Temporal Logic Task Planning for Autonomous Systems With Active Acquisition of Information

Shuaiyi Li, Mengjie Wei, Shaoyuan Li, *Senior Member, IEEE*, and Xiang Yin, *Member, IEEE*

***Abstract*—High-level task planning is one of the central problems in autonomous systems such as unmanned ground vehicles (UGV). In this context, the agent makes decisions online to ensure the satisfaction of complex tasks under dynamic environment. In practice, control decisions are made based on the information acquired by sensors that can be turned on/off. Therefore, in the task planning problem, one needs to synthesize control and sensing decisions jointly in order to achieve the tasks. In this paper, we formulate and solve a *control-sensing co-synthesis* problem for linear temporal logic (LTL) tasks. The objective is to synthesize an active-sensing controller such that a given LTL formula accepted by a deterministic Büchi automaton can always be satisfied with a provably correct formal guarantee. To solve this problem, we propose a new approach integrating offline computations with online executions. Based on the winning regions pre-computed offline, the autonomous system can generate both control and sensing decisions online to drive the system. We show that the proposed approach is both sound and complete. The scalability and effectiveness of the proposed method are evaluated by both numerical experiments and hardware implementations in UGV task planning.**

***Index Terms*—Autonomous systems, formal methods, task planning, UGVs.**

I. Introduction

A. Motivation

AUTONOMOUS systems, including unmanned ground vehicles (UGVs) [1] and unmanned aerial vehicles (UAVs) [2], have garnered increasing attentions in both academia and industry over the past years, owing to their successful application in diverse fields such as smart manufacturing, environmental surveillance, and beyond. These systems have demonstrated the capacity to operate intelligently and independently. Consequently, researchers have directed their attention toward addressing two critical problems in the development of autonomous systems: motion planning and task planning. In the context of motion planning, the primary objective is to find feasible trajectories for the vehicle to navigate, which has been investigated very extensively in the literature [3], [4], [5], [6]. Meanwhile, *high-level task planning* for autonomous systems has gained considerable interest more recently as it enables sophisticated decision-making and advanced functionalities in intelligent vehicles.

Linear temporal logic (LTL) offers a highly expressive and user-friendly approach for specifying high-level formal requirements and has found extensive application in autonomous system task planning [7]. Through the inductive use of temporal and Boolean operators, it enables the precise description of complex formal tasks, such as *visiting two target regions infinitely often while avoiding reaching some dangerous regions*. In recent years, LTL-based high-level task planning has been studied very extensively and has demonstrated successful implementation in UGVs [8] and UAVs [9]. For example, in [10], [11], [12], LTL task planning for multi-agent systems is investigated. In [13], [14], [15], Petri nets are used to mitigate the computational complexity in LTL planning problem. In [16], [17], the authors use reinforcement learning to solve the LTL task planning problem for unknown environments.

In the task planning problem, the autonomous system must make informed decisions based on its own information, acquired through sensors that can be dynamically activated or deactivated online. For instance, consider a UGV entering an unexplored region. It needs to intelligently choose whether to activate a camera to perceive its surroundings or activate the GPS to gather essential data for determining appropriate control actions. This integrated approach, combining both sensing and control strategies, is known as an *active-sensing control*. Therefore, for LTL planning in a general setting with information uncertainty, the challenge lies in effectively solving the *joint synthesis* of both sensing and control strategies, ensuring that the agent fulfills its tasks optimally.

B. Our Results

In this work, we formulate and solve a *control-sensing co-synthesis* problem for linear temporal logic (LTL) tasks. Specifically, we consider an active-sensing control mechanism, where the autonomous system needs to determine sensing decisions (what to observe) and control decisions (what to execute) alternatively based on the real-time information received on-the-fly. The objective is to synthesize an active-sensing controller such that a formal task requirement specified by a given LTL formula can always be satisfied despite transition non-determinism and information uncertainty. Particularly, we consider a fragment of LTL formulae that can be accepted by deterministic Büchi

Manuscript received 10 August 2023; revised 22 September 2023 and 16 October 2023; accepted 21 October 2023. Date of publication 24 October 2023; date of current version 23 February 2024. This work was supported by the National Natural Science Foundation of China under Grants 62061136004, 61803259, and 61833012. *(Corresponding author: Xiang Yin.)*

The authors are with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: lishuaiyi@sjtu.edu.cn; mj.wei@sjtu.edu.cn; syli@sjtu.edu.cn; yinxiang@sjtu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIV.2023.3327312.

Digital Object Identifier 10.1109/TIV.2023.3327312

automata (DBA). This fragment is restrictive but still expressive enough to capture many task requirements.

To solve this problem, we propose a new approach integrating offline computations with online executions. Specifically, at the offline computation stage, we first construct the belief transition system that captures the knowledge evolution of the controller through sensing actions and control inputs. Then starting from each state in the belief transition system, we augment additional information regarding the satisfaction of the tasks and iteratively compute the winning region. Based on the winning region computed offline, at the online stage, we propose an algorithm that effectively determines both sensing and control decisions on-the-fly. We prove that the proposed algorithm is both sound and complete.

The worst-case complexity for the offline computation is exponential in the number of system states. However, it is well-known that such an exponential complexity is unavoidable for synthesis problems under partial observation. However, our numerical experiments show that the actual performance of our algorithm is much better than the exponential upper-bound. Furthermore, once the offline computation is done, the online execution for each step only requires a simple search over the computed structure. Finally, we have also implemented our method in a real-world UGV task planning system to demonstrate its effectiveness.

C. Related Works

Controller synthesis under partial observations has been extensively studied in the context of discrete event systems (DES). Notably, various algorithms have been proposed in [18], [19], [20], [21] for synthesizing partial-observation supervisory controllers to fulfill non-blockingness requirements. Additionally, in [22], [23], [24], sensor activation algorithms are developed to acquire sufficient information for control or diagnosis purposes. However, it is important to note that the non-blockingness requirement in DES literature is different from the LTL task considered in our work. The former represents a branching time property, while the latter is a linear time property [25]. Moreover, existing works in DES literature solely focus on the synthesis of control policies or sensing policies *independently*, and to the best of our knowledge, no investigation has been made into the control-sensing co-synthesis problem.

Our work is also closely related to the field of reactive controller synthesis with game graphs under imperfect information, as demonstrated in [26], [27], [28]. Specifically, when active sensing is not considered, i.e., the observation capability is fixed, our problem can be reduced to an ω-regular objective synthesis problem with imperfect information. Previous work by [29] has made partial progress in addressing this type of problem. However, a significant difference exists between our work and [29]. In [29], the assumption is made that all states with the same observation possess the same properties, allowing the application of existing full observation control synthesis algorithms to the partial observation setting. In contrast, our work does not make this restrictive assumption. As a result, novel techniques are required to effectively solve our problem.

In the context of LTL task planning for autonomous systems, there have been plentiful works investigating the scalable planning methods under perfect information; see, e.g. [30], [31], [32]. However, in our work, we consider the case of imperfect information and agent needs to react to the environment based on its observation. Recently, some studies have focused on addressing the planning problem in scenarios where the environment is either unknown or only partially known, requiring active sensing capabilities; see, e.g., [33], [34], [35], [36], [37]. Among these, [34] bears the closest relevance to our work. Particularly, [34] investigates the control-sensing co-synthesis problem for LTL specifications accepted by DBA. A key advantage of [34] lies in its efficient approach to synthesizing a winning strategy without the need to construct the entire belief space. However, it should be noted that the algorithm proposed in [34] may not guarantee completeness, as it can potentially lead to situations where the autonomous system becomes blocked, and no further control actions can be taken. On the contrary, our algorithm is both sound and complete, providing strong guarantees that no such blocked states can be reached when achieving the LTL tasks.

D. Organization

The paper is structured as follows. Section II provides a brief introduction to some necessary preliminaries. In Section III, we formulate the LTL task planning problem with active information acquisition. In Section IV, we construct the belief transition system and discuss the difficulties involved in this problem. Our main solution is presented in Section V, which involves both offline computation and online execution. In Section VI, we present illustrative simulations, numerical experiments, and hardware experiments in UGV planning. Finally, we conclude the paper in Section VII.

II. Preliminary

A. System Model

The mobility of the autonomous system (or agent) is modeled as a *labeled transition system* (LTS), which is a 6-tuple

$$\mathcal{T} = \langle X, Act, \Delta, X_0, \mathcal{AP}, L \rangle,$$

where

- X is a finite set of system states representing the physical locations of the agent;
- Act is a finite set of control actions or controller inputs;
- $\Delta : X \times Act \to 2^X$ is the non-deterministic transition function, where $x' \in \Delta(x, a)$ means that the system may move from state x to x' by taking action $a \in Act$;
- $X_0 \subseteq X$ is the set of possible initial states;
- $\mathcal{AP}$ is the set of atomic propositions representing some basic properties of interest;
- $L : X \to 2^{\mathcal{AP}}$ is the labeling function that assigns each state a set of atomic propositions, which hold at that state.

Let A be a set. We denote by A^ω and A^* the set of all infinite sequences and finite sequences over A, respectively. For states $x, x' \in X$ and action $a \in Act$, we also write as $x \xrightarrow{a} x'$

if $x' \in \Delta(x,a)$ and as $x \to x'$ if $x \xrightarrow{a} x'$ for some a. Then a *path* generated by LTS $\mathcal{T}$ is an infinite sequence of states $\tau = x_0x_1x_2\cdots \in X^\omega$ such that $x_0 \in X_0$ and $x_i \to x_{i+1}, \forall i \geq 0$. A finite path generated by LTS $\mathcal{T}$ is a finite prefix of some (infinite) path generated by LTS $\mathcal{T}$. We denote by $\mathsf{Path}(\mathcal{T})$ and $\mathsf{Path}^*(\mathcal{T})$ the set of all paths and finite paths generated by $\mathcal{T}$. The *trace* of path $\tau \in \mathsf{Path}(\mathcal{T})$ is $\mathsf{Trace}(\tau) = L(x_0)L(x_1)L(x_2)\cdots \in (2^{\mathcal{AP}})^\omega$.

