TASE.2024.3418409

I. Introduction

LINEAR temporal logic (LTL) is a formal language capable of expressing rich task specifications. Recently, motion planning with LTL specifications has generated substantial interest [1], [2]. For instance, for the collaborative human-robot [3], LTL was used to specify manipulation tasks [4]. In [5], LTL was applied for long-range navigation in outdoor environments. In [6], [7], and [8], task allocation and planning for temporal logic goals were developed for heterogeneous multi-robot systems. In [9], deep reinforcement learning was investigated for continuous motion planning of autonomous systems with temporal logic specifications. Other representative results on motion planning with LTL specifications include distributed motion coordination [10], reactive planning in unknown environments [11], [12], [13], and optimal probabilistic motion planning [14]. Despite these successful applications, LTL generally cannot express explicit time constraints. For example, a surveillance task, e.g., infinitely visiting regions A and B, can be expressed in LTL, but tasks with explicit time constraints, e.g., visiting area A immediately followed by servicing B within 10-time units or two consecutive visits of the area A need to be at least 5 time units apart, cannot be described by LTL formulas.

A. Related Works

To deal with explicit time constraints, signal temporal logic (STL) was originally introduced in [15] to reason about continuous-time real signals, which can be viewed as a variant of metric temporal logic (MTL) [16], [17] specifically tailored for real signals. STL has been widely applied to describe missions with time constraints, such as trajectory planning [18], trajectory prediction [19], human behavior prediction [20], traffic systems [21], autonomous vehicle [22], [23], multi-robot systems [24], [25] etc. Besides STL, bounded temporal logics, such as truncated linear temporal logic (TLTL) [26], metric interval temporal logic (MITL) [27], time window

temporal logic (TWTL) [28], [29], can also capture the time constraints over the tasks. Compared with MITL, TLTL, and STL, the time-bounded TWTL specifications can be represented more compactly and comprehensively, since TWTL uses exact time bounds as opposed to the STL where the time bounds need to be shifted. TLTL is a specification language with an extended set of operators defined over finite-time trajectories of a robot's states. Similar to STL, TLTL is defined over predicate and has quantitative semantics (also referred to as robustness degree) to measure the satisfaction of tasks. Different from STL, TLTL does not require a time bound to be specified on each temporal operator. In [30], TLTL was successfully integrated with safe reinforcement learning to perform robotic cooking tasks.

These aforementioned temporal logics are generally defined over fixed operators and propositions, and cannot take into account a variety of time-varying factors that might affect the feasibility of motion planning. Thus, they are limited in semantics expressivity, especially in a dynamic environment with time-varying propositions. The work of [31] considers the probabilistic satisfiability of LTL tasks in uncertain situations. The work of [32] considers bounded time and time-dependent obstacles. In [33], the moving areas of interest and mobile uncertain semantic targets are considered. However, complex dynamic constraints require a long solution time. Therefore, although aforementioned temporal logics can model the time-varying propositions by labeling function, it is hard to be applied for online planning. Besides, complex tasks with the proposition related to other propositions cannot be expressed. For instance, if the executed time window of one proposition is related to the executing time of another proposition, none of these aforementioned temporal logics can fully express the task specifications.

Besides augmenting the semantics of temporal logic, generating a timely motion planning that can quickly respond to environmental changes is also highly desired. Conventional motion planning with LTL specifications, as well as its variants such as TWTL, MITL, etc., mainly rely on optimization-based approaches [6] or graph search techniques [34], [35] applied to the product automaton. Since the product automaton is built upon the transition system, its size can grow exponentially with the complexity of the environment. Thus, such model-checking methods suffer from the curse of dimensionality and generally require a long solution time, limiting their applications in a dynamic environment where real-time planning is necessary. Although the product automaton is not required for STL specifications, robustness evaluation criteria are generally established to transform the formula-oriented planning to an integer linear program or mixed integer linear program (ILP/MILP) problems [36], [37]. Unfortunately, ILP/MILP is in general computationally expensive and thus cannot also scale well and provide timely planning. By building a tree to approximate the product automaton, sampling-based approaches were developed to jointly consider long-term temporal logic goals with short-term reactive requirements [38], [39]. An abstraction-free approach was also developed in [6] for the optimal control synthesis of multi-robot systems. Although sampling-based approaches scale well, they suffer from strong randomness and poor stability. In addition, due to the existence of a large number of useless samples, long exploration time is often needed. The work of [40] uses special combinations of formulas to relate atomic propositions in LTL to predicates in STL, enabling solving STL problems via control barrier functions (CBFs). However, it can not formulate formulas over long-horizons and manually constructing valid CBFs for each STL satisfaction is also challenging.

B. Contribution

Motivated by the above practical challenges, we expanded LTL to encompass broader predicates termed as extended predicate-based temporal logic (E-pTL), incorporating space, time, and order constraints. Specifically, E-pTL can efficiently represent complex tasks with dynamic propositions, time windows, and even relative time window, which enhances the semantic expressiveness of conventional LTL. Nevertheless, the current motion planning algorithms designed for LTL lack applicability to this more expressive formal language. Specially, graph search methods cannot directly handle dynamic information and should replan in each time step without the guarantee of completeness. The rewiring in sampling methods can affect the feasibility of searched states. Optimization-based methods with long solution time are not suitable for real-time system. Hence, our primary contribution lies in devising a fast motion planning algorithm for tasks with complex constraints modeled by extended predicate-based temporal logic.

Specifically, to cope with E-pTL, a novel framework based the automaton is developed, called the planning decision tree (PDT). PDT implements task planning through sub-task search and system state iteration without the transition system. Therefore, compared with conventional model checking methods, PDT can track the task progress without requiring sophisticated product automaton and directly verifies complex constraints such as time windows and dynamic interest areas. By growing the tree from the root node to the leaf nodes, PDT can search for motion planning that satisfies the E-pTL task and the robotic system. Such a design enables fast motion planning compared with existing method for complex semantic formulas, especially in a dynamic environment where prompt reactive planning is highly desired.

Rigorous analysis shows that the PDT-based planning is feasible (i.e., the generated planning is applicable and satisfactory with respect to the E-pTL task) and complete (i.e., a feasible solution, if exists, is guaranteed to be found). we demonstrate that PDT-based planning proves more efficient than ILP/MILP baselines, with the time taken to find satisfactory planning being linearly proportional to the square of the automaton states. Extensive simulations and experiments substantiate its effectiveness and efficiency.

C. Organization

Sec. II reviews the definitions of temporal logic and introduces system models. Sec. III presents the developed E-pTL and the semantic analysis. Sec. IV describes the PDT-based motion planning framework with E-pTL specifications. The feasibility, completeness, and complexity of the motion planning algorithm are then analyzed in Sec. V. Simulation and

experiment results are presented in Sec. VI, followed by the discussions and conclusions in Sec. VII.

II. Preliminaries and Problem Formulation

A. Linear Temporal Logic

As a formal language, LTL is defined over a set of atomic propositions AP with Boolean and temporal operators. The syntax of LTL is defined as:

$$\phi := true|ap|\phi_1 \wedge \phi_2|\neg\phi_1|X\phi|\phi_1 U \phi_2$$

where $ap \in AP$ is an atomic proposition, *true*, $\neg$ (negation), and $\wedge$ (conjunction) are propositional logic operators, and X (next) and U (until) are temporal operators. Other propositional logic operators such as *false*, $\vee$ (disjunction), $\rightarrow$ (implication), and temporal operators such as G (always) and F (eventually) can also be defined [41].

Let 2^{AP} denote the power set of AP and a word o is defined as an infinite sequence $o = o_0 o_1 \ldots \in (2^{AP})^{\omega}$, where ω denotes an infinite repetition and $o_i \in 2^{AP}$, $\forall i \in \mathbb{Z}_{\geq 0}$. Given a word $o = o_0 o_1 \ldots$, denote by $o[j \ldots] = o_j o_{j+1} \ldots$, $o[\ldots j] = o_0 \ldots o_j$, and $o[i : j] = o_i \ldots o_j$ with $i < j$. The semantics of LTL formulae are interpreted over the word o as:

$$\begin{aligned}
&o \models true\\
&o \models ap \quad \Leftrightarrow ap \in o_0\\
&o \models \phi_1 \wedge \phi_2 \Leftrightarrow o \models \phi_1, o \models \phi_2\\
&o \models \neg\phi \Leftrightarrow o \not\models \phi\\
&o \models X\phi \Leftrightarrow o[1 \ldots] \models \phi\\
&o \models \phi_1 U \phi_2 \Leftrightarrow \exists i \text{ s.t. } o_i \models \phi_2, \forall j \in [0, i), o_j \models \phi_1
\end{aligned}$$

We denote by $o \models \phi$ if o satisfies ϕ. More details about LTL syntax, semantics, and model checking are referred to [41]. An LTL formula can be converted to a Non-deterministic Büchi Automaton (NBA) [42].

Definition 1: An NBA is a tuple $B = (S, S_0, \Sigma, \delta, S_F)$, where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, $\Sigma = 2^{AP}$ is the finite alphabet, $\delta : S \times \Sigma \rightarrow 2^S$ is a transition function, and $S_F \subseteq S$ is the set of accepting states.

Let B_ϕ denote the NBA of the LTL formula ϕ. Let $\Delta : S \times S \rightarrow 2^{\Sigma}$ denote the set of atomic propositions that enables state transitions in B_ϕ, i.e., $\forall o' \in \Delta(s, s')$, $s' \in \delta(s, o')$. A trajectory of NBA $\tau_B = s_0 s_1 s_2 \ldots$ generated by the word o with $o_{i+1} \in \Delta(s_i, s_{i+1})$ is called accepting, if τ_B intersects with S_F infinitely often. An LTL formula can be translated to an NBA by the tool [43]. In this paper, NBA will be used to track the progress of the satisfaction of LTL missions.

B. System Models

Consider a bounded workspace $Env \subset \mathbb{R}^2$ containing l non-overlapped regions of interest. Denote by E_i the ith region of interest, $E_I = \cup_{i=1}^{l} E_i$ the union of the regions of interest, and E_U the region of non-interest (e.g., obstacles or the regions that the robots can not traverse or operate within), where $E_I \cap E_U = \emptyset$ and $E_i \cap E_j = \emptyset$ with $i \neq j$ and $\forall i, j \in [l]$. Let $L : Env \times \mathbb{R}^+ \rightarrow AP$ be a labeling function indicating the atomic proposition executable at a given area in Env at certain time stamps.

In this work, we consider a ground mobile robot $\mathcal{R}$ (e.g., a wheeled robot or a quadruped robot) operating in Env. The robot is governed by the dynamics $\dot{x} = f(x(t), u(t))$, where $x(t) = \left[p^T(t), \theta(t)\right]^T$ represents the robot state including the position $p \in \mathbb{R}^2$ and orientation $\theta \in \mathbb{R}$, $\dot{x}$ represents the linear velocity $v(t) \in \mathbb{R}^2$ and angular velocity $\omega(t) \in \mathbb{R}$, and $u(t) \in \mathbb{R}^2$ represents an admissible input. Let X be the state trajectory starting from an initial state $x(t_0)$. Then, let P be the corresponding robot path in Env with $P(t_0)$ indicating the robot initial position. Following the control notations [44], we denote by $X \models \mathcal{R}$ if X is generated following the robot dynamics $\dot{x} = f(x(t), u(t))$.

III. Extended Predicate-Based Temporal Logic

Expressing robotic tasks with LTL specifications allows us to use NBA to verify its satisfaction, whose transitions are based on atomic propositions. However, the LTL formula might not fully express the semantics of a real-world mission. Although STL formulas can describe explicit time constraints, its solution is based on optimization method which requires long solution time and is difficult to apply in real-time system. In this section, we propose extended predicate-based temporal logic (E-pTL) defined on continuous time and introduce time, space, and order attributes to the atomic proposition. It not only enriches the semantic scope of conventional LTL, but also can be checked by NBA, which greatly facilitates the real-time motion planning of robotic systems in various scenarios.

Definition 2: [E-pTL] The extended predicate-based temporal logic formula is defined by augmenting the conventional LTL with a labeling function $SA : AP \times \mathbb{R}^+ \times (2^{AP})^{\omega} \rightarrow \{true, false\}$ to indicate the satisfaction of ap (i.e., whether ap is executable) by taking into account time, space, and order constraints of ap.

As stated in [15], interpreting the next operator as the next integration step is too implementation-dependent to serve as a solid basis. Hence, the operator X is not considered in E-pTL.

Let $\Delta t_i = i\Delta_t, i \in \mathbb{Z}_{\geq 0}$, be a time sequence, where $\Delta_t \in \mathbb{R}^+$ is a sampling period. The word $o_t(t)$ defined on continuous time $t \in \mathbb{R}^+$ can be sampled to yield a sequence of time stamped values $o_t(\Delta t_0) o_t(\Delta t_1) \ldots o_t(\Delta t_i) \ldots$ with $o_t(\Delta t_i) \in 2^{AP}$ indicating the word at time Δt_i. To facilitate the following analysis, we denote by (o_t, t) the trajectory starting from time t, i.e., sampled at the time stamps $t, t+\Delta t_1, \ldots, t+\Delta t_i \ldots$. For easy checking, based on o_t, we extract a discrete embedding sequence $o = o_0 o_1 \ldots$ and, $\forall t \in [t_i, t_{i+1}]$ with $i \in \mathbb{Z}_{\geq 0}$, $o_t(t) = o_i$ or $o_t(t) = \emptyset$. Similarly, given an E-pTL formula ϕ_{E-pTL}, we can extract an LTL formula ϕ_{LTL} as the temporal constraints of ϕ_{E-pTL}, such that o_t is satisfied with ϕ_{E-pTL} if o is satisfied with ϕ_{LTL} and $\forall ap \in o_i \in o$, $SA(ap, t, o_t) = true$, denoted as $o_t \models \phi_{E-pTL}$. Similar to LTL, the semantics of E-pTL are defined over the trajectory (o_t, t) starting from time t as

$$\begin{aligned}
&(o_t, t) \models true\\
&(o_t, t) \models \phi_1 \wedge \phi_2 \Leftrightarrow (o_t, t) \models \phi_1, (o_t, t) \models \phi_2\\
&(o_t, t) \models \neg\phi \quad \Leftrightarrow (o_t, t) \not\models \phi\\
&(o_t, t) \models \phi_1 U \phi_2 \Leftrightarrow \exists i \text{ s.t. } (o_t, t_i) \models \phi_2, \forall t_j \in [0, t_i),\\
&(o_t, t_j) \models \phi_1.
\end{aligned}$$

The difference between the semantics of LTL and E-pTL is

$$(o_t, t) \models ap \Leftrightarrow ap \in o_t(t) \text{ and } SA(ap, t, o) = true,$$

where o is the embedded word of o_t. Specifically, given an atomic proposition $ap \in AP$, the word o_t, and the current time t, the task label of ap is defined as

$$SA(ap, t, o) \triangleq time(ap, t) \wedge space(ap, t) \wedge order(ap, o),$$

where $time(ap, t) \in \{true, false\}$ is the time constraint, $space(ap, t) \in \{true, false\}$ is the space constraint, and $order(ap, o) \in \{true, false\}$ is the order constraint. That is, ap is labeled true if and only if it satisfies the time, space, and order constraint simultaneously. These constraints are determined by several attributes of atomic positions, which are explained in detail in the sequel.

A. Time Constraint

The time constraint, $time(ap, t)$, links to the time attribute $TI(ap)$ of atomic proposition ap. $TI(ap) = (t_s, t_e, t_r(ap'))$ signifies the feasible time interval for sub-task $ap \in AP$, where $ap' \in AP \cup true$ denotes the atomic proposition to be completed before ap with $t_r(ap')$ indicating the execution time stamp of ap'. The interval $[t_s, t_e]$ indicates the executable time window of ap relative to $t_r(ap')$. That is, the absolute time interval is $[t_s + t_r(ap'), t_e + t_r(ap')]$. For a given word o_t where $ap \in o_i$, the execution time stamp of ap' is determined as

$$t_r(ap') = \begin{cases} \infty, & ap' \notin o[\dots i], \\ t_{ap'}, & ap' \in o[\dots i], \\ 0, & \text{else}, \end{cases}$$

which indicates the time adjustment of the feasible time interval of ap based on the execution of ap', where $t_{ap'}$ is the time stamp of completing ap'. Specifically, if ap' has been executed at $t_{ap'}$, i.e., $\exists t_{ap'} \in [0 : t_i)$, $ap' \in o_t(t_{ap'})$, $SA(ap', t, o) = true$, $TI(ap)$ is adjusted such that ap should be executed after the execution of ap' within the absolute time window $[t_s + t_{ap'}, t_e + t_{ap'}]$. If ap' has not been executed, i.e., $\forall t_{ap'} \in [0 : t_i)$, $ap' \notin o_{t_{ap'}}$ or $ap' \in o_{t_{ap'}}$, $SA(ap', t, o) = false$, we set $t_{ap'} = \infty$ to indicate that ap' has not been executed yet and thus ap cannot be executed. If $ap' = true$, then $t_r = 0$ and no adjustment of $TI(ap)$ is needed, the absolute time interval is $[t_s, t_e]$. The time constraint $time(ap, t)$ is then determined based on the time t and $TI(ap)$. If $t \in [t_s + t_r, t_e + t_r]$, then ap is satisfied with the time constraint, i.e., $time(ap, t) = true$.

B. Space Constraint

Let $space(ap, t)$ represent the space constraints defined based on ap's space attributes: $LC(ap, t)$ in Env as the location where ap operates at time t, $SC(ap)$ as the operational radius, and $ET(ap)$ as its execution duration. It is worth pointing out that $LC(ap, t)$ labels can change over time in a dynamic setting. For robot $\mathcal{R}$ and the time t', ap satisfies the space constraint, i.e., $space(ap, t) = true$, if the distance between the robot path $P(t)$ and $LC(ap, t)$ is within $SC(ap)$ for all $t \in [t', t' + ET(ap)]$.

C. Order Constraint

Let $order(ap, o)$ represent the order constraint defined based on the embedded word o and ap's attributes: $BF(ap)$ and $AF(ap)$, where $BF(ap), AF(ap) \subseteq AP$ denote the set of atomic propositions to be executed and avoided before ap, respectively. Therefore, if all ap' in $BF(ap)$ have been executed prior to ap in o, and all ap'' in $AF(ap)$ have not been executed, then $order(ap, o) = true$. Note that ap's ordering attribute can substitute certain time constraints like $(\neg ap_1) U ap_2$ (meaning ap_1 shouldn't be executed before ap_2) and $(\neg ap_2) U ap_1 \wedge Fap_2$ (meaning ap_1 must be executed before ap_2). Hence, the order constraint provides an alternative to express ordered sub-tasks. Since the order constraints can be encoded in atomic propositions, it is easier to construct temporal formulae, making the formula more compact. For instance, consider LTL formulae $\phi_1 = ((\neg ap_2 \wedge \neg ap_4) U ap_3) \wedge ((\neg ap_2 \wedge \neg ap_4) U ap_1) \wedge Fap_5$, $\phi_2 = ((\neg ap_2 \wedge \neg ap_4) U ap_3) \wedge Fap_2$, and $\phi_3 = (\neg ap_4) U ap_5 \wedge ((\neg ap_2 \wedge \neg ap_4) U ap_1)$. If ap_2 and ap_4 should be done after ap_1 and ap_3, such constraints can be encoded in ap_2 and ap_4 by $AF(ap_1) = \{ap_2, ap_4\}$ and $AF(ap_3) = \{ap_2, ap_4\}$. Then, the original LTL formulae can be rewritten in a more compact E-pTL formula as $\phi_1 = Fap_3 \wedge Fap_1$, $\phi_2 = Fap_3 \wedge Fap_2$, $\phi_3 = (\neg ap_4) U ap_5 \wedge Fap_1$.

Example 1: Consider a hospital environment as shown in Fig. 1, where the robot is tasked to get the medical report from the doctor who is walking around in the hospital (i.e., the mobile region 1) and then delivers it to the nurse in the nurse station (i.e., region 2) within $60s$. Afterwards, the robot should send the medicine to patient room 1 or 2 (i.e., region 3 and 4, respectively). To specify such a mission, we first construct a base LTL formula as $\phi = F(ap_1 \wedge (Fap_2)) \wedge F(ap_3 \vee ap_4)$, where ap_i, $i = 1, 2, 3, 4$, represents the visit of the ith region, respectively. Then, we augment the atomic propositions in ϕ with time, space, and order constraints to construct the corresponding E-pTL formula. As the medical report should be sent to the nurse station within $60s$, we set $TI(ap_2) = (0, 60, t_r(ap_1))$, where $t_r(ap_1)$ is the time stamp that ap_1 has been executed. The mobile and static regions are set as $LC(ap_1, t) = (1.5, 4 + 2\sin(\frac{\pi t}{20}))$, $LC(ap_2, t) = (8.5, 6)$, $LC(ap_3, t) = (1.5, 9)$, $LC(ap_4, t) = (8.5, 9)$. Since the medicine should be fetched from the nurse station first before delivering to the patient room, we set $BF(ap_4) = \{ap_2\}$ and $BF(ap_3) = \{ap_2\}$. Suppose there exists a path P and a word o_t, satisfying $L(P(t)) = o_t(t)$, $\forall t \geq 0$. The embedded word of o_t is $o = ap_1 ap_2 ap_4$ with time stamps t_1, t_2, and t_3, respectively. If $t_2 \in [0 + t_1, 60 + t_1]$, then $time(ap_2, t_2) = true$. If $\forall t \in [t_1, t_1 + ET(ap_1)]$, $\|P(t) - LC(ap_1, t)\| \leq SC(ap_1)$, then $space(ap_1, t) = true$. As there exists ap_2 occurring before ap_4, the order constraints of ap_4 in o are satisfied, i.e. $order(ap_4, o) = true$.

In conventional product automaton based approaches, the state transitions are always described in a weighted transition system (wTS). A wTS of a robot in Env is a tuple $T = (\mathcal{P}, p_0, \rightarrow_T, AP, L_{\mathcal{P}}, C_T)$, where $\mathcal{P}$ is the set of infinite (continuous) positions in Env, p_0 is the initial position of robot; $\rightarrow_T \subseteq \mathcal{P} \times \mathcal{P}$ is the transition relation that considers robot dynamics as [45]. For instance, $p \rightarrow_T p'$ exits if there exist collision-free navigation control inputs driving the robot from p to p'. AP is the set of atomic propositions as the

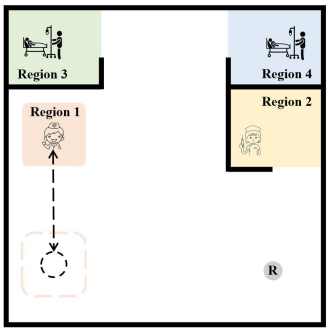


Fig. 1. A hospital environment where the thick black lines indicate the walls, region 1 indicates the doctor, region 2 indicates the nurse station, and region 3 and 4 indicate the patient room 1 and 2, respectively. The gray circle represents the robot.

labels of regions, $L_{\mathcal{P}} : \mathcal{P} \to AP$ is the labeling function that returns an atomic proposition satisfied at a location p, and $C_T : (\to_T) \to \mathbb{R}^+$ indicates the user specified transition cost, e.g., the Euclidean distance. However, $\to_T$ and $L_{\mathcal{P}}$ in T are determined without considering time stamps, and thus are difficult to be applied in the environment with dynamic areas of interest. To address this issue, we directly encode L as space constraints in E-pTL as it contains the time stamp t and can describe dynamic areas of interest. The time constraints can also be defined by L, i.e., $\forall ap \in AP$, $\forall t \in \mathbb{R}^+$, $\forall p$ s.t. $\|LC(ap,t) - p\| \leq SC(ap)$, one has $L(p,t) = ap$. Hence, wTS is no longer needed in E-pTL.

For an E-pTL formula ϕ, the plan is defined as $\tau = \tau^0\tau^1\tau^2\ldots$, where $\tau^i = (o_i, s_i)^{t_i}$. Each $\tau^i \in \tau$ indicates that the robot will complete the sub-tasks represented by $o_i \in 2^{AP}$ at the time stamp $t_i \in \mathbb{R}^+$, and the NBA state will transitions to s_i after completing the sub-tasks. Let τ^0 denote the initial plan tuple where $s_0 \in S_0$, $t_0 = 0$. Since the areas of interest do not overlap and each area is related to one atomic proposition, the robot can only execute one task once. Therefore, we consider $o_i \in AP$ in this work.

Given an E-pTL formula ϕ with its corresponding LTL formula ϕ_{LTL} and a robot system $\mathcal{R}$, we denote by $\tau \models (\phi, \mathcal{R})$, if the word $o = o_0 o_1 \ldots$ extracted from τ satisfies $o \models \phi_{LTL}$ and $SA(ap,t,o) = true$, $\forall ap \in o_i \in o$, and the robot state trajectory holds $X \models \mathcal{R}$ with the the associated path satisfying

$$\forall i \in \mathbb{Z}_{>0}, P(t) = LC(o_i, t), \forall t \in [t_i, t_i + ET(o_i)]. \quad (1)$$

Similar to LTL, an infinite plan τ of E-pTL also admits a sequence in the form of prefix-suffix structure, i.e., $\tau = \tau_{pre}\tau_{suf}\tau_{suf}\ldots$, where the prefix part τ_{pre} is only executed once and the suffix part τ_{suf} is executed infinitely often. However, as it is difficult to construct a lasso-shaped plan in a dynamic environment, we define the infinite plan as $\tau = \tau_{pre}\tau^1_{suf}\tau^2_{suf}\tau^3_{suf}\ldots$ to model the suffix stage with different plan tuples. For the sake of brevity, τ_{suf} is used as a shorthand for τ^1_{suf} when the context is clear. The cost of a finite plan $\tau = \tau^0\tau^1\ldots\tau^{|\tau|}$ is defined as the time stamp that the task is completed, i.e., $C(\tau) = t_{|\tau|}$. Hence, the problem is formulated as follows.