Remark 1: Here, we make some remarks regarding the LTS model considered in this work. First, since our work focuses on the *high-level task planning problem*, the LTS model is essentially an abstracted mobility model representing the high-level behaviors of the UGV (e.g. from regions to regions). Once a high-level plan is determined by our model, then a low-level hybrid controller can be deployed to navigate in physical world. This two-level architecture is widely used in task planning of UGV so that researchers can better focus on the high-level decision synthesis based on the present discrete (symbolic) LTS models; see, e.g., [10], [37]. Second, we note that the transition function Δ is non-deterministic in general. In practice, transition non-determinism may be due to unknown disturbances or uncontrollable parts in real-world environments. Also, abstraction errors between the high-level model and the low-level dynamics may also lead to transition non-determinism.

B. Linear Temporal Logic

The control objectives of the autonomous system are described by Linear Temporal Logic (LTL) formulae with the following syntax

$$\phi ::= \mathsf{True} \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2,$$

where $a \in \mathcal{AP}$ is an atomic proposition; $\neg$ and $\wedge$ are Boolean operators "negation" and "conjunction", respectively; $\bigcirc$ and $\mathcal{U}$ are temporal operators "next" and "until", respectively. We can also define other temporal or Boolean operators such as "eventually" by $\Diamond\phi := \mathsf{True}\,\mathcal{U}\,\phi$, "always" by $\Box\phi := \neg\Diamond\neg\phi$, "disjunction" by $\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2)$, and "implication" by $\phi_1 \to \phi_2 := \neg\phi_1 \vee \phi_2$.

LTL formulae are evaluated on infinite words (infinite sequences over $2^{\mathcal{AP}}$). For an infinite word $\sigma \in (2^{\mathcal{AP}})^\omega$, we denote by $\sigma \vDash \phi$ if it satisfies LTL formula ϕ. For example, $\sigma \vDash \Box\phi_1$ means that ϕ_1 always holds at any instant along σ, while $\sigma \vDash \Diamond\phi_1$ means that ϕ_1 holds for some instant along σ. The reader is referred to [38] for more details on the semantics of LTL. We denote by $\mathsf{Word}(\phi) = \{\sigma \in (2^{\mathcal{AP}})^\omega : \sigma \vDash \phi\}$ the set of all infinite words satisfying ϕ.

In this work, we consider a fragment of LTL formula ϕ that can be accepted by a *deterministic* Büchi automaton (DBA); see, e.g. [38]. It is important to notice that, this assumption is restrictive since it does not include formulae such as $\Diamond\Box$. However, it is still expressive enough for many practical specifications. Formally, a DBA is a 5-tuple

$$\mathcal{A} = \langle S, \Sigma, \xi, s_0, S_F \rangle,$$

where S is a finite set of states, $\Sigma = 2^{\mathcal{AP}}$ is the finite alphabet, $\xi : S \times \Sigma \to S$ is the deterministic transition function, s_0 is the unique initial state, and $S_F \subseteq S$ is the set of accepting states. For an infinite word $\rho = \sigma_1\sigma_2\cdots \in \Sigma^\omega$, the induced infinite run is the unique infinite sequence of states $\pi = s_0s_1\cdots \in S^\omega$ in $\mathcal{A}$ such that $s_{i+1} = \xi(s_i, \sigma_{i+1})$. An infinite run π is said to be accepted if and only if $\mathsf{inf}(\pi) \cap S_F \neq \emptyset$, where $\mathsf{inf}(\pi)$ is the set of states that occurs infinite times in π. An infinite word ρ is said to be accepted if its induced run is accepted. Given an LTL formula ϕ, we denote by $\mathcal{A}_\phi$ the DBA such that $\mathcal{L}(\mathcal{A}_\phi) = \mathsf{Word}(\phi)$, where $\mathcal{L}(\mathcal{A}_\phi)$ is the set of all accepted words of $\mathcal{A}_\phi$.

III. Problem Formulation

A. Controllers With Active Acquisition of Information

In many cases, the movement of the autonomous system may generate some undesirable behaviors that violate the LTL task. Therefore, a controller is designed to plan for the system so that its behavior will satisfy the specification. In practice, the controller may not always be able to access the full state information of the system or the environment. In this paper, we consider an active information acquisition setting, where the controller needs to take *sensing action* to obtain additional information.

Formally, the sensing module is described by a function

$$\mathcal{O} : X \times Sens \to Obs,$$

where $Sens$ is a finite set of *sensing actions* and Obs is a finite set of *observation symbols*. Intuitively, for any $x \in X$ and $\gamma \in Sens$, $o = \mathcal{O}(x,\gamma) \in Obs$ means that if the agent takes sensing action γ at state x, then it will observe symbol o. Therefore, for each pair of sensing action $\gamma \in Sens$ and observation $o \in Obs$, they induce an equivalent class on X defined by

$$[X]_{\gamma,o} = \{x \in X \mid \mathcal{O}(x,\gamma) = o\}.$$

Note that the observation function is independent from the labeling function L in general. The former captures the external measurements of the system, while the latter captures the internal properties of our interest.

The above defined sensing module is very general and subsumes many existing observation models. For example, for the setting of *static/passive partial observation*, the agent can always obtain some information at each state without taking sensing actions. In this case, $Sens$ is a singleton, or can be omitted as $\mathcal{O} : X \to Obs$. Furthermore, when $Obs = X$ and $\mathcal{O}$ is a bijection, the static observation further reduces to the case of full state observation, i.e., the agent always knows its current location precisely.

Remark 2: It is worth remarking that, in our sensing model, a *sensing action* $\gamma \in Sens$ does not necessarily correspond to an *actual sensor* physically. Instead, it can be a combinational deployment strategy for multiple sensors. Therefore, at each instant, the UGV will issue a unique sensing action, which may correspond to using multiple sensors. For example, suppose that a UGV is equipped with *three actual sensors* s_0, s_1, s_2: one passive sensor s_0 that is always activated, and two active sensors s_1 and s_2, which can be turned on/off actively online. Then without further restriction on the number of sensors the

UGV can use at each instant, the UGV has *four sensing actions* $Sens = \{\{s_0\}, \{s_0, s_1\}, \{s_0, s_2\}, \{s_0, s_1, s_2\}\}$. Furthermore, in practice, the number of sensors the UGV can use at each instant may be limited due to energy constraints. Such sensor constraint can also be captured by our general sensing model. Specifically, in the above example with three actual sensors s_0, s_1 and s_2, if we further require that "*the UGV can use at most two actual sensors at each instant including the passive sensor*", then the sensing action space will be further constrained to $Sens = \{\{s_0\}, \{s_0, s_1\}, \{s_0, s_2\}\}$, where $\{s_0, s_1, s_2\}$ is no longer a feasible sensing action. If we change our requirement to "*the UGV cannot simultaneously activate sensors* s_0 *and* s_1", then the sensing action space will be further constrained to $Sens = \{\{s_0\}, \{s_0, s_2\}\}$, since $\{s_0, s_1\}$ is also no longer a feasible sensing action. Therefore, in our framework, constraints on physical sensors actually are handled at the modeling stage when constructing the sensing model $\mathcal{O} : X \times Sens \to Obs$. Later on, there is no need for the synthesis algorithm to take these constraints into account anymore since all infeasible actions have been excluded.

Given an active sensing module $\mathcal{O} : X \times Sens \to Obs$, an active-sensing-based controller works as follows:

- Initially, the controller needs to make a sensing decision and observes an initial symbol;
- Based on the first sensing decision and its outcome observation, the controller makes the first control decision and upon which the agent moves to a new state (maybe non-deterministically) according to the transition function;
- Once the movement is complete, the controller needs first to make a new sensing decision at the new state encountered in order to obtain a new observation, and then to update its control decision based on the new observation;
- The above process is repeated indefinitely.

To capture the above active-sensing-based control setting, we define the *history information* of the controller as an alternating sequence of sensing action, observation symbol and control input of the following form

$$\gamma_0 o_0 a_0 \gamma_1 o_1 a_1 \cdots \in (Sens \cdot Obs \cdot Act)^*.$$

Then an active-sensing controller is defined as a tuple

$$C = \langle C_a, C_s \rangle,$$

where

- $C_a : Sens \cdot Obs \cdot (Act \cdot Sens \cdot Obs)^* \to Act$ is the control module that determines the current control input; and
- $C_s : (Sens \cdot Obs \cdot Act)^* \to Sens$ is the active sensing module that determines the current sensing decision.

Then given an active sensing module $\mathcal{O}$ and an active-sensing controller C, we denote by $\mathcal{T}_C$ the closed-loop system under control of C and by $\mathsf{Path}(\mathcal{T}_C)$ the set of all paths generated by $\mathcal{T}_C$. Formally, we say a path $\tau = x_0 x_1 x_2 \cdots$ is generated by $\mathcal{T}_C$ with history information $\gamma_0 o_0 a_0 \gamma_1 o_1 a_1 \cdots \in (Sens \cdot Obs \cdot Act)^\omega$, if for any $i = 0, 1, \ldots$, we have (i) $x_{i+1} \in \Delta(x_i, a_i)$; (ii) $o_i = \mathcal{O}(x_i, \gamma_i)$; (iii) $\gamma_i = C_s(\gamma_0 \ldots a_{i-1})$; and (iv) $a_i = C_a(\gamma_0 \ldots o_i)$.

B. Problem Formulation

Since system $\mathcal{T}$ is non-deterministic, the actual movement of the UGV at each instant is determined by both the control input and the environment, which is not perfectly known by the UGV due to the imperfect observation. Specifically, at each state $x \in X$, the controller chooses an action $a \in Act$ that is defined at x, and the environment determines which successor $x' \in \Delta(x, a)$ the agent will actually move into. We assume that the UGV knows its mobility perfectly. That is, it knows the possibility of transitions at each state as well as the labeling function. However, since there is no prior assumption on the environment's behavior, the controller needs react to the environment based on its observation in order to guarantee the accomplishment of the LTL task. Therefore, our objective is to synthesize an active-sensing controller that determines both the control input and the sensing decision at each time instant such that a given LTL task is fulfilled *for sure* for all possible traces. Formally, we aim to solve the following problem.