Problem 1: Given the map Env, the robotic system $\mathcal{R}$, an E-pTL formula ϕ, the goal is to obtain a feasible plan $\tau = \tau_{pre}\tau^1_{suf} = \tau^0\tau^1\ldots\tau^{|\tau|}$ with small cost $C(\tau)$ and continuously construct the following suffix stage τ^{i+1}_{suf} from the end state of τ^i_{suf}, $i = 1, \ldots,$ on the fly for the dynamic environment.

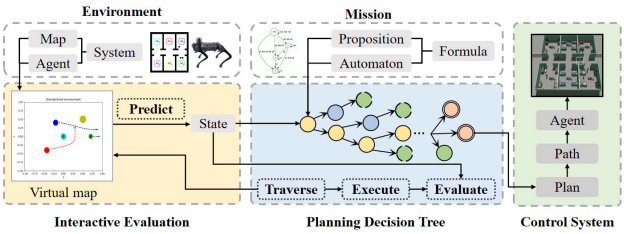


Fig. 2. Overview of planning decision tree (PDT) based motion planning. Taking advantage of E-pTL in associating time, space, order constraints to the atomic proposition, the developed PDT enables fast motion planning with guaranteed completeness and satisfaction of task specifications.

IV. Planning Decision Tree

Due to the introduced attributes, traditional automaton-based methods using model checking cannot handle E-pTL motion planning. Attributes like $TI(ap)$ and $LC(ap,t)$, particularly time-related ones, cannot be adequately modeled by conventional transition systems. Maity and Baras in [32] devised a transition system employing time-stamped discretization for time and space, solving plans in dynamic environments with bounded time temporal logic. However, this approach has high solution complexity due to dense discrete time and doesn't accommodate continuous time or environment-related problems. Furthermore, attributes tied to other propositions, like the relative time interval from $TI(ap)$ and the order attribute $BF(ap)$, cannot be represented in standard state-based graph search methods. For the sampling-based methods, the incremental approach can verify the atomic proposition satisfaction for current system states in the progress of expansion. However, the rewiring for optimization will lead to the change of cost and execution time, which is not suitable for the time window attribute and dynamic areas of interest. One alternative is to cast it into a mixed integer linear program (MILP). However, MILP often suffers from high computational costs and scalability issues and requires discretization of time and space. To tackle these challenges, this section proposes a novel framework, namely planning decision tree (PDT), leveraging the fact that mission attributes correspond to E-pTL atomic propositions. This framework enables rapid motion planning while ensuring completeness, i.e., if a solution exists, it is guaranteed to be found and satisfies the E-pTL formula by the PDT-based motion planning.

The main idea of E-pTL is to develop a tree to represent the task progression and update the system states. Starting from the root node that indicates the initial state of task and system, a series of plan tuples will be searched as in Alg. 2. Then, the predict module will forward to simulate the sub-task execution process to obtain the sub-task completion time and ensure the feasibility of the sub-task as in Alg. 3. The system state and cost of each plan tuple will be returned to Alg. 2 for the evaluation of the plan. Through the pruning method in Alg. 4, the tree can be constructed quickly and Alg. 5 will select the plan with minimum cost. For an overview of the PDT-based E-pTL motion planning, refer to Fig. 2.

A. Planning Decision Tree

To address Problem 1, we develop a planning tree to record feasible plan tuples, system states and cost values.

Definition 3: The planning tree $\mathcal{T}_\mathcal{D}$ is constructed based on a set of nodes $\{Nd_i\}$, $i \in \mathbb{Z}_{\geq 0}$, where Nd_0 represents the root of the tree. Each node in $\mathcal{T}_\mathcal{D}$ is defined as a tuple

$$Nd_i = \left(B_s, P_s, Prog, ET_{pre}, EP_{pre}, V_c\right)$$

where

- $B_s \in S$ denotes the automaton state of node i;
- $P_s \in AP$ denotes the atomic proposition of node i;
- $Prog = \{pre, oth\} \cup S_F$ indicates the current task progress, where pre and oth represent the prefix stage and other cases, respectively, and $s \in S_F$ represents the start and end state of cyclic path if in the suffix stage.
- ET_{pre} denotes the predicted time that the robot $\mathcal{R}$ completes the previous task;
- EP_{pre} denotes the predicted locations of the robot $\mathcal{R}$ for the previous task;
- V_c measures the performance of node Nd_i from initial node Nd_0;

Throughout this work, we will adopt the data structure to represent the node components, e.g., $Nd_i.B_s$ represents the automaton states associated with node i. Then, the node sequence indicates a plan τ, where each node in the tree indicates a plan tuple $\tau^i = (o_i, s_i)^{t_i} = (Nd.P_s, Nd.B_s)^{Nd.ET_{pre}} \in \tau$. $Nd.Prog$ indicates the task progress for plan tuple $(Nd.P_s, Nd.B_s)^{Nd.EP_{pre}}$, which is used for the tree pruning and the indication of task completion. Besides, the tree-related attributes are defined as functions. Let $FlagTra : Nd \rightarrow \{0, 1\}$ be a flag, where $FlagTra(Nd) = 1$ if Nd cannot generate its child nodes and 0 otherwise. Let $Parent(Nd)$ denote the father node of Nd and $\mathcal{P} = \mathsf{Node\_Path}(Nd)$ indicates the sequence of nodes from Nd_0 to Nd, respectively, which record the structure of planning tree $\mathcal{T}_\mathcal{D}$. $PreS(Nd)$ records the automaton states of the nodes with the same mission progress in $\mathsf{Node\_Path}(Nd)$, which is applied to avoid repeated exploration. And $PreAP(Nd)$ records the atomic propositions in $\mathsf{Node\_Path}(Nd)$, which is applied to verify the order constraints. The node properties are illustrated in Fig. 3. Note that, for any node Nd_i in $\mathcal{T}_\mathcal{D}$, it holds that $Nd_i.P_s \in \Delta((Parent(Nd_i)).B_s, Nd_i.B_s)$ and $Nd_i.EP_{pre} = LC(Nd_i.P_s, Nd_i.ET_{pre})$. By $Nd_i.ET_{pre}$, $(Nd.P_s, Nd.B_s)$ is guaranteed to be a feasible plan tuple satisfying (1).

The Algorithm 1 outlines how the tree $\mathcal{T}_\mathcal{D}$ is constructed. After generating B_ϕ from the E-pTL formula ϕ, $\mathcal{T}_\mathcal{D}$ is first initialized as the root node Nd_0. The algorithm ends if all nodes have been traversed. Otherwise, for each node Nd that has not been traversed, the function $\mathsf{Children\_Generation}$ is invoked to generate the set of child nodes $Children$. To limit the tree expansion, the function Bound is further applied. The child nodes $Children$ are then added to the tree $\mathcal{T}_\mathcal{D}$ and the parent node Nd is marked as traversed. After all nodes have been traversed, the function $\mathsf{Get\_Plan}$ is used to generated the satisfactory plannings τ_{pre} and τ_{suf}. The overview of PDT is illustrated in Fig. 4.

Example 2: Continue with the task in Example 1. The NBA corresponding to $\phi = F(ap_1 \wedge (Fap_2)) \wedge F(ap_3 \vee ap_4)$ is shown in Fig. 5(a). In each expansion, the child node

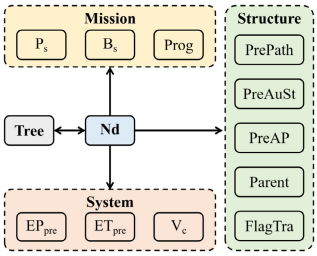


Fig. 3. Each node Nd consists of 11 properties, which can be classified into system properties, mission properties, and structure properties. The mission properties indicate the mission planning related states, i.e., $\tau^i = (o_i, s_i)^{t_i} = (Nd.P_s, Nd.B_s)^{Nd.ET_{pre}} \in \tau$ and $Nd.Prog$ indicates the mission progress. The system properties record the task related metrics and cost value from Nd_0. The structure properties indicate the parameters related to tree constructions.

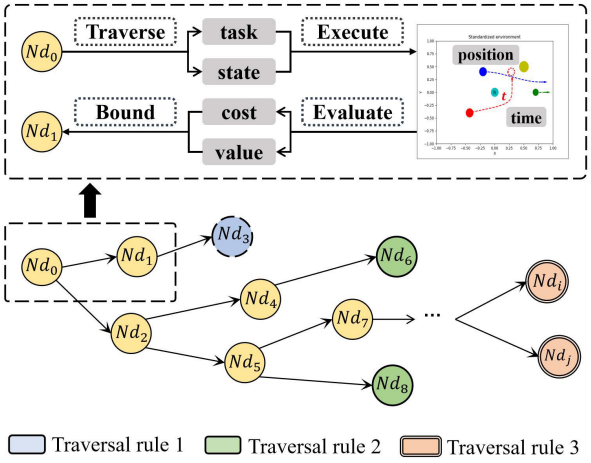


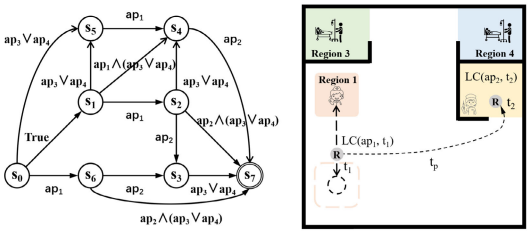
Fig. 4. The framework of the planning decision tree for the E-pTL. By expansion of parent nodes, the tree can obtain the child nodes until no parent nodes can be expanded. Each parent node can obtain the tasks and states by traversal. Then, the robot can execute the task in the virtual environment and obtain the predicted position and the time of arrival. Next, it will evaluate the progress of execution and obtain the value and cost. Finally, after pruning by Bound function, child nodes with all information are returned to the tree for the next expansion.

Algorithm 1 Planning Decision Tree

Input: E-pTL formula ϕ, workspace Env, root node Nd_0
Output: τ_{pre}, τ_{suf}

Convert ϕ to NBA B_ϕ;
Initialize the $\mathcal{T}_\mathcal{D} = \{Nd_0\}$;
while *1* **do**
  if $FlagTra(Nd) = 1,\ \forall Nd \in \mathcal{T}_\mathcal{D}$ **then**
    break;
  end
  for $Nd \in \mathcal{T}_\mathcal{D}$, *s.t.* $FlagTra(Nd) = 0$ **do**
    $Children = \mathsf{Chilren\_Generation}(Nd, \mathcal{T}_\mathcal{D})$;
    $[Chilren, \mathcal{T}_\mathcal{D}] = \mathsf{Bound}(Children, \mathcal{T}_\mathcal{D})$;
    add $Children$ to $\mathcal{T}_\mathcal{D}$;
    set $FlagTra(Nd) = 1$;
  end
end
$[\tau_{pre}, \tau_{suf}] = \mathsf{Get\_Plan}(\mathcal{T}_\mathcal{D})$;
Return τ_{pre}, τ_{suf};

will obtain its mission states and predict its system states. Suppose there exists a node Nd_1 in $\mathcal{T}_\mathcal{D}$ with $Nd_1.P_s = ap_1$, $Nd_1.B_s = 6$, $Nd_1.Prog = pre$. Then, the child node $Child$ of Nd_1 is generated as $Child.P_s = ap_2$, $Child.B_s = 3$, $Child.Prog = pre$. Based on the mission states of $Child$, the predicted completion time of ap_1 is t_1 and the predicted



(a) (b)

Fig. 5. (a) The NBA of task ϕ. (b) The predicted path and time of progress from ap_1 to ap_2.