Problem 1: Given labeled transition system $\mathcal{T}$, observation function $\mathcal{O}$ and LTL formula ϕ, synthesize an active-sensing controller $C = \langle C_a, C_s \rangle$ such that $\mathsf{Trace}(\mathcal{T}_C) \subseteq \mathsf{Word}(\phi)$.

Remark 3: The above formulated problem is essentially a *robust control synthesis problem* in the sense we require that the LTL task is fulfilled for any possible traces in $\mathsf{Trace}(\mathcal{T}_C)$. The motivation for this robust synthesis setting is twofold. First, since we do not have the information of probability distribution on transitions, Δ is purely non-deterministic and therefore, one has to consider all possible outcomes under control. Second, in many applications, either the adversarial nature of the environment or the inherent safety-critical requirement of the system also leads to the robust control setting by requiring all traces are satisfactory.

IV. Belief-Based Information Structures and Challenges

In this section, we first define the belief transition system, which serves as the basis of our solution. Then we discuss some particular challenges in our problem.

A. Product Systems

In order to incorporate the completion status of LTL task ϕ into the labeled transition system $\mathcal{T}$, we first define the *product system* between $\mathcal{T}$ and A_ϕ.

Definition 1 (Product Systems): Given LTS $\mathcal{T} = \langle X, Act, \Delta, X_0, \mathcal{AP}, L \rangle$ and DBA $\mathcal{A}_\phi = \langle S, \Sigma, \xi, s_0, S_F \rangle$ that accepts all words satisfying ϕ, the product system is a new LTS

$$\tilde{T} = \mathcal{T} \times \mathcal{A}_\phi = \left(\tilde{X}, Act, \tilde{\Delta}, \tilde{X}_0, \mathcal{AP}, \tilde{L} \right),$$

where $\tilde{X} = X \times S$ is the set of (product) states, Act is the same set of control inputs, $\tilde{\Delta} : \tilde{X} \times Act \to 2^{\tilde{X}}$ is the non-deterministic transition function defined by: for any two states $\tilde{x} = (x, s), \tilde{x}' = (x', s') \in \tilde{X}$ and control input $a \in Act$, we have

$$\tilde{x}' \in \tilde{\Delta}(\tilde{x}, a) \quad \text{iff} \quad x' \in \Delta(x, a) \wedge \xi(s, L(x')) = s',$$

$\tilde{X}_0 = X_0 \times \{s_0\}$ is the set of initial states, and $\tilde{L} : \tilde{X} \to 2^{\mathcal{AP}}$ is the labeling function defined by: for any $\tilde{x} = (x, s) \in \tilde{X}$, we have $\tilde{L}(\tilde{x}) = L(x)$.

In the product system $\tilde{T}$, we define $F = \{(x, s) \in \tilde{X} \mid s \in S_F\}$ as the set of *accepting states* in which the second components are accepting. We also extend the observation function to $\mathcal{O} : \tilde{X} \times Sens \to Obs$ by: for any $\tilde{x} = (x, s) \in \tilde{X}, \gamma \in Sens$, we have $\mathcal{O}(\tilde{x}, \gamma) = \mathcal{O}(x, \gamma)$.

B. Belief Transition Systems

In the partial observation setting, the controller does not precisely know the current state of the system due to the observation equivalence. Hence, it can only maintain a set of possible current states based on the observation results of the control inputs and the sensing actions applied in history. This set of possible states is referred to as a *belief state* reflecting controller's knowledge of the system and can be updated based on the sensing and control actions.

Also, an active-sensing controller essentially takes a *joint-action*, which is a tuple containing both the control input and the sensing decision. Then the belief of the system evolves when each join-action is issued or when each observation is received. This is captured by the following *belief transition system*.

Definition 2 (Belief Transition Systems): Given product system $\tilde{\mathcal{T}}$, observation function $\mathcal{O} : X \times Sens \to Obs$, the belief transition system is defined as a 6-tuple

$$\mathcal{B} = \langle Q, U, \Omega_B, Q_0 \rangle$$

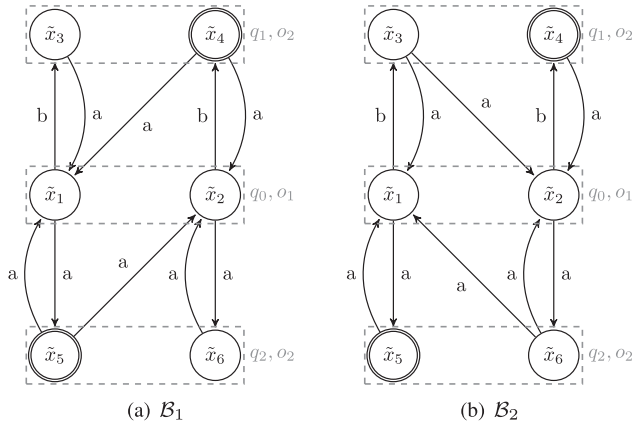
- $Q = \{(q_x, q_s) \in 2^{\tilde{X}} \times Sens \mid \forall \tilde{x}, \tilde{x}' \in q_x, \mathcal{O}(\tilde{x}, q_s) = \mathcal{O}(\tilde{x}', q_s)\}$ is the set of belief states, where for each $q = (q_x, q_s)$, $q_x \in 2^{\tilde{X}}$ represents the set of possible (product) states and $q_s \in Sens$ represents the current sensing decision.
- $U = Act \times Sens$ is the set of joint actions, which are control-sensing pairs;
- $\Omega_B : Q \times U \to 2^Q$ is the non-deterministic transition function defined by: for any $q = (q_x, q_s), q' = (q'_x, q'_s) \in Q$ and $u = (a, \gamma) \in U$, we have $q' \in \Omega_B(q, u)$ iff
  i) $q'_s = \gamma$; and
  ii) $q'_x = (\cup_{\tilde{x} \in q_x} \tilde{\Delta}(\tilde{x}, a)) \cap [\tilde{X}]_{\gamma, o}$ for some $o \in Obs$.
- $Q_0 = \{(q_x, q_s) \in Q \mid \exists o \in Obs \text{ s.t. } q_x = \tilde{X}_0 \cap [\tilde{X}]_{q_s, o}\}$ is the set of all possible initial belief states.

By construction, for each belief state $q = (q_x, q_s)$, all product states in q_x have the same observation under sensing decision $q_s \in Sens$. Therefore, we denote by $\mathcal{H}(q) \in Obs$ the observation of belief state q, which is the unique observation such that

$$\forall \tilde{x} \in q_x : \mathcal{H}(q) = \mathcal{O}(\tilde{x}, q_s).$$

C. Challenges in Control Synthesis

The interaction between the controller and the environment can usually be formulated by a two-player game between the controller and the environment, where the controller's objective is to enforce system's path to satisfy certain properties to win the game, which is to infinitely visit accepting states in this case, and the environment's objective is to prevent the controller from winning the game; see, e.g., [39]. By taking the information uncertainty into account, we capture the game between controller and environment along with information evolution by the belief transition system.



(a) $\mathcal{B}_1$ (b) $\mathcal{B}_2$

Fig. 1. Two examples of belief transition systems.

It has been shown in the literature that, if two states with the same observation always have the same property of being accepting states, then one can solve the control synthesis problem as a full observation Büchi game over the belief transition system. However, the following examples show that, without this assumption, which does not make much sense for our case of active sensing, the control synthesis problem becomes particularly challenging.

The first example shows that, it is problematic to define "accepting states" in the belief transition system.

Example 1 (Non-Existence of Accepting State): Let us consider a belief transition system $\mathcal{B}_1$ presented in Fig. 1(a). For simplicity, we assume that there is a unique sensing action, which corresponds to the case of passive sensing. Therefore, we omit the sensing action in each belief state and joint action. States $\tilde{x}_1, \ldots, \tilde{x}_6$ are already assumed to be product states for some LTL tasks. The initial states are $\{\tilde{x}_1, \tilde{x}_2\}$ and the accepting states are $\{\tilde{x}_4, \tilde{x}_5\}$. The control input set is $U = \{a, b\}$ and the observation set is $O = \{o_1, o_2\}$. We consider a (passive) observation function defined by

$$\mathcal{H}(q_i) = o_1, i = 0 \quad \text{and} \quad \mathcal{H}(q_i) = o_2, i \in \{1, 2\}$$

The belief states are those dashed rectangles in the figure.

If one attempts to solve a complete observation Büchi game over $\mathcal{B}_1$, i.e., ensure infinite visits of accepting states, then the question naturally arises as to "*how to define accepting states in* $\mathcal{B}_1$". There appear to be two options:

i) a belief state is accepting if all states in it are accepting;
ii) a belief state is accepting if some state in it is accepting.

If we adopt the first definition, then there is no accepting state in $\mathcal{B}_1$, and hence, no solution exists. If we adopt the second definition, then belief states $q_1 = \{\tilde{x}_3, \tilde{x}_4\}$ and $q_2 = \{\tilde{x}_5, \tilde{x}_6\}$ are both accepting. Then by solving the full observation Büchi game over $\mathcal{B}_1$, the controller can always take action a at $q_0 = \{\tilde{x}_1, \tilde{x}_2\}$ to visit q_2 infinitely often. However, this is not a correct

solution since the underlying system may loop via $(\tilde{x}_2\tilde{x}_6)^\omega$ in which no accepting state exists.

The following result further shows that, in fact, a belief state-based control strategy is even not sufficient, and one may require additional memory to realize the controller.

Example 2 (Non-Existence of Belief-Based Strategy): We still consider the system in the above example. We can easily check that there is no belief-based control strategy to achieve the objective. For example, we have shown that fixing control decision a at belief state q_0 is not correct; and the same for fixing control decision b. However, we can make control decision a and b *alternatively* at belief state q_0 such that: if the system is actually at $\tilde{x}_1$, then it will get chance to reach accepting state $\tilde{x}_5$, and if the system is actually at $\tilde{x}_2$, then it will get chance to reach accepting state $\tilde{x}_4$. This ensures the infinite visit of accepting states although we still cannot precisely identify the current state.

In the last example, we show that, in fact, simply relying on the information of the belief structure is not sufficient to solve the problem. To this end, we show that it is possible that two systems have exactly the same belief structures, one has a solution but the other has no solution.