Algorithm 2 Children_Generation

Input: Nd, $\mathcal{T}_\mathcal{D}$
Output: $Children$
1 **for** $s \in (S/PreS(Nd))$ **do**
2 **for** $ap \in \Delta(Nd.B_s, s)$ **do**
3 **if** $\exists ap^* \in BF(ap), ap^* \notin PreAP(Nd)$ *or* $\exists ap^* \in AF(ap), ap^* \in PreAP(Nd)$ **then**
4 continue;
5 **end**
6 Initialize $Child$ by s and ap;
7 Record structure of $Child$;
8 **if** $Child.Prog = Nd.Prog$ **then**
9 $PreS(Child) = \{PreS(Child), s\}$;
10 **else**
11 $PreS(Child) = \emptyset$;
12 **end**
13 Obtain $TI(ap)$ by Nd and $\mathcal{T}_\mathcal{D}$;
14 $[Child.ET_{pre}, Child.EP_{pre}, Child.V_c] =$ Predict$(Nd, Child, B_\phi, ap, TI)$;
15 Add $Child$ to $Children$;
16 **end**
17 **end**
18 Return $Children$;

location of robot is $LC(ap_1, t_1)$, which is shown in Fig. 5(b). Then, the predicted completion time of ap_2 is $Child.ET_{pre} = t_2 = t_1 + t_p + ET(ap_2)$ where t_p is the predicted arrival time from ap_1 to ap_2. And the predicted completion location is $Child.EP_{pre} = LC(ap_2, t_2)$. Based on the predicted path, cost value $Child.V_c$ can also be obtained by invoking Alg. 3.

B. Tree Expansion

The tree $\mathcal{T}_\mathcal{D}$ grows by generating child nodes. As explained in Alg. 2, the function Children_Generation first searches the mission states $s \in S$ and the propositions $ap \in AP$ under the state transition Δ. The proposition ap that satisfies the mission order constraints is selected (lines 3-5). The new node $Child$ is then created based on the mission states of the selected task, i.e., $Child.B_s = s$, $Child.P_s = ap$, $Child.Prog =$ Flag$(Nd.Prog, s)$, where Flag is defined as

$$\text{Flag}(prog, s) = \begin{cases} prog, & \text{if } s \notin S_F, \\ s \in S, & \text{if } s \in S_F, prog = pre, \\ oth, & \text{if } s = prog. \end{cases}$$

Then, structure related parameters are set as $Parent(Child) = Nd$, $Node\_Path(Child) = \{Node\_Path(Nd), Nd\}$, $PreAP(Child) = \{PreAP(Nd), ap\}$ and $PreS(Child)$ will be updated according to $Child.Prog$ (line 8-12), which records the automaton states with the same task progress to avoid repeated exploration as shown in line 1. The predicted value and system states are obtained based on the selected task ap (lines 13-14). Since the time interval of task ap may be related to other tasks, based on $Nd.ET_{pre}$ the time interval $TI(ap) = (t_s^{abs}, t_e^{abs}, Nd.ET_{pre})$ is obtained and applied. The function Predict is developed to predict the completion time $Child.ET_{pre}$, the completion position $Child.EP_{pre}$, and the cost value $Child.V_c$ of the child node. Finally, the $Child$ node is added to the set $Children$.

Algorithm 3 Predict

Input: Nd, $Child$, B_ϕ, ap, TI
Output: e_t, e_p, e_c
1 Initialize $AP_{inf} = \emptyset$;
2 **for** $ap_{oth} \in AP/ap$ **do**
3 **if** $(ap_{oth} \wedge ap) \notin \Delta(Nd.B_s, Child.B_s)$ **then**
4 add ap_{oth} to AP_{inf};
5 **end**
6 **end**
7 $env =$ Get_env$(AP_{inf}, ap, Nd.ET_{pre})$;
8 $[e_{arr}, e_{dis}] =$ Path_Plan$(env, \mathcal{R})$;
9 $e^*_{arr} = \max(e_{arr}, t_s^{abs} - Nd.ET_{pre})$, $e_t = Nd.ET_{pre} + e^*_{arr} + ET(ap)$, $e_p = LC(ap, e_t)$;
10 $e_c =$ Evaluate$(Nd.V_c, e^*_{arr}, e_{dis}, ap)$;
11 Return e_t, e_p, e_c;

By function Children_Generation, mission related child nodes can be obtained. To complete the task successfully, the predicted position and time duration of the task are desired. Due to the consideration of dynamic environment, we construct a virtual environment with the time-dependent target area and obstacles within certain time steps, in which we can predict the estimated arrival time and position by path planning, as well as the cost value. This gives rise to the function Predict in Alg. 3. Since PDT only generates the mission plan, not the actual trajectory, we need to guarantee that the execution path of the mission does not violate the specifications. Therefore, the set of infeasible atomic propositions, denoted as AP_{inf}, is firstly obtained in lines 1-6. The set of areas $\{p \in Env | \|p - LC(ap_{inf}, t)\| \leq SC(ap_{inf}), ap_{inf} \in AP_{inf}\}$ is inaccessible at time t, which can be considered as dynamic barriers. For the atomic proposition $ap_{oth} \in AP/ap$, if $(ap_{oth} \wedge ap) \notin \Delta(Nd.B_s, Child.B_s)$, it indicates that ap_{oth} is inaccessible when executing the sub-task ap, which is added to the set AP_{inf}. In line 7, in order to establish a viable trajectory from $Nd.P_s$ to $Child.P_s$, a virtual dynamic environment env is constructed by the function Get_env, which considers the areas labeled with infeasible atomic proposition in AP_{inf} as barriers from $Nd.P_s$ to $Child.P_s$. The virtual dynamic environment contains the predicted trajectory and size of the moving target and barriers for the next several time stamps from the current time $Nd.ET_{pre}$. After path planning based on the virtual dynamic environment and robotic system, we can obtain the path and further obtain the arrival time e_{arr} and the minimum distance e_{dis} to the barriers by the dynamic path planner. If there is no feasible path, then e_{arr} is set as $Cost_{max}$, and $e_{dis} = 0$. As the robot may arrive at ap before t_s^{abs}, the actual travelling time may be larger than the arrival time, denoted as e^*_{arr}, which is defined as the maximum of the arrival time e_{arr} and the fastest arrival time $t_s^{abs} - Nd.ET_{pre}$, i.e., $e^*_{arr} = \max(e_{arr}, t_s^{abs} - Nd.ET_{pre})$.

Then, the completion time and position of ap are defined as $e_t = Nd.ET_{pre} + e^*_{arr} + ET(ap)$ and $e_p = LC(ap, e_t)$, respectively.

Assumption 1: It is assumed that the target areas move slower than the robot.

Assumption 1 indicates that the robot can follow the target until it enters the time interval, i.e., $\max(e_{arr}, t_s^{abs} - Nd.ET_{pre}) + Nd.ET_{pre} \geq t_s^{abs}$, which indicates early arrival. Then, the robot can follow the moving target location $LC(ap)$ within the execution time $ET(ap)$ under Assumption 1.

When facing a dynamic environment, the path can be altered to account for temporary obstacle avoidance. Therefore, we define a cost considering possible path alteration. As the paths close to obstacles are more likely to change, we define

$$\text{Evaluate}(c_0, e^*_{arr}, e_{dis}, ap) = c_0 + (1 + \frac{\alpha}{e_{dis}})e^*_{arr} + ET(ap), \tag{2}$$

to indicate the cost from the initial state to the current atomic task ap. The cost in (2) contains the previous cost value c_0, the cost for travelling $(1 + \frac{\alpha}{e_{dis}})e^*_{arr}$, and the execution time of ap, where α is the coefficient related to the robotic system. As indicated in line 10 of Alg. 3, if the executing time is outside the time window, i.e., $Nd.ET_{pre} + e_{arr} > t_e^{abs}$ as e_{arr} is set as $Cost_{max}$, the cost value $e_c > Cost_{max}$ indicates the infeasible atomic sub-task.

C. Tree Pruning

In Alg. 2, $\mathcal{T}_\mathcal{D}$ grows by adding all possible child nodes, which can lead to dimension explosion. To mitigate this issue, the following traversal rules are developed to reduce the search space.

Definition 4: The traversal rules are defined as follows:

1) For any Nd in $\mathcal{T}_\mathcal{D}$, if $Nd.Prog = oth$, Nd will no longer be traversed;
2) For any Nd in $\mathcal{T}_\mathcal{D}$, the traversed states in $PreS(Nd)$ will not be sampled if the planning stage (i.e., plan prefix, plan suffix) remains the same;
3) For any Nd_i and Nd_j in $\mathcal{T}_\mathcal{D}$, if $\exists Nd_i.B_s = Nd_j.B_s$, $Nd_i.Prog = Nd_j.Prog$, and $Nd_i.V_c < Nd_j.V_c$, then the node Nd_j will no longer be traversed and the child nodes of Nd_j will be pruned off from $\mathcal{T}_\mathcal{D}$.

The traversal rule 2) can be found in Children_Generation (Line 1) and the rules 1) and 3) can be found in Bound to limit the tree expansion. The function Bound is designed to update the traversal flag $FlagTra(Nd)$. As shown in Alg. 4, if Nd and $Child$ have the same NBA state and $Nd.V_c \leq Child.V_c$, we set $FlagTra(Child) = 1$ to indicate that $Child$ will no longer be traversed (lines 2-11). It can reduce the number of traversals without affecting the completeness of PDT. Since $Child.Prog = oth$ indicates that the first suf stage has been finished, $Child$ should not be traversed for new child nodes, and thus we set $FlagTra(Child) = 1$ (lines 12-14). In lines 15-19, we remove $Child$ that does not satisfy the conditions to indicate that the next traversal is not required. Since $Child.V_c \geq Cost_{max}$ indicates the robot can not finish the task $Nd.P_s$ within the feasible time interval, we set $FlagTra(Child) = 1$ and the cost of all nodes in Node_Path($Child$) and $Child.V_c$ as $Cost_{max}$, where $Cost_{max}$ is a extremely large value (e.g., ∞). As there may exist nodes that have not generated their child nodes, there may exist other feasible node paths. Therefore, the traversal flags of these nodes in $\mathcal{T}_\mathcal{D}$ need to be reset. Since the node cost in the path from Nd_0 to $Child$ are updated to $Cost_{max}$, the nodes with the minimum cost value and the same NBA states and mission stage may also change. The traversal flag of these nodes are then reset by function Reset for the completeness of PDT. The analysis of completeness can be found in Section V. The nodes, whose traversal flag are reset, will be verified if there is a feasible path through the same automaton states and mission stages. Otherwise, we set $FlagTra(Child) = 0$.

Algorithm 4 Bound

Input: $Children$, $\mathcal{T}_\mathcal{D}$
Output: $Children$, $\mathcal{T}_\mathcal{D}$

```
1  for Child ∈ Children do
2      for Nd ∈ T_D do
3          if Nd.Prog ≠ Child.Prog or Nd.B_s ≠ Child.B_s
           then
4              continue;
5          end
6          if Nd.V_c ≤ Child.V_c then
7              FlagTra(Child) = 1;
8          else
9              Delete all child nodes of Nd from T_D;
10         end
11     end
12     if Child.Prog = oth then
13         FlagTra(Child) = 1;
14     end
15     if Child.V_c ≥ Cost_max then
16         FlagTra(Child) = 1;
17         Set the cost value of all nodes in Node_Path(Child)
           and Child.V_c as Cost_max;
18         Reset(T_D);
19     end
20 end
21 Return Children, T_D;
```

Example 3: Consider a node $Child$ and a node $Nd_{ori} \in$ Node_Path($Child$) such that $Child.V_c \geq Cost_{max}$ and $Nd_{ori}.V_c = Cost_{max}$. Due to the large cost of Nd_{ori}, the function Reset will be applied to identify another node $Nd_{replace}$ such that $Nd_{replace}.B_s = Nd_{ori}.B_s$, $Nd_{replace}.Prog = Nd_{ori}.Prog$, and $Nd_{replace}.V_c \leq Nd.V_c$ for any node Nd that satisfies $Nd.B_s = Nd_{ori}.B_s$ and $Nd.Prog = Nd_{ori}.Prog$. As Nd_{ori} has generated child nodes, $Nd_{replace}$ must have no child nodes since $Nd_{replace}$ had a larger cost value before setting $Nd_{ori}.V_c = Cost_{max}$. Therefore, $Nd_{replace}$ may generate a new node path, which satisfies ϕ and $\mathcal{R}$. So, function Reset will set $FlagTra(Nd_{replace}) = 0$. The other nodes in the Node_Path($Child$) also has their corresponding $Nd_{replace}$.

Lemma 1: With the traversal rules, the tree $\mathcal{T}_\mathcal{D}$ has finite nodes, which indicates that the tree $\mathcal{T}_\mathcal{D}$ can be generated in finite time.

Proof: By the traversal rule 1), there are only 3 stage in the $\mathcal{T}_\mathcal{D}$ and the node in the third stage is the leaf node. By the traversal rule 2), there are at most $|S|$ nodes with the same mission stage in one path. Therefore, the maximum length of $\mathcal{T}_\mathcal{D}$ is $2 \times |S| + 1$. Besides, by the traversal rule 3), there are

Algorithm 5 Get_Plan

Input: $\mathcal{T}_\mathcal{D}$
Output: τ_{pre}, τ_{suf}
1 Get $Nd_{min} \in \mathcal{T}_\mathcal{D}$, which $Nd_{min}.Prog = oth$ and $Nd_{min}.V_c$ is the minimum cost value;
2 $\tau_{suf} = [(Nd_{min}.P_s, Nd_{min}.B_s)]$;
3 $\tau_{pre} = \emptyset$;
4 $Nd = Nd_{min}$;
5 **while** $Nd \neq Nd_0$ **do**
6 **if** $Parent(Nd).Prog \in S$ **then**
7 add $(Nd_{min}.P_s, Nd_{min}.B_s)$ with time stamp $Nd.ET_{pre}$ to τ_{suf};
8 **end**
9 **if** $Parent(Nd).Prog = pre$ **then**
10 add $(Nd.P_s, Nd.B_s)$ with time stamp $Nd.ET_{pre}$ to τ_{pre};
11 **end**
12 $Nd = Parent(Nd)$;
13 **end**
14 Return τ_{pre}, τ_{suf};

Algorithm 6 Real Time framework

Input: ϕ, Env
1 Initialize the root node Nd_0 by initial task and system states;
2 Get initial plan $\tau = PDT(\phi, Env, Nd_0)$;
3 Initialize the index $i = 1$;
4 **while** *1* **do**
5 According to current plan tuple $\tau^{i(t_i)} = (o_i, s_i)^{t_i}$, set target $p_t = LM(o_i, t_i)$;
6 Control$(p_t, \mathcal{R})$;
7 **if** o_i *has been completed* **then**
8 $i = i + 1$;
9 **end**
10 **if** *plan* τ *has been finished, i.e.* $i > |\tau|$ **then**
11 Initialize the root node Nd_0 by current task and system states;
12 Get initial plan $\tau_{new} = PDT(\phi, Env, Nd_0)$;
13 Update plan $\tau = \tau\tau_{new}$;
14 **end**
15 **end**

only $(1 + |F|) \times |S|$ nodes that can generate child nodes. The maximum number of child nodes is $|S| \times |AP|$. Thus, the width of the tree $\mathcal{T}_\mathcal{D}$ is also finite. Therefore, the tree $\mathcal{T}_\mathcal{D}$ has finite nodes, which indicates that the tree $\mathcal{T}_\mathcal{D}$ can be generated in finite time. ■

Lemma 2: Setting the cost value of all nodes in the path as $Cost_{max}$ does not affect cost value of the new child nodes.

Proof: Assume that $Child.V_c \geq Cost_{max}$ and $\forall Nd \in$ Node_Path$(Child)$, set $Nd.V_c = Cost_{max}$. Set the node path Node_Path$(Child) = Nd_0Nd_1 \dots Nd_n$. Then, for Nd_i, $i < n$, Nd_i has child node Nd_{i+1} and for Nd_n, Nd_n has child node $Child$. Therefore, each node in the node path has generated its child nodes. Therefore, there is no new child node can be generated by the node in the node path, which indicates that the parent nodes of new child nodes in the following expansions are not in the node path. As the cost value is only related to its parent node, setting the cost value of all nodes in the path as $Cost_{max}$ does not affect cost value of the new child nodes. ■

D. Plan Generation

Function Get_Plan is used to get the plan from $\mathcal{T}_\mathcal{D}$. As shown in Alg. 5, the node Nd_{min} with the minimum cost in all completion nodes is first obtained, which satisfies, $\forall Nd \in \mathcal{T}_\mathcal{D}$, $Nd.Prog = oth$ and $Nd.V_c \geq Nd_{min}.V_c$. After initializing the plan τ_{pre} and τ_{suf}, the parent node $Parent(Nd)$ is retrieved until there is no parent node (root node Nd_0). The atomic proposition and NBA state of Nd are then constructed as a plan tuple with the time stamp $Nd.ET_{pre}$. The plan tuples are assigned to the corresponding plan (τ_{pre} or τ_{suf}) through $Parent(Nd).Prog$. Finally, the plan τ_{pre} and τ_{suf} are obtained.

E. Real-Time Planning for Dynamic Environments

As it is difficult to construct a lasso-shaped plan in a dynamic environment, the real-time planning framework is developed to track the progress of the plan and continuously construct the suffix plan based on the current environment as shown in Alg. 6. Specifically, through PDT, we first obtain the plan $\tau = \tau_{pre}\tau^1_{suf} = \tau^0 \dots \tau^i \dots \tau^{|\tau|}$ composed of the prefix and the first suffix plan. The online framework in lines 4-15 aims to trace the plan progression and update the plan accordingly. According to the current plan tuple τ^i, we can obtain the target position $p_t = LM(o_i, t_i)$ and control the robot to complete o_i. If the current plan tuple has been completed, the index then proceeds to $i+1$. When the current plan τ has been finished at time t, then a root node for the next suffix plan can be initialized as Nd_0 with $Nd_0.B_s = s_{|\tau|}$, $Nd_0.EP_{pre} = LM(o_{|\tau|}, t)$, $Nd_0.ET_{pre} = t$, $Nd_0.Prog = s_{|\tau|}$, which ensures that τ^2_{suf} starts from the end of τ^1_{suf}. After replanning, the next suffix plan τ^2_{suf} can be obtained and added to the original plan as $\tau = \tau_{pre}\tau^1_{suf}\tau^2_{suf}$. The above process will be repeated and continuously update τ on the fly.

V. Algorithm Analysis

A. The Performance of PDT

Before investigating PDT, we first show in Lemma 3 that Predict in Alg. 3 can select the node with feasible arrival time and path. Theorem 2 shows the PDT, without using the traversal rules, is guaranteed to find a feasible mission plan for the robot $\mathcal{R}$. We then show in Lemma 4 - 7 that the traversal rules do not compromise the feasibility and completeness of PDT, which concludes in Theorem 3 that PDT with traversal rules can search for the mission plans more efficiently.

Lemma 3: Given a PDT $\mathcal{T}_\mathcal{D}$, for any $Nd \in \mathcal{T}_\mathcal{D}$ with $Nd.V_c < Cost_{max}$, there always exists a feasible path between $(Parent.Nd).EP_{pre}$ and $Nd.EP_{pre}$ and it holds that the arrival time $Nd.ET_{pre} \in TI(Nd.P_s)$, i.e., the function Predict *can select the node with the feasible arrival time and path.*

Proof: Assume that the path planner we use is feasible and probabilistic complete, such as RRT [46] with receding horizon control [47]. Then, due to the feasibility of the path planner, the generated path P from the current position to a dynamic target is guaranteed to follow the system dynamics. Due to the completeness of the path planner, if there exists a feasible path, the path planner is also guaranteed to be found. If $Nd \in \mathcal{T}_\mathcal{D}$, $Nd.V_c < Cost_{max}$, then the arrival time e_t exists and $e_t \in TI$. Therefore, the path planner can find a feasible

path with a feasible arrival time. Hence, the function Predict can select the node with the feasible arrival time and path by $Nd.V_c$. ■

To facilitate the analysis, let Planning denote a function that maps the node sequence to a plan, i.e., $\tau = \tau_0 \dots \tau_{|\mathcal{P}|} = \text{Planning}(\mathcal{P})$, where $\tau_i = (Nd_i.P_s, Nd_i.B_s)^{(Nd_i.ET_{pre})}$, $i = 0, 1, \dots, |\mathcal{P}|$.

Theorem 1: Give the LTL formula ϕ, the robotic system $\mathcal{R}$, and the environment Env, if there exists a plan satisfying ϕ, the plan obtained by Alg. 1 must be satisfied with (1).

Proof: Suppose $\mathcal{P} = Nd_0 Nd_1 \dots Nd_n$ is the path generated by Alg. 5. Then, each node Nd_i in $\mathcal{P}$, $i \in \{1, \dots n\}$ satisfies $Nd_i.P_s \in \Delta(Nd_{i-1}.B_s, Nd_i.B_s)$ because of line 2 in Alg. 2, which indicates $\mathcal{P}$ satisfies (1). The plan prefix τ_{pre} and plan suffix τ_{suf} can then be obtained from $\mathcal{P}$ (i.e., the sub-path of $\mathcal{P}$ before entering the accepting states is τ_{pre} while the rest is τ_{suf}). Hence, the plan returned by Alg. 1 satisfies (1). ■

Theorem 2: Give the LTL formula ϕ, the robotic system $\mathcal{R}$, and the environment Env, if there exists a plan satisfying ϕ, the PDT in Alg. 1 is ensured to find it without using traversal rules.

Proof: Both path and mission planning are considered in PDT. Since the feasibility and completeness of path planning will not affect that the task planning as indicated in Alg. 2, they can be individually investigated. Given that the path has been proven to be feasible and complete in Lemma 3, we only need to show that the mission planning is also feasible and complete.

If there exists a plan τ_{pre} and τ_{suf} satisfying the task ϕ in the given environment Env, the corresponding trajectories of atomic proposition and automation states are τ_o and τ_s which satisfy $o_i \in \Delta(s_{i-1}, s_i)$, $i > 0$. Note that a core idea of PDT is to build a tree $\mathcal{T}_\mathcal{D}$. By definition, $Nd_0.B_s = s_0 \in \tau_s$, $Nd_0.P_s = true \in \tau_o$. Since $\mathcal{T}_\mathcal{D}$ traverses all possible nodes, there exists a node $Nd \in \mathcal{T}_\mathcal{D}$ such that $Nd.B_s = s_1 \in \tau_s$, $Nd.P_s = o_1 \in \tau_o$. Then, for $\forall(o_i, s_i) \in (\tau_s, \tau_o)$, there always exists a node $\tilde{Nd} \in \mathcal{T}_\mathcal{D}$, which satisfies $\tilde{Nd}.B_s = s_i$, $\tilde{Nd}.P_s = o_i$. Hence, if there exist feasible plans, PDT is ensured to find at least one feasible plan. ■

Theorem 2 shows that, without using traversal rules, the PDT in Alg. 1 is complete and feasible. The following lemmas and theorems show that the traversal rules only reduce the search space without compromising the feasibility and completeness of PDT. To facilitate the analysis, we define a cost function $Cost : \tau_o \times \tau_t \to \mathbb{R}$ that evaluates the performance of a task sequence $\tau_o = o_0 o_1 \dots o_e$. Let $\tau_t = t_0 t_1 \dots t_e$ denote the time stamp sequence, where $t_i \in \tau_t$ is the time stamp that $o_i \in \tau_o$ is finished. And define the optimal time sequence $tBest : \tau_o \to \tau_t = t_0 t_1 \dots t_e$, which satisfies the robotic system with the minimum t_e.

Lemma 4: Suppose there are two task sequences $\tau_o = o_0 \dots o_e$, and $\tau_o^* = o_0 \dots o_t \dots o_e$, where τ_o^* is the same with τ_o but differs in containing additional task o_t in the middle. It holds that $Cost(\tau_o, \tau_t) \le Cost(\tau_o^*, \tau_t^*)$.

Proof: Given two task sequences $\tau_o^1 = o_0 o_e$ and $\tau_o^2 = o_0 o_1 o_e$. Let $\tau_t^1 = tBest(\tau_o^1) = [t_0, t_e^1]$ and $\tau_t^2 = tBest(\tau_o^2) = [t_0, t_1^2, t_e^2]$ denote the optimal time sequence for τ_o^1 and τ_o^2, respectively. Under Assumption 1, if $t_e^1 > t_e^2$, then $\tau_t^* = t_0, t_1^2, t_e^2, t_e^1$ is also satisfies with the robotics system. Then, there exists $\tau_t^{**} = t_0 t_e^2$, $Cost(\tau_o^1, \tau_t^{**}) \le Cost(\tau_o^1, \tau_t^1)$, which violates the definition that τ_t^1 is the optimal time stamp sequence. Therefore, the assumption $t_e^1 > t_e^2$ is invalid. Hence, $Cost(\tau_o^1, \tau_t^1) = t_e^1 \le Cost(\tau_o^2, \tau_t^2) = t_e^2$. Then, for $\tau_o^i = o_0 o_1 \dots o_{i-1} o_e$, $Cost(\tau_o^1, \tau_t^1) \le \dots \le Cost(\tau_o^{i-1}, \tau_t^{i-1}) \le Cost(\tau_o^i, \tau_t^i)$. Therefore, Lemma 4 is proved. ■

Lemma 5: If $\tau = (\tau_o, \tau_s)^{(\tau_t)} = \tau_{pre}\tau_{suf}$ denotes a least-cost plan, the states in $\tau_s \in \tau_{pre}$ are all different, i.e., $s_i \ne s_j, \forall s_i, s_j \in \boldsymbol{s}, i \ne j$. And it is the same for $\tau_s \in \tau_{suf}$.