Example 3 (Information of Inter-Connections): We consider a belief transition system $\mathcal{B}_2$ that is slightly different from $\mathcal{B}_1$ by changing transitions $\tilde{x}_4 \to \tilde{x}_1$ and $\tilde{x}_5 \to \tilde{x}_2$ into $\tilde{x}_3 \to \tilde{x}_2$ and $\tilde{x}_6 \to \tilde{x}_1$, as shown in Fig. 1(b). Obviously, $\mathcal{B}_1$ and $\mathcal{B}_2$ have the same transition structure between belief states. For this example, however, if the controller attempts to monitor every possible path and alternately enforce future visits to accepting states, it will soon discover that this process will never converge. For example, initialized at q_0, if the controller "guesses" that the system is at state $\tilde{x}_2$ aims to visit accepting state $\tilde{x}_4$ by control input b, then it is possible that the system actually moves from $\tilde{x}_1$ to $\tilde{x}_3$. Furthermore, state $\tilde{x}_3$ may return to both states $\tilde{x}_1$ and $\tilde{x}_2$ with the same observation. Therefore, the controller need to guess again which actual state is at belief state q_0. In fact, no matter what control strategy we use, the loop of $\tilde{x}_1 \to \tilde{x}_3 \to \tilde{x}_2 \to \tilde{x}_6 \to \tilde{x}_1$ can never be avoided. Hence, we can conclude that there is no controller that can ensure the infinite visits of accepting states without the precise state information. Therefore, this example shows that the inter-connection information of states within belief structure is also crucial for control policy synthesis.

V. Synthesis Procedures

In this work, we propose an approach to synthesize control and sensing strategies jointly. Specifically, our approach contains two steps: offline computation and online execution.

- In the offline stage, we solve a reachability game for each belief state by augmenting the belief space to ensure the visit of accepting states in the presence of information uncertainty. We repeat this process iteratively until all belief states have the ability to reach accepting states and maintain the ability globally.
- In the online stage, the controller maintains the current belief state. During each decision round, the controller applies a reachability strategy from the initial belief state. Once all states in the augmented belief space become accepting, the initial belief state is updated for another decision round. This allows us to synthesize a controller that can handle information uncertainty and ensure the satisfaction of the desired specifications.

Next, we detail our synthesis procedures.

A. Augmented Transition Systems

In the belief transition system, a belief state can only capture all possible current (product) states, and can only indicate whether each current state is accepting or not. However, this information is not sufficient to determine a control strategy, as we have shown in the previous examples. To overcome this limitation, we augment each belief state by tracking not only the system state but also whether or not an accepting state *has been visited* within a "decision round" initiated from a belief state $q \in Q$. This leads us to define the *augmented transition systems*, which provides a more comprehensive representation of the system's behavior and enables us to synthesize a more effective control strategy.

Definition 3 (Augmented Transition Systems): Given belief transition system $\mathcal{B}$, and a belief state $q = (q_x, q_s) \in Q$, the augmented transition system is defined as a 5-tuple

$$\mathcal{M}_q = \langle M_q, U, \Omega_q, m_{q0}, M_{qF} \rangle$$

- $M_q \subseteq 2^{\tilde{X}\times\{0,1\}} \times Sens$ is the set of augmented states;
- U is the same set of joint actions;
- $\Omega_q : M_q \times U \to 2^{M_q}$ is the non-deterministic transition function defined by: for any $m = (m_x, m_s) \in M_q \setminus M_{qF}, m' = (m'_x, m'_s) \in M_q$ and $u = (a, \gamma) \in U$, we have $m' \in \Omega_q(m, u)$ if for some $o \in O$, $m'_s = \gamma$,

$$m'_x = \left\{ (\tilde{x}', b') : \begin{array}{c} \exists (\tilde{x}, b) \in m_x, \tilde{x}' \in \tilde{\Delta}(\tilde{x}, a) \cap [\tilde{x}]_{\gamma,o}, \\ b' = 0 \text{ iff } (b = 0 \wedge \tilde{x}' \notin F) \end{array} \right\}$$

  and for any $m \in M_{qF}$, we define $\Omega_q(m, u) = \emptyset, \forall u \in U$;
- $m_{q0} = (m_{q_0,x}, m_{q_0,s})$ is the initial augmented state, where $m_{q_0,s} = q_s$ and

$$m_{q_0,x} = \{(\tilde{x}, b) : \tilde{x} \in q_x, b = 0\}.$$

- $M_{qF} = \{m \in M_q \mid \forall (\tilde{x}, b) \in m_x, b = 1\}$ is the set of accepting augmented states in $\mathcal{M}_q$.

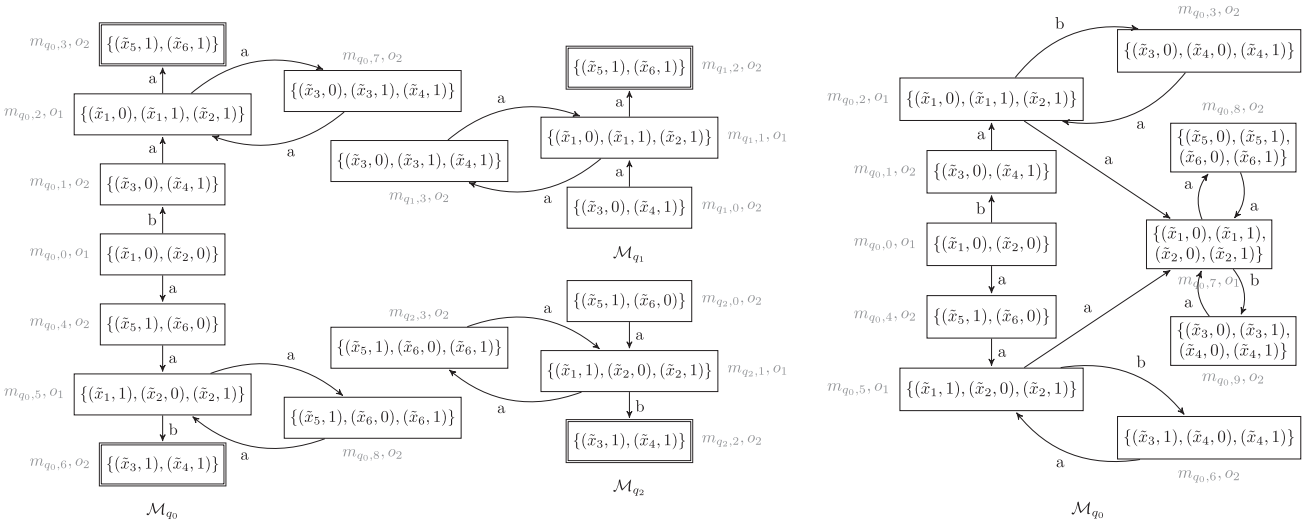
Intuitively, $\mathcal{M}_q$ augments $\mathcal{B}$ by adding a binary counter $b \in \{0, 1\}$ for each possible state in order to track whether or not accepting states have been visited along the path from initial belief q. Therefore, once all binary counters become 1 in the augmented state, we know for sure that all states have visited an accepting state in the decision round, and we terminate the expansion. For each augmented state $m \in M_q$, we denote

$$X(m) = \{\tilde{x} \in \tilde{X} \mid (\tilde{x}, b) \in m_x\}$$

as the corresponding belief state of augmented state m by erasing the binary variables.

The following result formally states the property of the augmented transition system.

Lemma 1: Given a belief transition system $\mathcal{B}$, a belief state $q \in Q$ and its corresponding augmented transition system $\mathcal{M}_q$



(a) $\mathcal{M}_{q0}, \mathcal{M}_{q_1}$ and $\mathcal{M}_{q_2}$ for $\mathcal{B}_1$. (b) $\mathcal{M}_{q0}$ for $\mathcal{B}_2$.

Fig. 2. Examples of augmented transition systems.

and let $\omega = X(m_{q0})X(m_{q_1})\dots X(m_{q_k})$ be a finite path in the $\mathcal{M}_q$ such that $m_{q_k} \in M_{qF}$. Then for any finite paths $\tau = \tilde{x}_0\tilde{x}_1\cdots\tilde{x}_k$ with history information $\gamma_0 o_0 a_0 \gamma_1 o_1 a_1 \cdots \gamma_k o_k$, if $o_0\cdots o_k = \mathcal{H}(\omega)$, then τ must have visited accepting states.

Proof: We prove Lemma 1 by contradiction. Assume there exists a path $\tau = \tilde{x}_0\tilde{x}_1\dots\tilde{x}_k$ with $o_0\cdots o_k = \mathcal{H}(\omega)$ such that $\tilde{x}_i \notin F$ for $\forall i$. We denote $\delta_x(m)$ and $\delta_s(m)$ as the set of possible product states and sensing action, respectively. For $\forall 0 \leq i \leq k-1$, clearly, we have $\tilde{x}_i \in \delta_x(m_{q_i})$ because $o_0\cdots o_k = \mathcal{H}(\omega)$. Given action u, if $m_{q_{i+1}} \in \Omega_q(m_{q_i}, u), \tilde{x}_{i+1} \in \Omega(\tilde{x}_i, u)$ and $(\tilde{x}_i, b) \in m_{q_i,x}$ with $b = 0, \tilde{x}_{i+1} \notin F$. Then, according to the definition of Ω_q, there exists a pair $(\tilde{x}_{i+1}, b') \in m_{q_{i+1},x}$ such that $b' = 0$. So, if $\tilde{x}_i \notin F, \forall 0 \leq i \leq k-1$ and $\exists(\tilde{x}_0, b) \in m_{q_0,x}, b = 0$, then $\exists(\tilde{x}_k, b') \in m_{q_k,x}$, such that $b = 0$, which is contradict to $m_{q_k} \in M_{qF}$. So, there does not exist a path $\tau = \tilde{x}_0\tilde{x}_1\dots\tilde{x}_k$ with $o_0\cdots o_k = \mathcal{H}(\omega)$ such that $\tilde{x}_i \notin F, \forall i$ and Lemma 1 is proved. ■

The following example illustrate the structure of augmented transition system.