Proof: Take τ_{pre} as an example. Suppose its corresponding states sequence is $\tau_s = s_0 s_1 \dots s_n$ and the task sequence is $\tau_o = o_0 o_1 \dots o_n$. If there exists $s_i = s_j \in S$, $i \le j$, satisfying $\Delta(s_i, s_{j+1}) \ne \emptyset$ and $o_{j+1} \in \Delta(s_i, s_{j+1})$, there must exist a new state sequence $\tau_s^* = s_0 \dots s_i s_{j+1} \dots s_n$ and a new task sequence $\tau_o^* = o_0 \dots o_i o_{j+1} \dots o_n$ that satisfy the same task ϕ in the environment M. According to Lemma 4, there exists a plan whose cost is smaller than τ_{pre}, i.e., $Cost(\tau_o^*, tBest(\tau_o^*)) \le Cost(\tau_o, tBest(\tau_o))$, which contradicts that τ_{pre} is cost least. Therefore, the states in $\tau_s \in \tau_{pre}$ are all different. And it is same for $\tau_s \in \tau_{suf}$. Hence, Lemma 5 is proved. ■

Lemma 6: The traversal rule 2) does not increase the path cost and affect the completeness of PDT.

Proof: According to Theorem 2, without using the traversal rules, the PDT can still obtain all feasible plans. Suppose Nd_{end} is the leaf node with the least cost in $\mathcal{T}_\mathcal{D}$. Then $\mathcal{P} = Nd_0 Nd_1 \dots Nd_{end}$ is a least-cost path in $\mathcal{T}_\mathcal{D}$ generated by $\mathcal{P} = \text{Node\_Path}(Nd_{end})$ and the corresponding least-cost plan is $\tau = \text{Planning}(\mathcal{P})$, $\tau = (\tau_o, \tau_s)^{(\tau_t)}$. If there exist two nodes $Nd_i, Nd_j \in \mathcal{P}$, $i < j$, such that $Nd_i.B_s = Nd_j.B_s = s_k \in \tau_s$ and $Nd_i.Prog = Nd_j.Prog$, according to Lemma 5 $Cost(\tau_o^*, tBest(\tau_o^*)) \le Cost(\tau_o, tBest(\tau_o))$ with $\tau_o^* = [o_0, \dots, o_i, o_{j+1}, \dots, o_n]$. Therefore, $\mathcal{P}$ is not the least-cost path in $\mathcal{T}_\mathcal{D}$, which is against the assumption. Hence, the traversal rule 2) does not increase the path cost and affect the completeness of PDT. ■

Lemma 7: The traversal rule 3) does not affect the completeness.

Proof: Assume that all nodes satisfy the time condition, which indicates that $\forall Nd \in \mathcal{T}_\mathcal{D}$, $Nd.V_c < Cost_{max}$. Then, suppose there exist two nodes $Nd_i \notin \mathcal{P}$ and $Nd_j \in \mathcal{P}$ in $\mathcal{T}_\mathcal{D}$ such that $Nd_i.B_s = Nd_j.B_s$, $Nd_i.Prog = Nd_j.Prog$, $Nd_i.V_c \le Nd_j.V_c$. Then, due to $Nd_i.B_s = Nd_j.B_s$ and $Nd_i.Prog = Nd_j.Prog$, there exists a new node sequence $\mathcal{P}_{new} = Nd_0 \dots Nd_i Nd_{j+1} \dots Nd_{end}$ that satisfies the same task ϕ. It indicates that if the least-cost plan exists and is not in $\mathcal{T}_\mathcal{D}$ because of the traversal boundary rules, then there must still exist an approximate least-cost path satisfying the formula ϕ.

If there exists a node Nd that does not satisfy the time condition, which indicates that $\exists Nd \in \mathcal{T}_\mathcal{D}$, $Nd.V_c \ge Cost_{max}$. There may exist another node that can generate a new path satisfying the time condition. However, because of the traversal rule 3), all nodes with the same NBA states and stage flag as one node in $\mathcal{P} = \text{Node\_Path}(Nd, \mathcal{T}_\mathcal{D})$ can not be traversed. There may exist a node that can generate a new plan satisfying with the time condition although it has a larger cost value than the nodes in $\mathcal{P}$. Therefore, we should choose another node to replace the node in $\mathcal{P} = \text{Node\_Path}(Nd, \mathcal{T}_\mathcal{D})$ for another

generation. Then, based on lines 15-19 in Alg. 4, we change the cost value of all nodes in $\mathcal{P} = \mathsf{Node\_Path}(Nd, \mathcal{T}_\mathcal{D})$. By Lemma 2, the change of cost value does not affect the cost value of newly generated child nodes. And it can change the nodes with minimum cost value, which are the replaced nodes in $\mathcal{P}$. The Reset function can find them and generate another paths of nodes by PDT until all replaced nodes satisfy the time condition or all nodes have been traversed. By Theorem 2, if all nodes have been traversed and there is no node satisfying the time condition, then no feasible plan $\tau_{pre}\tau_{suf} \models (\phi, \mathcal{R})$ exists. Therefore, based on lines 15-19 in Alg. 4, the traversal rule 3) does not affect the completeness. ■

Theorem 3: PDT with Traversal Rules has feasibility and completeness.

Proof: By Theorem 2, Lemma 6, and Lemma 7, the feasibility and completeness of PDT are ensured with the traversal rules. Since function Predict guarantees the feasibility and completeness of the path, PDT with traversal rules are ensured to find the feasible plan $\tau_{pre}\tau_{suf} \models (\phi, \mathcal{R})$. ■

B. Complexity

Since the PDT algorithm is only mission-related, it can greatly reduce the algorithm complexity, making it suitable for reactive planning in real-time systems. After removing the nodes with the same flag and NBA states, $\mathcal{T}_\mathcal{D}$ contains at most $(1 + |F|) \times |S|$ nodes in two stages. Therefore, the space complexity for the number of NBA states is $O(n)$. As the path planning is decoupled with the task planning, the time complexity in this paper is only for task planning, which is measured by the times for generating child nodes and path planning. As there are only two stages during tree expansion and each NBA state can only be traversed once in each stage, based on the traversal rule 1) and 2), there are at most $2 \times |S|$ times traversal for one mission. In each traversal, at most $(1 + |F|) \times |S|$ nodes can generate child nodes based on the traversal rule 3). At most $|S| \times |AP|$ nodes can be generated. Then, the maximum number of times for generating child nodes and predicting the completion time is $2 \times (1 + |F|) \times |S|^3 \times |AP|$. Therefore, the maximum time complexity for the number of NBA states is $O(n^3)$, and the maximum time complexity for the number of atomic propositions is $O(n)$.

VI. Results

Numerical simulation and physical experiments are performed in this section to demonstrate the PDT based fast motion planning algorithm. LTL2STAR is used to convert the LTL formula to NBA [43]. Python 3.8 is used for simulation.

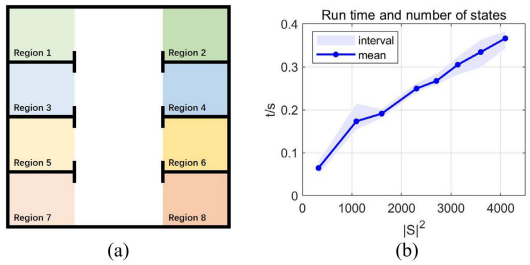
A. Time Complexity of PDT

As the core algorithm of our motion planning framework, we first evaluate the computation efficiency of the PDT algorithm in generating feasible plans. Consider a robot operating in an environment consisting of 8 areas of interest as shown in Fig. 6(a). A set of E-pTL formulas with different numbers of NBA states and atomic propositions are used to evaluate the computation efficiency of PDT for task planning. To evaluate its efficiency, receding horizon control (RHC) and

TABLE I

Runtime for Task and Path Planning Fordifferent E-pTL Formulas

$\|AP\|$	$\|S\|$	Task(/s)	Path(/s)	$\|AP\|$	$\|S\|$	Task(/s)	Path(/s)
3	8	0.0240	0.00891	6	56	0.305	0.182
4	16	0.0587	0.0340	6	60	0.334	0.214
5	32	0.155	0.0867	6	64	0.366	0.221
6	18	0.0643	0.0244	7	96	0.588	0.320
6	33	0.173	0.0707	7	128	0.873	0.452
6	40	0.191	0.105	8	192	1.57	0.810
6	48	0.250	0.138	8	256	2.33	1.12
6	52	0.267	0.124				



(a) (b)

Fig. 6. (a) is the test environment, which consists of 8 areas of interest corresponding to 8 atomic propositions. The thick black lines indicate the barriers that the robot cannot traverse. (b) is the relationship between solution time and number of mission states in task planning. X-axis is the number of squared states $|S|^2$ and Y-axis is the solution time. The interval is obtained by different initial states of the robot system.

TABLE II

Comparison With MILP

Formulas	$\|AP\|$	$\|S\|$	MILP(/s)	PDT(/s)
$Fap_1 \wedge Fap_2$	2	4	6.65	0.00661
$Fap_2 \wedge Fap_3 \wedge (\neg ap_2 \wedge \neg ap_3) U (ap_1)$	3	5	18.9	0.00940
$Fap_2 \wedge Fap_3 \wedge (\neg ap_3) U (ap_1)$	3	6	15.3	0.0127
$Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge (\neg ap_3) U (ap_1 \vee ap_2)$	3	7	16.5	0.0167
$Fap_1 \wedge Fap_2 \wedge Fap_3$	3	8	12.3	0.0225
$Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4$	4	16	25.4	0.0568
$Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4 \wedge Fap_5$	5	32	59.9	0.141

RRT are applied and the step size is set as $\frac{1}{40}$ of the map size. The solution time is listed in Table I and Fig. 6(b). It indicates that the solution time is approximately linearly proportional to $|S|^2$, which is smaller than the upper bound of time complexity $O(n^3)$ in Sec. V-B.

Since MILP is a SOTA method for temporal logic with time constraints, our approach is further compared with the MILP method proposed in [48], [49], and [50]. We consider an environment with 5 areas of interest related to 5 atomic propositions and various formulas with different numbers of NBA states. As MILP is applied to optimize the discretized workspace and time without explicit execution time of each sub-task, the tested time constraints are set as integer which can be discretized and the relative-time attributes will not be considered. The time constraints are set as $TI(ap_2) = (0, 11, 0)$, $TI(ap_i) = (0, \infty, 0)$, $i = \{1, 3, 4, 5\}$. As reported in Table II, PDT outperforms MILP in terms of the solution time.

As graph search methods cannot consider the time interval, we only compare the proposed method with the graph search methods in the environment without time windows constraints. To apply graph search in dynamic environments, TS has to be updated by the current positions of all areas of interest at each time stamp. Then, the robot will target the first area of interest in the task planning and update the current NBA state to initialize the next graph search until the first suffix

TABLE III
COMPARISON WITH GRAPH SEARCH

Formulas	$\|AP\|$	$\|S\|$	Graph(/s)	PDT(/s)
$Fap_1 \wedge Fap_2$	2	4	0.112	0.00757
$Fap_2 \wedge Fap_3 \wedge (\neg ap_2 \wedge \neg ap_3) U (ap_1)$	3	5	0.521	0.00933
$Fap_2 \wedge Fap_3 \wedge (\neg ap_3) U (ap_1)$	3	6	0.664	0.0132
$Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge (\neg ap_3) U (ap_1 \vee ap_2)$	3	7	0.861	0.0175
$Fap_1 \wedge Fap_2 \wedge Fap_3$	3	8	0.932	0.0229
$Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4$	4	16	5.52	0.0609
$Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4 \wedge Fap_5$	5	32	25.9	0.151

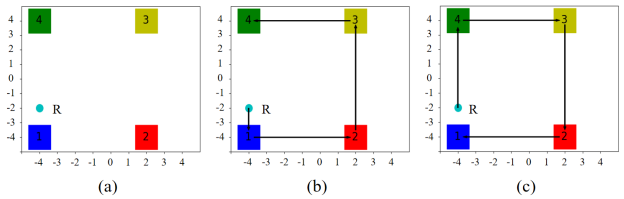


Fig. 7. Performance of order attributes. (a) The environment consists of 4 areas of interest (i.e., color blocks) and a robot (i.e., the cyan dot). (b) The visit sequence without order attributes. (c) The visit sequence with specified order attributes.

stage has been finished. There are five areas of interest ap_i, $i \in [5]$ and their related areas are defined as $LC(ap_1, t) = (5, 5)$; $LC(ap_2, t) = (5 + \sin(2\pi\omega t), 1)$; $LC(ap_3, t) = (5 + \sin(2\pi\omega t + \pi/3), 9)$; $LC(ap_4, t) = (1, 5 + \cos(2\pi\omega t))$; $LC(ap_5, t) = (9, 5 + \cos(2\pi\omega t + \pi/3))$, where ω is the frequency. The velocity is set as $v = 0.2$, the frequency is set as $\omega = 1/12$, and the initial position is set as $p_0 = (6, 5)$. As summarized in Table III, with more propositions and tasks, the solution time of graph search increases significantly. Besides, with the increase of frequency ω, the short-term goals of the robot are prone to change, resulting in an uncertain solution time.