Example 4: In Fig. 2 , we present the augmented transition systems for belief transition systems in Fig. 1 from different belief states. For example, $\mathcal{M}_{q0}$ in Fig. 2(a) shows the augmented transition systems for belief transition system $\mathcal{B}_1$ from state $q_0 = \{\tilde{x}_1, \tilde{x}_2\}$. Intuitively, the augmented transition systems starts from $\{(\tilde{x}_1, 0), (\tilde{x}_2, 0)\}$ since no accepting state has been visited. The structure will terminate at states $\{(\tilde{x}_5, 1), (\tilde{x}_6, 1)\}$ and $\{(\tilde{x}_3, 1), (\tilde{x}_4, 1)\}$ since all binary variables in the states are 1, which means that although we do not know the precise state, we still know for sure that each state has visited an accepting state from q_0. However, augmented transition system for $\mathcal{B}_2$ from q_0 shown in Fig. 2(b) does not contain any accepting augmented states. This implies that it is impossible for belief state q_0 to reach a belief state where every possible underlying actual path has visited accepting states.

B. Offline Computation of Winning Regions

Suppose that our current belief state regarding the system is $q \in Q$. To achieve a successful decision round, we need to apply a control strategy in $\mathcal{M}_q$ such that accepting states $\mathcal{M}_{qF}$ can be visited for sure. This is essentially a *reachability game* in which a target region needs to be reached within finite steps.

Formally, let $\imath \subseteq M_q$ be a set of belief states in $\mathcal{M}_q$ one wants to reach. We define the *controlled predecessor* of $\imath$ by

$$\texttt{CPre}(\imath) = \{m \in M_q \mid \exists u \in U \text{ s.t. } \Omega_q(m, u) \subseteq \imath\}.$$

The k-*step attractor* $\texttt{Attr}^{(i)}(\imath)$ of $\imath$ is defined inductively by

$$\texttt{Attr}^{(0)}(\imath) = \imath$$

$$\texttt{Attr}^{(k+1)}(\imath) = \texttt{Attr}^{(k)}(\imath) \cup \texttt{CPre}(\texttt{Attr}^{(k)}(\imath)).$$

Then the *attractor* of $\imath$ is defined by

$$\texttt{Attr}(\imath) = \bigcup_{k\geq 0} \texttt{Attr}^{(k)}(\imath).$$

Clearly, for each belief state $q \in Q$, if $m_{q0} \notin \texttt{Attr}(M_{qF})$, then q should be considered as an "illegal" belief state since we cannot ensure the finite reachability of accepting state. Furthermore, recall that our overall objective is to visit accepting states infinitely often. Therefore, once we have determined that q is an illegal belief state and delete it, this may further affect the legality of each belief state q', which was legal. Therefore, we need to iteratively determine the legality of belief states by solving reachability games on a dynamically maintained augmented space.

To this end, given a belief state $q \in Q$, augmented transition system $\mathcal{M}_q$ and a set of belief states $R \subseteq Q$, which is referred to as *reference region*, we define $M_q(R) = \{m \in M_q \mid X(m) \in R\}$ as the set of corresponding augmented states in $\mathcal{M}_q$. Then

we define validation function $\texttt{val}(q, R)$ as follows:

$$\texttt{val}(q,R) = \begin{cases} \mathsf{True}, & \text{if } m_{q0} \in \texttt{Attr}(M_{qF} \cap M_q(R)) \\ \mathsf{False}, & \text{otherwise} \end{cases}$$

The validation function provides the result of whether the reachability game from the initial state m_{q0} to the accepting states M_{qF} in the given reference region R has a solution. In other words, this function measures whether the controller can start from belief state q and surely reach a belief state such that during this process all the possible paths have visited accepting states projected in reference region R.

Based on this reachability game in augmented transition system, the *winning region* Q_{win} can be computed iteratively by

$$\begin{aligned} R_0 &= Q \\ R_{i+1} &= \{q \in R_i \mid \texttt{val}(q, R_i) = True\} \\ Q_{win} &= \lim_{i\to\infty} R_i \end{aligned} \tag{1}$$

Intuitively, the reference region is initially set to be all belief states and then shrinks iteratively. In each iteration, the validation function is examined on each belief state with current reference region R_i. Belief states from which the controller cannot deterministically visit accepting states will be deleted. Since the reference region is changed, the validation of each state also needs to be updated by solving reachability games again. The final result of winning region is the limit of the above reference region sequence. Once the process converges, starting from any belief state $q \in Q_{win}$, the controller can induce a finite sequence ending at another $q' \in Q_{win}$ such that all the possible finite paths contain accepting state during this process, and maintain this ability for the next round.

C. Online Execution of Controller

Based on the pre-computed winning regions, we can now decode an active-sensing controller. The main online execution process works as follows to generate control and sensing outputs according to observation results indefinitely.

- Initially, the controller performs a sensing action to obtain some initial information. Formally, we define the *winning initial sensing set* by

  $$\Gamma = \{\gamma \in Sens : \forall o \in O, (\tilde{X}_0 \cap [\tilde{x}]_{\gamma,o}, \gamma) \in Q_{win}\}.$$

  If $\Gamma = \emptyset$, then any initial sensing action may result in a belief state outside of the winning region, which means that the joint strategy does not exist. Otherwise, we can design a sure winning control strategy beginning with any sensing action in Γ.
- Then starting from an arbitrary given state in $q \in Q_{win}$, the controller tries to enforce all possible product states towards accepting states in $\mathcal{M}_q$ by applying a (state-based) sure reachability strategy denoted by Φ_q, where each element is a pair of augmented state and its corresponding control decision. This is referred to as a *decision round*.

Algorithm 1: Control Strategy.

Input: belief transition system $\mathcal{B}$, winning region Q_{win}, augmented transition systems $\{\mathcal{M}_q\}_{q\in Q_{win}}$
Output: Control input $u = (a, \gamma)$

```
1: Apply γ ∈ Γ, and receive an initial observation o
2: q ← (X̃_0 ∩ [x̃]_{γ,o}, γ)
3: M ← M_q, Φ ← Φ_q, m ← m_0
4: find (m, u) ∈ Φ
5: while apply u and receive new observation o do
6:    q ← q', q' ∈ Ω_B(q, u), H(q') = o.
7:    m ← m', m' ∈ Ω(m, u), X(m') = q,
8:    if m ∈ M_F then
9:       M ← M_q, Φ ← Φ_q, m ← m_0
10:      find (m, u) ∈ Φ,
11:   else
12:      find (m, u) ∈ Φ,
13:   end
14: end
```

- After the generation of a finite belief path ending in Q_{win} with every possible path has visited accepting states, a new decision round will be initialized at current belief state.

The above process is formally summarized by Algorithm 1. Specifically, the initialization is conducted in line 1–3, where the controller chooses an initial sensing decision γ, observes the first observation o, and uses this information to form its first belief q. We use $\mathcal{M} = \langle M, U, \Omega, m_0, M_F\rangle$ and Φ to record current augmented transition system and its reachability strategy. Then the controller will keep applying joint decision $u = (a, \gamma)$ according to Φ. If no accepting state in M_F is reached, then the strategy Φ remains unchanged. However, if an accepting state is reached, then it means that a decision round has been finished. Therefore, we need to update the augmented transition state by initializing from the current belief q and also to update the reachability strategy based on the corresponding Φ_q.

Finally, for each $q \in Q_{win}$, the corresponding reachability strategy Φ_q can be generated based on each augmented transition system $\mathcal{M}_q$ by Algorithm 2. Given an augmented transition system $\mathcal{M}_q$, for an arbitrary $m \in M_q$, assuming it is in the $(i+1)$-step attractor of accepting augmented states, then the reachability strategy $\Phi_q$is to enforce its visit towards the i-step attractor. Note that, the augmented transition systems only provide a transition structure with visiting situations, and do not incorporate the winning region information. Simply reaching towards M_{qF} may result in leaving the winning region and thus losing the game. Therefore, accepting augmented state set should be as $M_{qF} \cap M_q(Q_{win})$, which ensures that the transition stays within the winning region. Once the strategy at each step is generated in augmented transition systems, it can be projected and implemented back in belief transition systems.

D. Correctness Proofs and Complexity Analysis

Finally, we prove the correctness of our approach and analyze the complexity of our algorithm. First, we prove the soundness

Algorithm 2: Computation of Φ_q.

Input: $\mathcal{M}_q, Q_{win}$
Output: Φ_q
$\Phi_q \leftarrow \emptyset$
for every $m \in M_q$ **do**
  find i such that
  $m \in \mathtt{Attr}^{(i+1)}(M_{qF} \cap M_q(Q_{win})) \setminus \mathtt{Attr}^{(i)}(M_{qF} \cap M_q(Q_{win}))$
  pick an action-sense pair u such that
  $\Omega_q(m,u) \subseteq \mathtt{Attr}^{(i)}(M_{qF} \cap M_q(Q_{win}))$
  $\Phi_q \leftarrow \Phi_q \cup (m,u)$
end

of our approach, i.e., if Algorithm 1 runs indefinitely, then the resulting active-sensing controller indeed solves the problem.

Proposition 1 (Soundness): Given a label transition system $\mathcal{T}$, the product system $\tilde{T}$, and its corresponding belief structure $\mathcal{B}$, if $\Gamma \neq \emptyset$, then we have $\mathsf{Trace}(\mathcal{T}_C) \subseteq \mathsf{Word}(\phi)$ under the online control strategy C defined by Algorithm 1.

Proof: The term $\mathsf{Trace}(\mathcal{T}_C) \subseteq \mathsf{Word}(\phi)$ can be transformed into the following equivalent term that for each infinite path $\tilde{\tau} \in \mathsf{Path}(\tilde{T}_C)$ we have $\mathsf{inf}(\tilde{\tau}) \cap F \neq \emptyset$, where $\tilde{T}_C = \mathcal{T}_C \times \mathcal{A}_\phi$. Since the standard reachability game algorithm is applied, from any initial state the controller can deterministically visit an accepting state in finite steps. So, from any belief state $q \in Q_{win}$, its corresponding m_{q0} will visit an augmented state $m \in M_{qF}$ with $\forall (x,b) \in m_x, b = 1$ in finite steps under controller C. Also, according to Lemma 1, the above implies that every finite path having the same observation with the belief path starting from q and ending at $q' = X(m)$ has visited accepting states. The process will be conducted infinitely, so the infinite paths induced by C satisfy the accepting condition. ■

Next, we show the completeness of our algorithm.