B. Semantics of E-pTL

To show the semantics of E-pTL, the following investigates the capability of E-pTL in describing the order attributes, time attributes, relative-time attributes, and time-varying attributes of atomic propositions.

1) Order Attributes: The environment in Fig. 7(a) consists of 4 areas of interest. Suppose the E-pTL formula is $\phi = Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4$, $LC(ap_1) = (-4, 4)$, $LC(ap_2) = (2, 4)$, $LC(ap_3) = (2, -4)$, $LC(ap_4) = (-4, -4)$. Without order attributes, which indicates that $\forall i = \{1, 2, 3, 4\}$, $AF(ap_i) = BF(ap_i) = \emptyset$, then $ap_1 ap_2 ap_3 ap_4$ will be visited sequentially as shown in Fig. 7(b). If additional order constraints are considered, e.g., ap_1 should be visited after ap_2, we can leverage the order attributes of E-pTL and define $AF(ap_2) = \{ap_1\}$ or $BF(ap_1) = \{ap_2\}$. Thus, without modifying ϕ, a valid path will be in the order of $ap_4 ap_3 ap_2 ap_1$, as shown in Fig. 7(c).

2) Time Attributes: Given the environment in Fig. 8(a), suppose the LTL formula is $\phi = F(ap_1 \wedge (F(ap_2 \wedge (Fap_4)))) \vee F(ap_1 \wedge (F(ap_3 \wedge (Fap_4))))$. Without considering time constraints, the traveling distance will be used as the cost in path planning, resulting in a path visiting ap_1, ap_2, ap_4 sequentially, as shown in Fig. 8(b). If time constraints are considered, e.g., ap_2 and ap_3 should be executed within the time interval of $TI(ap_2) = (0, 50, 0)$ and $TI(ap_3) = (0, 100, 0)$, respectively, the generated plan using PDT is to visit $ap_1 ap_3 ap_4$ sequentially in Fig. 8(c).

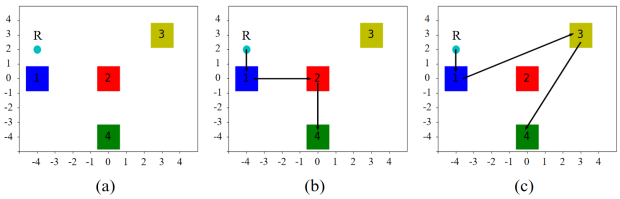


Fig. 8. Performance of time attributes. (a) The considered environment. (b) The visit sequence without time attributes. (c) The visit sequence with specified time attributes.

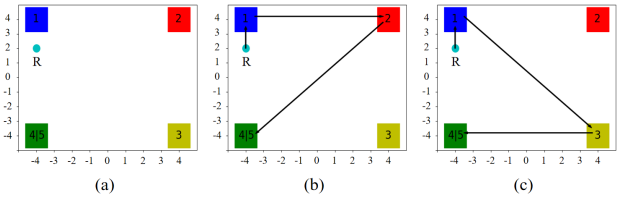


Fig. 9. Performance of relative-time attributes. (a) The considered environment. (b) The visit sequence without relative-time attributes. (c) The visit sequence with specified relative-time attributes.

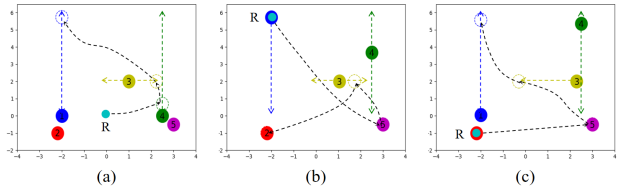


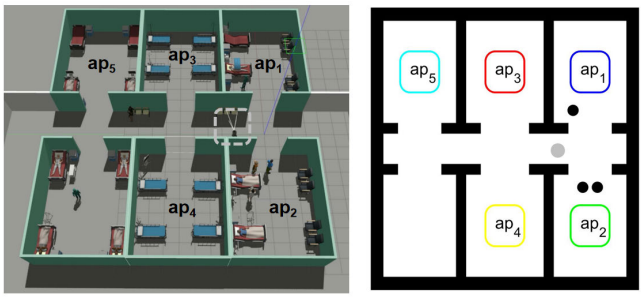
Fig. 10. Performance of real time framework. (a) The prefix plan starts at $t = 0$. (b) The first suffix plan start at $t = 57.3$. (c) The second suffix plan starts at $t = 121.1$.

3) Relative-Time Attributes: Given the environment in Fig. 9(a), suppose the LTL formula is $\phi = F(ap_1 \wedge (F(ap_2 \wedge (Fap_4)))) \vee F(ap_1 \wedge (F(ap_3 \wedge (Fap_5))))$, where ap_4 and ap_5 are coupled in the same position. Without considering relative-time constraints, ap_1, ap_2, ap_4 will be visited sequentially as shown in Fig. 9(b). If there exists relative-time constraints, e.g., ap_4 should be completed 80s after the completion of ap_2 and ap_5 should be completed 80s after the completion of ap_3, then there exist $TI(ap_4) = (0, 80, t_r(ap_2))$ and $TI(ap_5) = (0, 80, t_r(ap_3))$. As ap_3 is closer to ap_5, which indicates that the robot has a shorter traveling time from ap_3 to ap_5, the plan generated by PDT is then to visit $ap_1 ap_3 ap_5$, as shown in Fig. 9(c).

4) Time-Varying Attributes: Consider an environment as shown in Fig. 10(a) and suppose the E-pTL formula is $\phi = GF(ap_1 \vee ap_2) \wedge GF(ap_3) \wedge GF(ap_4 \vee ap_5)$. The space constraints are $LC(ap_1, t) = (-2, 3 - 3\cos(\frac{5\pi t}{100}))$, $LC(ap_2, t) = (-2.2, -1)$, $LC(ap_3, t) = (1, 1 + 1.3\sin(\frac{7\pi t}{100}))$, $LC(ap_4, t) = (2.5, 3 - 3\cos(\frac{\pi t}{100}))$, $LC(ap_5, t) = (3, -0.5)$. Then, the initial plan can be obtained as $\boldsymbol{\pi}_{pre} = ap_4 ap_3 ap_1$, $\boldsymbol{\pi}^1_{suf} = ap_5 ap_3 ap_2$. As the environment is changing, the new suffix plan $\boldsymbol{\pi}^2_{suf} = ap_5 ap_3 ap_1$ can be obtained after completing the first suffix plan by Alg 6. The results are shown in Fig. 10.

C. Numerical Experiments

A simulated hospital environment is established in Gazebo as shown in Fig. 11. Consider a mission $\phi = F(ap_1 \vee ap_2) \wedge Fap_3 \wedge Fap_4 \wedge Fap_5$, where ap_1 and ap_2 indicate the visit of the doctor's office 1 and 2, respectively, ap_3 and ap_4 indicate the visit of the general ward 1 and 2, respectively, and



(a) (b)

Fig. 11. (a) The hospital environment created in Gazebo. (b) The corresponding abstract map, where ap_1 and ap_2 represent the doctor's offices, ap_3 and ap_4 represent the wards, and ap_5 represents the ICU. The gray dot represents the robot and the black dots represent the obstacles.

ap_5 indicates the visit to the ICU. The mission ϕ requires the robot to arrive at the ICU, the general wards 1 and 2, and one of the doctor's offices. There are order constraints, i.e., the robot should arrive at the ICU before going to the office, and time constraints, i.e., the robot should arrive at the ICU within 100 seconds.

According to these constraints, the attributes of atomic propositions are defined as:

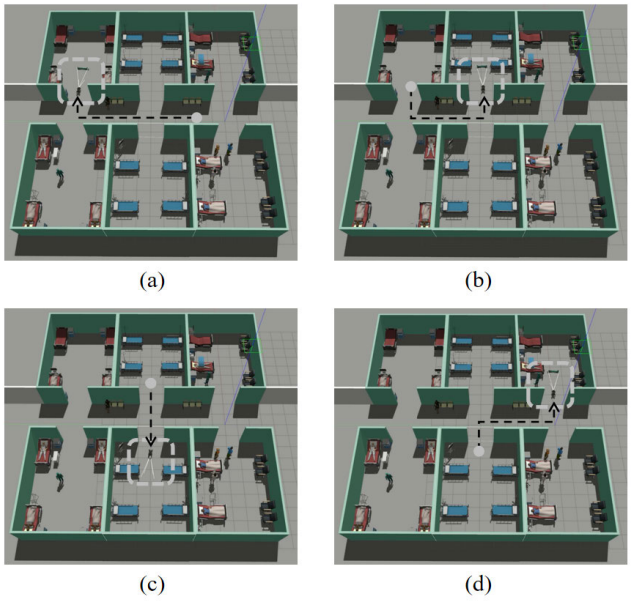
$$\begin{aligned} BF(ap_3) &= BF(ap_4) = BF(ap_5) = \emptyset, \\ BF(ap_1) &= BF(ap_2) = \{ap_5\}, \\ AF(ap_i) &= \emptyset,\ i \in \{1,2,3,4,5\}, \\ TI(ap_i) &= (0, t_{max}, 0),\ i \in \{1,2,3,4\}, \\ TI(ap_5) &= (0, 100, 0), \end{aligned}$$

which gives rise to the E-pTL formula.

Since ϕ is a finite plan, we only need to determine the prefix stage. Applying PDT, a tree with 34 nodes is constructed within $0.0908s$ and the generated plan is $o = ap_5 ap_3 ap_4 ap_1$ as shown in Fig. 12. Since robot should reach the ICU within 100 seconds, it will first go to the ICU. Then, the robot will go to the general ward 1 and 2 according to the cost of path. Finally, the robot choose the office 1 because the path to the office 1 has less obstacles and can be reached much faster. The simulation video is provided.[1]

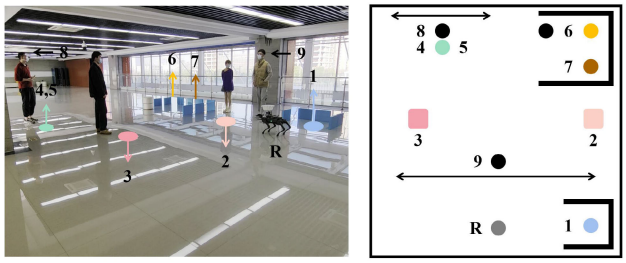
D. Physical Experiments

To evaluate the efficiency of our approach in dynamic environments, physical experiments are carried out in this section. As shown in Fig. 13, the environment contains an office (i.e., position 1) that assigns tasks to the robot, two distribution areas (i.e., position 2 and 3) that distribute packages to the robot, two moving barriers (i.e., position 8 and 9), a moving reception area next to the person (position 4 and 5) and two charging stations (i.e., positions 6 and 7). The robot needs to first visit the office to get a task assignment and then visit the distribution area to fetch the desired package and sends it to the reception area within $10s$ after obtaining the package. Finally, the robot goes to one of the charging stations. Such a mission can be expressed as $\phi = Fap_1 \wedge (F(ap_2 \wedge (Fap_4)) \vee F(ap_3 \wedge (Fap_5))) \wedge F(ap_6 \vee ap_7) \wedge G\neg ap_8 \wedge G\neg ap_9$. ap_1 represents the task of going to the office to get a task assignment.



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

Fig. 12. Simulation results. (a) The robot arrives at ICU. (b) The robot arrives at general ward 1. (c) The robot arrives at general ward 2. (d) The robot finally arrives at office 1.



(a) (b)

Fig. 13. (a) The physical environment. (b) The corresponding abstract map, where the thick lines indicate the wall. The black dot next to position 6 is a static obstacle while the black dots at position 8 and 9 indicate moving obstacles with the arrow lines indicating the moving directions. The quadruped robot is represented by the gray dot and the labeled dots and squares represent the areas of interest.

ap_2 and ap_3 represent the task of going to the distribution office at positions 2 and 3 to fetch the package, respectively. ap_4 and ap_5 represent the task of delivering the package from the positions 2 and 3 to the reception area, respectively. ap_6 and ap_7 represent the charging stations. ap_8 and ap_9 represent the constraints of not colliding with the static and moving obstacles, respectively. The order attribute is set as $AF(ap_1) = \{ap_2, ap_3\}$, $AF(ap_4) = AF(ap_5) = \{ap_6, ap_7\}$. The relative-time attributes are set as $TI(ap_4) = (0, 10, t_r(ap_2))$ and $TI(ap_5) = (0, 10, t_r(ap_3))$.