Proposition 2 (Completeness): Given a label transition system $\mathcal{T}$, the product system $\tilde{T}$, and its corresponding belief structure $\mathcal{B}$, if there exists a control strategy C such that $\mathsf{inf}(\tilde{\tau}) \cap F \neq \emptyset, \forall \tilde{\tau} \in \mathsf{Path}(\tilde{T}_C)$, then $\Gamma \neq \emptyset$.

Proof: Without losing generality, assume there exists a strategy C such that $\mathsf{inf}(\tilde{\tau}) \cap F \neq \emptyset, \forall \tilde{\tau} \in \mathsf{Path}(\tilde{T}_C)$, which is not necessarily our proposed strategy, but we can still project its induced path on belief transition systems and augmented transition systems. We denote the belief transition system under control as $\mathcal{B}_C$, and the set of all possible belief states appear in $\mathsf{Path}(\mathcal{B}_C)$ as $\hat{Q}$. We have $\mathsf{inf}(\mathsf{Path}(\tilde{T}_C)) \cap F \neq \emptyset$, so from any belief state $q \in \hat{Q}$, the controller can deterministically enforce every possible future path of visiting accepting states in finite steps. This means that for $\forall q \in \hat{Q}$, we have $\mathtt{val}(q, \hat{Q}) = True$. Also, it can be easily inferred that if the Reachability Game towards set A has a solution, then it also has a solution for any $A' \supseteq A$. So, if $\mathtt{val}(q, A) = True$, then for any $A' \supseteq A$, we have $\mathtt{val}(q, A') = True$. If we make an assumption that $\hat{Q} \nsubseteq Q_{win}$, and for any $q' \in \hat{Q} \setminus Q_{win}$, there exists an iteration term j such that $\mathtt{val}(q', R_j) = False, j \geq 0$ during the computation of Q_{win}. We consider the biggest corresponding iteration item among all the states in $\hat{Q}$, denoted by J. Clearly, $\hat{Q} \subseteq R_J$ and $J \geq j$, and because $\forall q \in \hat{Q}, \mathtt{val}(q, \hat{Q}) = True$, we have $\mathtt{val}(q', R_J) = True$, which is contradict to the assumption of $\mathtt{val}(q', R_j) = False$. So, for any strategy C that satisfies $\mathsf{inf}(\tilde{\tau}) \cap F \neq \emptyset, \forall \tilde{\tau} \in \mathsf{Path}(\tilde{T}_C)$, we have $\hat{Q} \subseteq Q_{win}$. Also, because $\hat{Q} \subseteq Q_{win}$, the result belief states of the initial sensing action $\hat{\gamma}$ in strategy $\hat{C}$ must also be inside Q_{win}, that is $(\tilde{X}_0 \cap [\tilde{x}]_{\hat{\gamma},o}, \hat{\gamma}) \in Q_{win}$ for $\forall o \in O$. So $\Gamma \neq \emptyset$ and at least contains an element of $\hat{\gamma}$, which completes the proof. ■

Finally, we analyze the complexity of our approach. First, the number of product states is at most $|\tilde{X}| = |X| \cdot |S|$, which is linear in both the original transition system and the DBA, The belief transition system is exponentially large than the product system and contains at most $n = |Sens| \cdot 2^{|\tilde{X}|}$ states. Similarly, each augmented transition system also contains at most $n' = |Sens| \cdot 2^{2|\tilde{X}|}$ states and $m' = |Sens|^2|Act|$ actions. Since the complexity for solving the reachability game is linear in both the number of states and the number of actions [39], the complexity for each reachability analysis for augmented transition system is $O(m'n')$. Furthermore, the reachability analysis needs to be iterated for at most n times. This is because if the reference region only excludes one belief state at each iteration, and the worst-case is $Q_{win} = \emptyset$, i.e., the iteration repeats $\frac{n \times (n+1)}{2}$ times, where n is the number of belief states. Therefore, the complexity of offline computation is $O(|Sens|^4|Act|2^{4|\tilde{X}|})$. Note that, once the offline computation is done, the online decision making part is a table search; the only computation effort is to update the belief state on-the-fly, which is linear in the size of the augmented state space.

VI. Experiments

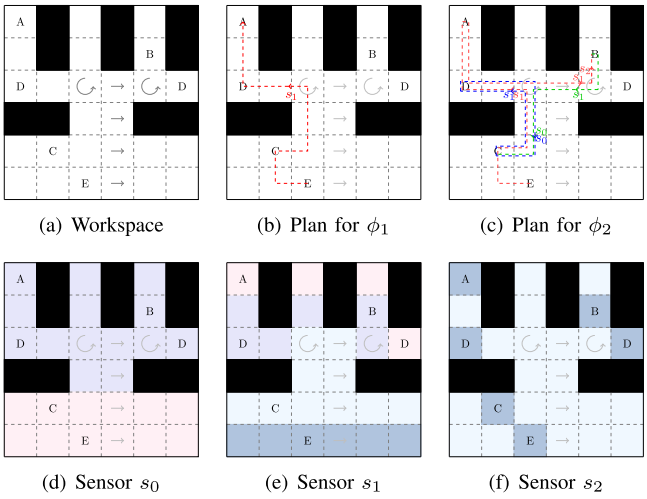
In this section, we provide a set of experiments to illustrate our theoretical results. In Section VI-A, we provide a case study simulation to better illustrate our method. In Section VI-B, we provide numerical experiments by increasing the system scale and sensing actions space to show the scalability of our algorithm for different scenarios. Finally, in Section VI-C we perform a hardware experiment to show the real-world feasibility of our method.

All simulations are implemented by integrating Python 3.10.7 and UGV simulation platform V−REP 4.2.0 on a PC with 64 cores with 2.80 GHz processors and 16 GB of RAM. The UGV used in the hardware experiment is a Turtlebot3-Burger mobile UGV equipped with Raspberry Pi camera v2. All codes and experiment videos are available at https://github.com/LiShuaiyi/active-LTL-planning.

A. Illustrative Case Study

In this section, we illustrate the proposed active-sensing control synthesis by a case study of UGV task planning in a grid world environment with uncertainty.

System Model: We consider an omnidirectional UGV moving in a workspace described by a 6×6 grid-world shown in Fig. 3(a). More specifically, the workspace represents a dark cold storage warehouse that is divided into storage region and outside region (see different colors in Fig. 3(d)). The storage



(a) Workspace (b) Plan for ϕ_1 (c) Plan for ϕ_2

(d) Sensor s_0 (e) Sensor s_1 (f) Sensor s_2

Fig. 3. Illustrative example of UGV task planning in grid-world environment.

region maintains a relatively lower temperature and has no light source, while the temperature of the outside region is higher and is installed a row of light sources. The two regions are separated by glass automatic door. Each grid represents a region in the UGV's workspace, and also corresponds to a state of the system model, denoted by $X = \{(x, y), 1 \leq x, y \leq 6\}$. Specifically, at each instant, the UGV may take a control action in $Act = \{\mathsf{left}, \mathsf{right}, \mathsf{up}, \mathsf{down}\}$ to move left/right/up/down to its adjacent region. In most of the states without gray arrows, we assume that the UGV will execute the control action perfectly. For example, if the robot is at state (1,1), which is bottom-left corner state and take control action up, then it will move to state (2,1) for sure. Furthermore, in those regions marked with gray arrows, we assume that the movement of the UGV is non-deterministic due to the severe pavement conditions. Specifically,

- If the UGV is in a region marked by $\rightarrow$, then when it takes actions up or down, it may also non-deterministically move to the adjacent region on its right. For example, if the robot is at state (4, 4) and takes control action down, then it will either move to states (3, 4) and (4, 5).
- If the UGV is in a region marked by $\circlearrowleft$, then any control action may fail for the first time such that the UGV is unmoved. For example, if the robot is at state (4, 3) and takes control action `left`, then it may either move to state (4, 2) or stay at (4, 3). However, if it stays at (4, 3), then the next control action will always be effectively executed.

Also, we assume that the UGV is initialized deterministically at region E, i.e., $X_0 = \{(3, 1)\}$.

Observation Model: We assume that the UGV is equipped with three physical sensors: one passive sensor s_0, which is a thermometer that monitors the temperature difference between different regions and is always activated, and two active sensors s_1 and s_2, which can be turned on/off actively online. Sensor s_1 is an ambient light sensor that detects different level of light intensity. Sensor s_2 is an infrared sensor that detects the infrared signal of specific electronic devices that are related to task completion. Due to energy constraints, we assume that the UGV can simultaneously activate two sensors including the passive sensor s_0. Therefore, the UGV essentially has three sensing actions, i.e., $Sens = \{\{s_0\}, \{s_0, s_1\}, \{s_0, s_2\}\}$. The capability of each sensor is shown in Fig. 3(d), (e), and (f). Specifically, in each figure, that states have the same color means that the sensor cannot distinguish these states, i.e., each sensor induces an observational equivalent class. Therefore, Obs contains $2 + 2 \times 4 + 2 \times 2 = 14$ symbols, which is the product of the equivalent class of each sensor. For example, with the passive sensor s_0, the UGV can distinguish whether it is in the bottom two rows or in the top four rows. Also, by activating sensor s_2, the UGV can distinguish whether it is in regions with letters (although cannot tell which one explicitly) or in those non-letter regions.

Simulation for Task 1: We assume that regions marked by letters A to E are those with our particular interests. Furthermore, regions marked by black color are dangerous regions the UGV should avoid. First, we consider a simple task for the UGV described by the following LTL formula:

$$\phi_1 = \Diamond A \wedge \Diamond C \wedge \Box \neg dangerous.$$

That is, the UGV needs to visit both regions A and C without order by avoiding the dangerous regions.

By applying our algorithm, the planned trajectory is shown in Fig. 3(b). Specifically, since there is no transition non-determinism before reaching region (4, 3), the UGV does not need to turn on active sensors s_1 or s_2. However, at region (4, 3), after the UGV takes control action `left`, it needs to activate sensor s_1 since this sensor can distinguish between regions (4, 3) and (4, 2); the former means that action `left` fails and the latter means that action `left` is successfully executed. This information is needed since the UGV needs to determine how many `left` actions it should take before applying `up` to region A.

Simulation for Task 2: Still for the same system, we consider a more complex task described by the following LTL formula:

$$\phi_2 = (\neg D \,\mathcal{U}\, C) \wedge (\neg A \,\mathcal{U}\, D) \wedge (\neg B \,\mathcal{U}\, A) \wedge \Diamond B \wedge \Box\Diamond C \wedge \Box\Diamond D \wedge \Box \neg dangerous$$

Essentially, the task consists of three parts:

i) the finite task of visiting C, D, A, B in order; and
ii) the infinite surveillance task of visiting C and D infinitely often; and
iii) the safety task of avoiding dangerous regions.

By applying our algorithm, the planned trajectory is shown in Fig. 3(c). Specifically, the red trajectory aims to fulfill the finite task. Still, after the UGV takes control action `left` at region (4, 3), it needs to activate sensor s_1 to resolve the transition non-determinism. Similarly, after the UGV takes control action `up` at region (4, 5), it needs to activate sensor s_2 to determine whether it has already reached B. Once the UGV reaches region B, it will follow the green trajectory to go to region C, and then follow the blue trajectory to visit between regions C and D indefinitely. Still, different sensors need to be activated at different regions

(marked in the figure) in order to resolve information uncertainty such that the planned trajectory can be executed.

B. Numerical Experiments

In the previous section, we analyzed that the worst-case complexity of our synthesis method is polynomial in the number of sensing actions and exponential in the number of states. However, this worst-case complexity is rarely achieved for physical systems. Specifically, the exponential upper-bound comes from the fact the belief state space is the power set of the original state space. In practice, however, the states in the beliefs are usually "closed" to each other, i.e., it is not practical for the UGV to maintain a belief that it may be at two possible regions far away from each other. To justify this argument, we perform numerical experiments on randomly generate grid-world scenarios and show the statistical results.

Experiment Setting: To show the scalability of our method, we run our algorithm on randomly generated $n \times n$ grid-world scenarios generated as follows:

- *State Space:* Each grid represents a region in the UGV's workspace, i.e., $X = \{(x, y) : 1 \leq x, y \leq n\}$. The initial region of UGV is randomly chosen.
- *Transition Function:* At each region, the UGV can take `left`/`right`/`up`/`down` to move left/right/up/down to its adjacent region, which are referred to normal transitions. In addition, to introduce transition non-determinism, we randomly add non-deterministic transitions moving diagonally such as $(1, 1) \xrightarrow{\text{up}} (2, 2)$; the number of non-deterministic transitions is 12.5% of that of normal transitions.
- *Observation Model:* For the sake of simplicity, here we directly define sensing actions without assigning physical meanings for the underlying sensors. Specifically, the UGV has N_s number of sensing actions. For each sensing action, we randomly assign each region to $\lceil \frac{(n-1)^2}{3} \rceil$ observational equivalent classes.
- *LTL Task:* We assume there are four atomic propositions. Then we select 25% states and assign them atomic propositions randomly. The overall LTL task is generated randomly with template

$$\phi = \varphi \wedge \left(\bigwedge_{i=1,\ldots,k, k \leq 4} \square \diamondsuit a_i \right),$$

  where φ is a randomly generated task containing at most 15 operators and there are at most four randomly chosen atomic propositions that are needed to be visited infinitely often.

Therefore, there are two parameters controlling the generation of the simulation: system size n and the number of sensing actions N_s.

For each randomly generated system, we compute the product system $\tilde{T}$, belief transition system $\mathcal{B}$, and the augmented transition system $\mathcal{M}_q$ for each q. We denote by $|M_q|$ and $|\Omega_q|$ the average number of states and transitions in $\mathcal{M}_q$ among all possible q. The offline computation involves computing $\mathcal{B}$ and all $\mathcal{M}_q$; the overall computation time is denoted by t_1. The online computation involves belief updates and reading decision from $\mathcal{M}_q$; we denoted by t_2 the average online computation time at each instant. For each pair of parameters n and N_s, we randomly generate 50 systems and all statistics are considered as the averaged values in the 50 experiments with the same parameter.

Experimental Results: In the first set of experiments, we fix the number of sensing actions to be $N_s = 2$ and increase the size of n. The average statistics for different n are shown in Table I. In the second set of experiments, we fix the size of the grid-world as 6×6 and increase the number of sensing actions from 2 to 6, denoted by N_s. The average statistics for different N_s are shown in Table II. For both cases, we see that the sizes of the belief transition system and the augmented transition system grow almost linearly when the parameter n or N_s increase. In particular, in terms of the scalability with respect to the size of the system, the offline computation time t_1 grows much slower than the exponential theoretical bound. Furthermore, for all cases, the online computation time can always be done within one sec. Therefore, our experiments show that our algorithm has good scalability performance for practical scenarios.

C. Hardware Experiments

Finally, we perform a hardware experiment to apply our algorithm in real-world and verify its feasibility. Note that our approach offers discrete high-level trajectory plans between adjacent regions, which can are executed by low-level motion control.

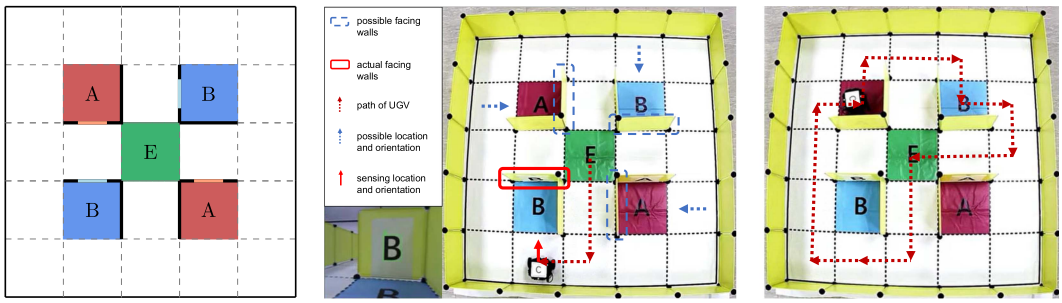
System Model: We consider a pseudo-omnidirectional UGV moving in a 5×5 environment whose map is shown in Fig. 4(a). There are walls in the workspace, denoted by thick black lines. In this pseudo-omnidirectional model, the state of the system is not only the current grid, but also the orientation of the UGV. Therefore, the set of states is $X = \{(x, y, z), 1 \leq x, y \leq 5, z \in \{\uparrow, \downarrow, \rightarrow, \leftarrow\}\}$.

At any given moment, the UGV has the option to turn left/right in situ or move forward, denoted by action sets $Act = \{\text{turn-left}, \text{turn-right}, \text{forward}\}$. We assume that the UGV starts from region E without knowing its orientation, i.e., we have $X_0 = \{(3, 3, z) : z \in \{\uparrow, \downarrow, \rightarrow, \leftarrow\}\}$. Furthermore, we assume that all action of the UGV can be deterministically executed expect that there exist disturbances once it reaches region E, simulated by artificially changing the UGV's orientation randomly by human operator.

Observation Model: Some walls (denoted by lines with colors in the middle in Fig. 4(a)) in the workspace have visual marks representing locations. The UGV is equipped with one sensor, a camera in the front, that can be turned on to detect the visual mark. Therefore, without the artificial disturbance in region E, the UGV always knows where it is, and therefore, no sensor is needed. However, due to the disturbance in region E, the UGV needs to activate the camera suitably at regions adjacent to A or B to localize itself.

TABLE I
SUMMARY STATISTICS FOR DIFFERENT n WITH FIXED $N_s = 2$

$\mathcal{T}$	$\tilde{T}$		$\mathcal{B}$		$\mathcal{M}_q$		t_1(sec)	t_2(sec)
	$\lvert\tilde{X}\rvert$	$\lvert\tilde{\Delta}\rvert$	$\lvert Q\rvert$	$\lvert\Omega_B\rvert$	$\lvert M_q\rvert$	$\lvert\Omega_q\rvert$		
4×4	72	247	144	996	118	794	0.82	0.007
5×5	112	401	224	1608	181	1257	2.31	0.01
6×6	155	593	314	2419	270	2050	5.95	0.05
7×7	213	823	427	3300	351	2680	12.26	0.09
8×8	293	1174	588	4707	474	3682	24.42	0.16
9×9	340	1319	680	5483	543	4903	39.01	0.23
10×10	375	1525	742	5818	693	5520	46.64	0.32
15×15	1013	4240	2031	17030	1629	14346	721.48	2.36
20×20	1505	6436	3014	24798	2135	18132	1194.72	3.78



(a) Workspace map. (b) Snapshots during the experiment. (c) Snapshots during the experiment.

Fig. 4. Hardware experiment results.

TABLE II
SUMMARY STATISTICS FOR DIFFERENT N_s WITH FIXED $n = 6$

N_s	$\tilde{T}$		$\mathcal{B}$		$\mathcal{M}_q$		t_1(sec)	t_2(sec)
	$\lvert\tilde{X}\rvert$	$\lvert\tilde{\Delta}\rvert$	$\lvert Q\rvert$	$\lvert\Omega_B\rvert$	$\lvert M_q\rvert$	$\lvert\Omega_q\rvert$		
2	155	593	314	2419	270	2050	5.95	0.05
3	155	580	467	5239	376	4140	23.66	0.13
4	151	566	594	8823	522	7620	59.57	0.27
5	152	568	699	14199	628	11492	102.41	0.34
6	145	543	871	19556	735	15671	237.62	0.48
7	155	601	1055	24766	939	28763	566.34	1.67
8	162	613	1312	38512	1173	36100	852.34	2.65

LTL Task and Experiment Result: We consider task ϕ_3 described as follows:

$$\phi_3 = \square\diamondsuit E \wedge \square(A \rightarrow \bigcirc(\neg A \wedge \neg E)\mathcal{U}B)$$
$$\wedge \square(B \rightarrow \bigcirc(\neg A \wedge \neg B)\mathcal{U}E)$$
$$\wedge \square(E \rightarrow \bigcirc(\neg B \wedge \neg E)\mathcal{U}A)$$

Intuitively, the UGV needs to visit regions E, A and B sequentially for infinite number of times.