By the proposed method, a tree with 41 nodes can be constructed within $0.0922s$ and the generated plan is $ap_1 ap_3 ap_5 ap_7$, because the time from ap_2 to ap_4 is more than $10s$ and the path from ap_5 to ap_6 has more barriers than to ap_7. As shown in Fig. 14, the robot first goes to the office and then goes to the distribution office 2 at position 3 while avoiding the moving person. Then, it delivers the package to the person within $10s$ and finally arrive at charging station 2 at position 7. The experiment video is provided.[2]

[1] https://youtu.be/N6MrmPr_Z08

[2] https://youtu.be/6lmVG_UEgGs

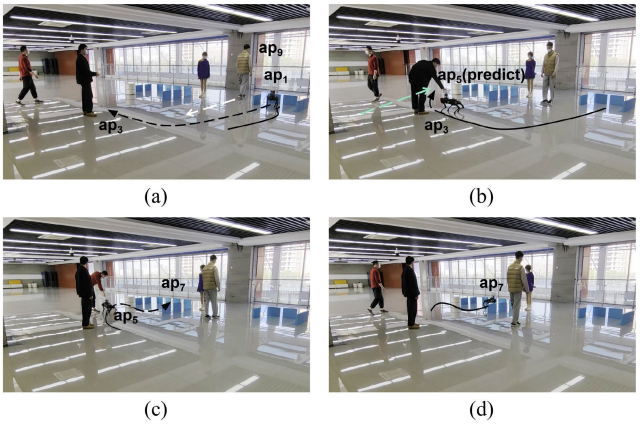


Fig. 14. Experiment results. The dashed lines indicate the robot trajectory. In (a), the robot goes to the office for task assignment, i.e., ap_1. In (b), the robot fetches the package in the distribution area 2 at position 3 to execute ap_3 while avoiding the moving person. In (c), the robot delivers the package to the moving reception area to execute ap_5. In (d), the robot goes to the charging station 2 at position 7 to execute ap_7.

VII. Conclusion

In this paper, the extended predicate-based temporal logic (E-pTL) formula is developed to incorporate the space, time, and order constraints. Based on E-pTL, the planning decision tree (PDT) is then developed for fast motion planning in dynamic environments. Additional research will consider extending PDT with deep learning methods. For instance, the modular deep learning network [9] can be leveraged to provide interpretable evaluation information to facilitate decision-making in more complex tasks. Besides, more semantic expressions of E-pTL will be explored for complex tasks and systems, such as heterogeneous robot systems.

References

[1] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta, "Automatic deployment of robotic teams," *IEEE Robot. Autom. Mag.*, vol. 18, no. 3, pp. 75–86, Sep. 2011.

[2] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion," *IEEE Robot. Autom. Mag.*, vol. 14, no. 1, pp. 61–70, Mar. 2007.

[3] Z. Li, G. Li, X. Wu, Z. Kan, H. Su, and Y. Liu, "Asymmetric cooperation control of dual-arm exoskeletons using human collaborative manipulation models," *IEEE Trans. Cybern.*, vol. 52, no. 11, pp. 12126–12139, Nov. 2022.

[4] M. Wells, Z. Kingston, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Finite-horizon synthesis for probabilistic manipulation domains," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 6336–6342.

[5] M. Berg, G. Konidaris, and S. Tellex, "Using language to generate state abstractions for long-range planning in outdoor environments," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 1888–1895.

[6] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Trans. Robot.*, vol. 38, no. 6, pp. 3602–3621, Dec. 2022.

[7] G. F. Schuppe and J. Tumova, "Multi-agent strategy synthesis for LTL specifications through assumption composition," in *Proc. IEEE 16th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2020, pp. 533–540.

[8] L. Li, Z. Chen, H. Wang, and Z. Kan, "Fast task allocation of heterogeneous robots with temporal logic and inter-task constraints," *IEEE Robot. Autom. Lett.*, vol. 8, no. 8, pp. 4991–4998, Aug. 2023.

[9] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan, "Modular deep reinforcement learning for continuous motion planning with temporal logic," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7973–7980, Oct. 2021.

[10] P. Yu and D. V. Dimarogonas, "Distributed motion coordination for multirobot systems under LTL specifications," *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 1047–1062, Apr. 2022.

[11] Y. Kantaros, M. Malencia, V. Kumar, and G. J. Pappas, "Reactive temporal logic planning for multiple robots in unknown environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 11479–11485.

[12] Z. Zhou, Z. Chen, M. Cai, Z. Li, Z. Kan, and C.-Y. Su, "Vision-based reactive temporal logic motion planning for quadruped robots in unstructured dynamic environments," *IEEE Trans. Ind. Electron.*, vol. 71, no. 6, pp. 5983–5992, Jun. 2024.

[13] Y. Kantaros, S. Kalluraya, Q. Jin, and G. J. Pappas, "Perception-based temporal logic planning in uncertain semantic maps," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2536–2556, Aug. 2022.

[14] M. Cai, S. Xiao, Z. Li, and Z. Kan, "Optimal probabilistic motion planning with potential infeasible LTL constraints," *IEEE Trans. Autom. Control.*, vol. 68, no. 1, pp. 301–316, Jan. 2023.

[15] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Berlin, Germany: Springer, 2004, pp. 152–166.

[16] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Syst.*, vol. 2, no. 4, pp. 255–299, Nov. 1990.

[17] S. Ahlberg and D. V. Dimarogonas, "Human-in-the-loop control synthesis for multi-agent systems under hard and soft metric interval temporal logic specifications," in *Proc. IEEE 15th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2019, pp. 788–793.

[18] Z. Xu, S. Saha, B. Hu, S. Mishra, and A. A. Julius, "Advisory temporal logic inference and controller design for semiautonomous robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 1, pp. 459–477, Jan. 2019.

[19] X. Li et al., "Vehicle trajectory prediction using generative adversarial network with temporal logic syntax tree features," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3459–3466, Apr. 2021.

[20] J. Karlsson, S. van Waveren, C. Pek, I. Torre, I. Leite, and J. Tumova, "Encoding human driving styles in motion planning for autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 1050–1056.

[21] S. Coogan, E. A. Gol, M. Arcak, and C. Belta, "Traffic network control from temporal logic specifications," *IEEE Trans. Control Netw. Syst.*, vol. 3, no. 2, pp. 162–172, Jun. 2016.

[22] C. E. Tuncali, G. Fainekos, D. Prokhorov, H. Ito, and J. Kapinski, "Requirements-driven test generation for autonomous vehicles with machine learning components," *IEEE Trans. Intell. Vehicles*, vol. 5, no. 2, pp. 265–280, Jun. 2020.

[23] N. Arechiga, "Specifying safety of autonomous vehicles in signal temporal logic," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 58–63.

[24] A. T. Buyukkocak, D. Aksaray, and Y. Yazıcioğlu, "Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1375–1382, Apr. 2021.

[25] Z. Xu and A. A. Julius, "Census signal temporal logic inference for multiagent group behavior analysis," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 1, pp. 264–277, Jan. 2018.

[26] X. Li, Y. Ma, and C. Belta, "A policy search method for temporal logic specified reinforcement learning tasks," in *Proc. Annu. Amer. Control Conf. (ACC)*, Jun. 2018, pp. 240–245.

[27] R. Alur, T. Feder, and T. A. Henzinger, "The benefits of relaxing punctuality," *J. ACM*, vol. 43, no. 1, pp. 116–146, 1996.

[28] C.-I. Vasile, D. Aksaray, and C. Belta, "Time window temporal logic," *Theor. Comput. Sci.*, vol. 691, pp. 27–54, Aug. 2017.

[29] E. Bonnah, L. Nguyen, and K. Anuarul Hoque, "Motion planning using hyperproperties for time window temporal logic," *IEEE Robot. Autom. Lett.*, vol. 8, no. 8, pp. 4386–4393, Aug. 2023.

[30] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Sci. Robot.*, vol. 4, no. 37, Dec. 2019, Art. no. eaay6276.

[31] M. Guo and M. Zavlanos, "Probabilistic motion planning under temporal tasks and soft constraints," *IEEE Trans. Autom. Control*, vol. 63, no. 12, pp. 4051–4066, Dec. 2018.

[32] D. Maity and J. S. Baras, "Motion planning in dynamic environments with bounded time temporal logic specifications," in *Proc. 23rd Medit. Conf. Control Autom. (MED)*, Jun. 2015, pp. 940–946.

[33] S. Kalluraya, G. J. Pappas, and Y. Kantaros, "Multi-robot mission planning in dynamic semantic environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 1630–1637.

[34] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Trans. Robot.*, vol. 26, no. 1, pp. 48–61, Feb. 2010.

[35] S. L. Smith, J. Tumová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *Int. J. Robot. Res.*, vol. 30, no. 14, pp. 1695–1708, Dec. 2011.
[36] Y. E. Sahin, R. Quirynen, and S. D. Cairano, "Autonomous vehicle decision-making and monitoring based on signal temporal logic and mixed-integer programming," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2020, pp. 454–459.
[37] R. Yan and A. Julius, "A decentralized B&B algorithm for motion planning of robot swarms with temporal logic specifications," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7389–7396, Oct. 2021.
[38] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *Int. J. Robot. Res.*, vol. 39, no. 8, pp. 1002–1028, Jul. 2020.
[39] Y. Kantaros and M. M. Zavlanos, "STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 812–836, Jun. 2020.
[40] D. Gundana and H. Kress-Gazit, "Event-based signal temporal logic synthesis for single and multi-robot tasks," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3687–3694, Apr. 2021.
[41] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.
[42] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Proc. IEEE Symp. Log. Comput. Sci.*, Sep. 1986, pp. 322–331.
[43] P. Gastin and D. Oddoux, "Fast LTL to büchi automata translation," in *Proc. Int. Conf. Comput. Aided Verif.* Cham, Switzerland: Springer, 2001, pp. 53–65.
[44] Z. Liu, B. Wu, J. Dai, and H. Lin, "Distributed communication-aware motion planning for multi-agent systems from STL and SpaTeL specifications," in *Proc. IEEE Conf. Decis. Control*, Sep. 2017, pp. 4452–4457.
[45] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.
[46] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 4817–4822.
[47] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Autom. Control*, vol. 57, no. 11, pp. 2817–2830, Nov. 2012.
[48] C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scLTL motion planning for mobility-on-demand," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 1481–1488.
[49] K. Leahy et al., "Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS)," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2516–2535, Aug. 2022.
[50] K. Nagae and T. Ushio, "Extension of counting LTL and its application to a path planning problem for heterogeneous multi-robot systems," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E107.A, no. 5, pp. 752–761, May 2024.

Ziyang Chen received the B.E. degree from Beihang University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree in control science and engineering with the University of Science and Technology of China, Hefei, China. His current research interests include robotics and multi-agent systems.

Mingyu Cai received the Ph.D. degree in mechanical engineering from The University of Iowa, Iowa City, IA, USA, in 2021. From 2021 to 2023, he was a Post-Doctoral Associate with the Department of Mechanical Engineering, Lehigh University, Bethlehem, PA, USA. He was with the Honda Research Institute as a Research Scientist. He is currently an Assistant Professor with the University of California at Riverside. His research interests include robotics, machine learning, control theory, and formal methods, with a focus on applications in motion planning, decision-making, nonlinear control, and human–robot interaction.

Zhangli Zhou (Graduate Student Member, IEEE) received the B.E. degree from Wuhan University of Technology, Wuhan, China, in 2016. He is currently pursuing the Ph.D. degree with the University of Science and Technology of China, Hefei, China. His research interests include robotics and reactive task and motion planning.

Lin Li (Graduate Student Member, IEEE) received the B.E. degree in mechanical engineering and automation from Hefei University of Technology, Hefei, China, in 2021. She is currently pursuing the Ph.D. degree in control engineering with the University of Science and Technology of China, Hefei. Her current research interests include multi-robot systems and formal methods.

Zhen Kan (Senior Member, IEEE) received the Ph.D. degree from the Department of Mechanical and Aerospace Engineering, University of Florida, in 2011. He was a Post-Doctoral Research Fellow with the Air Force Research Laboratory (AFRL), Eglin AFB, and the University of Florida REEF from 2012 to 2016, and was an Assistant Professor with the Department of Mechanical Engineering, The University of Iowa, from 2016 to 2019. He is currently a Professor with the Department of Automation, University of Science and Technology of China. His research interests include networked control systems, nonlinear control, formal methods, and robotics. He serves on program committees at several internationally recognized scientific and engineering conferences. He is an Associate Editor of IEEE TRANSACTIONS ON AUTOMATIC CONTROL.