The experiment snapshots are shown in Fig. 4(b), (c). In Fig. 4(b), the UGV begins at point E and is assigned by a human operator to orient the region below E (unknown to the UGV). From the UGV's perspective, it takes control actions: moving forward, turning right, moving forward as shown in Fig. 4(b). At this point, the UGV remains uncertain about its precise location, since it may also be at the other three possible locations shown in Fig. 4(b). Then the UGV will activate the camera and detect the mark in its front wall. This resolves its location uncertainty and it knows that the other three locations are not possible. Therefore, the UGV will not move forward to go to region B in front of it. Instead, it will follow the trajectory shown in Fig. 4(c) to first visit region A, and then region B, and finally back to region E without the need of further detection. The above process will be repeated indefinitely to accomplish the infinite task. The full experiment video is available in https://github.com/LiShuaiyi/active-LTL-planning.

VII. CONCLUSION

In this paper, we addressed the problem of joint synthesis of control and sensing strategies for LTL tasks. To this end, we proposed an effective approach integrating both offline computation of winning regions with online control executions. We show that our algorithm is both sound and complete. Compared with existing works on control synthesis with imperfect information of LTL tasks, our result relaxes the assumption that all states with the same observation must have the same property. The scalability and adaptability of our approach were illustrated by numerical experiments and real-world implements. In the future, we plan to extend our approach to general LTL formulae accepted by deterministic Rabin automata or non-deterministic Büchi automata. Also, we aim to investigate the trade-off between control and sensing by introducing cost metrics.

REFERENCES

[1] S. Khan and J. Guivant, "Design and implementation of proximal planning and control of an unmanned ground vehicle to operate in dynamic environments," *IEEE Trans. Intell. Veh.*, vol. 8, no. 2, pp. 1787–1799, Feb. 2023.

[2] D. Yin, X. Yang, H. Yu, S. Chen, and C. Wang, "An air-to-ground relay communication planning method for UAVs swarm applications," *IEEE Trans. Intell. Veh.*, vol. 8, no. 4, pp. 2983–2997, Apr. 2023.

[3] L. Chen et al., "Milestones in autonomous driving and intelligent vehicles: Survey of surveys," *IEEE Trans. Intell. Veh.*, vol. 8, no. 2, pp. 1046–1056, Feb. 2023.

[4] S. Teng et al., "Motion planning for autonomous driving: The state of the art and future perspectives," *IEEE Trans. Intell. Veh.*, vol. 8, no. 6, pp. 3692–3711, Jun. 2023.

[5] V. S. Chirala, K. Sundar, S. Venkatachalam, J. M. Smereka, and S. Kassoumeh, "Heuristics for multi-vehicle routing problem considering human-robot interactions," *IEEE Trans. Intell. Veh.*, vol. 8, no. 5, pp. 3228–3238, May 2023.

[6] Z. Wen, Y. Zhang, X. Chen, J. Wang, Y.-H. Li, and Y.-K. Huang, "TOFG: Temporal occupancy flow graph for prediction and planning in autonomous driving," *IEEE Trans. Intell. Veh.*, to be published, doi: 10.1109/TIV.2023.3296209.

[7] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI Commun.*, vol. 29, no. 1, pp. 151–162, 2016.

[8] M. Guo, T. Liao, J. Wang, and Z. Li, "Hierarchical motion planning under probabilistic temporal tasks and safe-return constraints," *IEEE Trans. Autom. Control*, vol. 68, no. 11, pp. 6727–6742, Nov. 2023.

[9] G. Silano, T. Baca, R. Penicka, D. Liuzza, and M. Saska, "Power line inspection tasks with multi-aerial robot systems via signal temporal logic specifications," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 4169–4176, Apr. 2021.

[10] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *Int. J. Robot. Res.*, vol. 34, no. 2, pp. 218–235, 2015.

[11] 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.

[12] X. Yu, X. Yin, S. Li, and Z. Li, "Security-preserving multi-agent coordination for complex temporal logic tasks," *Control Eng. Pract.*, vol. 123, 2022, Art. no. 105130.

[13] M. Kloetzer and C. Mahulea, "Path planning for robotic teams based on LTL specifications and petri net models," *Discrete Event Dyn. Syst.*, vol. 30, pp. 55–79, 2020.

[14] W. Shi, Z. He, W. Tang, W. Liu, and Z. Ma, "Path planning of multi-robot systems with boolean specifications based on simulated annealing," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 6091–6098, Jul. 2022.

[15] P. Lv, G. Luo, Z. Ma, S. Li, and X. Yin, "Optimal multi-robot path planning for cyclic tasks using petri nets," *Control Eng. Pract.*, vol. 138, 2023, Art. no. 105600.

[16] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic, "Model-free reinforcement learning for stochastic games with linear temporal logic objectives," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 10649–10655.

[17] M. Cai, E. Aasi, C. Belta, and C.-I. Vasile, "Overcoming exploration: Deep reinforcement learning for continuous control in cluttered environments from temporal logic specifications," *IEEE Robot. Automat. Lett.*, vol. 8, no. 4, pp. 2158–2165, Apr. 2023.

[18] K. Cai, R. Zhang, and W. M. Wonham, "Relative observability of discrete-event systems and its supremal sublanguages," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 659–670, Mar. 2015.

[19] X. Yin and S. Lafortune, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 5, pp. 1239–1254, May 2016.

[20] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Trans. Autom. Control*, vol. 61, no. 8, pp. 2140–2154, Aug. 2016.

[21] Y. Hu, Z. Ma, and Z. Li, "Design of supervisors for active diagnosis in discrete event systems," *IEEE Trans. Autom. Control*, vol. 65, no. 12, pp. 5159–5172, Dec. 2020.

[22] X. Yin and S. Lafortune, "A general approach for optimizing dynamic sensor activation for discrete event systems," *Automatica*, vol. 105, pp. 376–383, 2019.

[23] W. Wang, "Online minimization of sensor activation for supervisory control," *Automatica*, vol. 73, pp. 8–14, 2016.

[24] W. Wang, C. Gong, and D. Wang, "Optimizing sensor activation in a language domain for fault diagnosis," *IEEE Trans. Autom. Control*, vol. 64, no. 2, pp. 743–750, Feb. 2019.

[25] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi, "Supervisory control and reactive synthesis: A comparative introduction," *Discrete Event Dyn. Syst.*, vol. 27, pp. 209–260, 2017.

[26] B. Ramasubramanian, L. Niu, A. Clark, L. Bushnell, and R. Poovendran, "Secure control in partially observable environments to satisfy specifications," *IEEE Trans. Autom. Control*, vol. 66, no. 12, pp. 5665–5679, Dec. 2021.

[27] R. Majumdar, N. Ozay, and A.-K. Schmuck, "On abstraction-based controller design with output feedback," in *Proc. 23rd Int. Conf. Hybrid Syst.: Comput. Control*, 2020, pp. 1–11.

[28] M. Khaled, K. Zhang, and M. Zamani, "A framework for output-feedback symbolic control," *IEEE Trans. Autom. Control*, vol. 68, no. 9, pp. 5600–5607, Sep. 2023.

[29] J.-F. Raskin, T. A. Henzinger, L. Doyen, and K. Chatterjee, "Algorithms for omega-regular games with imperfect information," *Log. Methods Comput. Sci.*, vol. 3, no. 4, pp. 1–23, 2007.

[30] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 4817–4822.

[31] D. Gujarathi and I. Saha, "MT*: Multi-robot path planning for temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 13692–13699.

[32] 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, 2020.

[33] H. Bai, D. Hsu, and W. S. Lee, "Integrated perception and planning in the continuous space: A POMDP approach," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1288–1302, 2014.

[34] J. Fu and U. Topcu, "Synthesis of joint control and active sensing strategies under temporal logic constraints," *IEEE Trans. Autom. Control*, vol. 61, no. 11, pp. 3464–3476, Nov. 2016.

[35] R. R. Da Silva, V. Kurtz, and H. Lin, "Active perception and control from temporal logic specifications," *IEEE Contr. Syst. Lett.*, vol. 3, no. 4, pp. 1068–1073, Oct. 2019.

[36] R. A. MacDonald and S. L. Smith, "Active sensing for motion planning in uncertain environments via mutual information policies," *Int. J. Robot. Res.*, vol. 38, no. 2/3, pp. 146–161, 2019.

[37] 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.

[38] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[39] C. Belta, B. Yordanov, and E. Aydin Gol, "Finite temporal logic control," in *Formal Methods for Discrete-Time Dynamical Systems*. Berlin, Germany: Springer, 2017, pp. 81–108.

Shuaiyi Li was born in Liaoning, China, in 2001. She received the B.Eng. degree in automation in 2021 from Shanghai Jiao Tong University, Shanghai, China, where she is currently working toward the master's degree. Her research interests include formal methods and task planning.

Mengjie Wei was born in Shanxi, China in 2001. She received the B.Eng. degree in electrical engineering in 2023 from Shanghai Jiao Tong University, Shanghai, China, where she is currently working toward the M.Eng. degree. Her research interests include formal methods and SLAM.

Shaoyuan Li (Senior Member, IEEE) was born in Hebei, China, in 1965. He received the B.S. and M.S. degrees in automation from the Hebei University of Technology, Tianjin, China, in 1987 and 1992, respectively, and the Ph.D. degree from Nankai University, Tianjin, in 1997. Since 1997, he has been with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China, where he is currently a Professor. His research interests include model predictive control, dynamic system optimization, and cyber-physical systems. He is the Vice-President of the Chinese Association of Automation.

Xiang Yin (Member, IEEE) was born in Anhui, China, in 1991. He received the B.Eng. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2012, and the M.S. and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2013, and 2017, respectively. Since 2017, he has been with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China, where he is currently an Associate Professor. His research interests include formal methods, discrete-event systems and cyber-physical systems. Dr. Yin is the Co-Chair of the IEEE CSS Technical Committee on Discrete Event Systems, an Associate Editor for the *Journal of Discrete Event Dynamic Systems: Theory & Applications*, and a Member of the IEEE CSS Conference Editorial Board. Dr. Yin was the recipient of the IEEE Conference on Decision and Control (CDC) Best Student Paper Award Finalist in 2016.