Temporal-Logic-Constrained Hybrid Reinforcement Learning to Perform Optimal Aerial Monitoring with Delivery Drones

Ahmet Semi Asarkaya[1], Derya Aksaray[1], and Yasin Yazıcıoğlu[2]

Abstract— **In this paper, we consider a package delivery drone that is desired to simultaneously perform aerial monitoring as a secondary mission. To integrate this secondary mission, we utilize a reward function representing the value of information gathered via aerial monitoring. We use time window temporal logic (TWTL) specifications to define the pick-up and delivery tasks while utilizing reinforcement learning (RL) to maximize the expected sum of rewards. The high-level decision-making of the drone is modeled as a Markov decision process (MDP). In this regard, we extend the previous work where a model-free RL algorithm was used to solve this optimization problem. We propose a modified Dyna-Q algorithm to address the shortage of online samples. We provide extensive simulation results to compare the performance of the model-free and hybrid RL algorithms in this application and investigate the effect of the different system parameters on the overall performance.**

I. Introduction

Delivery and aerial monitoring are two major applications of unmanned aerial vehicles, or drones, that have been expanding rapidly (e.g., [1], [2], [3], [4]). In many cases, a delivery drone is equipped with a camera and can simultaneously perform aerial monitoring (e.g., infrastructure, environment, traffic) as a secondary task by modifying its route accordingly. Motivated by such a multi-use of drones, we investigate the high-level planning of a delivery drone for performing optimal aerial monitoring while maintaining probabilistic guarantees on the successful completion of its pick-up and delivery tasks as a constraint.

A pick-up and delivery task requires an agent to take a path that visits specific locations in a particular order, usually during specific time windows. Finding such paths can be posed as vehicle routing problems (VRPs), which are typically NP-hard and solved approximately (e.g., [5], [6], [7]). However, standard formulations of VRPs do not yield a compact representation of the feasible trajectory set to be used as a constraint in the proposed aerial monitoring problem. One way to incorporate such trajectories defined by complex specifications is to use temporal logics (TLs), which have been used for control and planning of autonomous systems in various applications (e.g., [8], [9], [10], [11]). In this paper, we utilize time window temporal logic (TWTL) [12], which can be used to encode pick-up and delivery tasks with time-windows.

[1]D. Aksaray and A.S. Asarkaya are with the Department of Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, MN, 55455, daksaray@umn.edu, asark001@umn.edu

[2]Y. Yazıcıoğlu is with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, 55455, ayasin@umn.edu.

In this paper, we consider a high-level planning problem modeled as a Markov decision process (MDP), where the reward encodes the value of data collected in aerial monitoring, under TWTL constraints, which encode the pick-up and delivery tasks. In this formulation, each region in the discretized mission area yields a reward representing the value of monitoring that region. A finite set of pick-up and delivery tasks are defined using TWTL. The drone is randomly assigned a task from this set in each episode. Considering the uncertainty in the system dynamics and the reward distribution over the mission area, we propose a constrained learning problem to find an optimal routing policy for each pick-up and delivery task. Such an optimal policy maximizes the expected sum of rewards while satisfying the corresponding TWTL constraint with a desired probability. This paper extends our prior work in [13], which utilized a modified Q-learning (model-free RL) algorithm to learn optimal routing policies [14]. Here, we propose a modified Dyna-Q (hybrid RL) algorithm that achieves a significantly faster learning rate than the model-free approach in [13]. Such a faster learning rate typically yields a better policy after a finite learning horizon. We demonstrate the performance of the proposed hybrid RL algorithm and compare it to the model-free RL algorithm via extensive simulations.

This paper is organized as follows: Section II presents an overview of related work. Section III reviews the essential background of the basic concepts including TWTL, MDP, and Dyna-Q framework. In Section IV, we state the problem after introducing the system model and the system definition. In Section V, we thoroughly explain the algorithms that are the foundation of the proposed method. Section VI demonstrates the simulation results and discusses the results.

II. Related Work

Model-free reinforcement learning (RL) algorithms can be successfully used to learn optimal policies in MDPs through trial and error (e.g., [15]). Model-free RL algorithms often require a significantly large number of interactions with the environment when the state-space is large. In principle, model-based RL algorithms are considered more data-efficient (e.g., [16], [17]). However, the success of the model-based approaches strongly depends on the environment model, and the trade-off comes into the picture in the form of bias and computational cost [18]. A new breed of RL, also referred as *hybrid RL* (e.g., [19], [20], [21], [22]), has emerged to inherit the advantages of both methods.

In many applications, autonomous agents need to operate under constraints on their trajectories due to safety or critical mission requirements. In such constrained settings, traditional RL algorithms facilitate learning optimal feasible solutions by severely penalizing infeasible actions. However, this is not a viable approach for most physical systems since constraint violations during training may correspond to catastrophic or unacceptable events (e.g., [23], [24]). The area of safe RL is mainly focused on addressing this limitation (e.g., see [25] and the references therein). Safe RL aims to guarantee safety during the training while maximizing the total expected reward. Many different approaches have been proposed to learn the optimal policy under safety considerations. In [26] and [27], the authors offer an algorithm where the exploration phase is modified by incorporating the prior knowledge of the mission to avoid risky actions. Such a biased exploration often leads to a sub-optimal policy, and it may even be an infeasible option for satisfying complex constraints. Some studies (e.g., [28], [29], [30]) apply invariance-based approaches or model predictive control (MPC) for model-based methods to ensure safety constraints, which approximately correspond to a limited family of temporal logic specifications. Authors of [31], [32] propose a switch structure between the backup controllers to guarantee safety. While these approaches may ensure that the probability of violating the constraints falls to zero over time, they do not ensure constraint satisfaction during the learning phase. Some works (e.g., [33]) have introduced barrier functions to avoid unsafe actions during the learning; however, they cannot accommodate complex constraints (e.g., temporal logics). There also exist some safe RL methods that can incorporate linear temporal logic constraints [34], [35] but these works do not provide formal probabilistic guarantees on constraint satisfaction (e.g., not ensuring the probability of constraint satisfaction greater than a desired threshold). To address this limitation, in [14], we have proposed a constrained exploration strategy that achieves bounded TL constraints with a desired probability during RL. In this paper, we build on the method in [14] to develop a constrained Dyna-Q algorithm for achieving optimal aerial monitoring with delivery drones.

III. Preliminaries

A. Time Window Temporal Logic (TWTL)

In this paper, we use TWTL to express pick-up and delivery tasks. This section includes some essential information on TWTL, and the reader is referred to [12] for further details. Let AP be a set of atomic propositions, each of which has a truth value over the state-space. Consider that $AP = \{A\}$ denotes the region A over the state-space. If the system is inside region A, the atomic proposition A becomes true, otherwise, it becomes false. TWTL is defined over the set AP with the following syntax:

$$\phi ::= H^d x | H^d \neg x | \phi_1 \wedge \phi_2 | \phi_1 \vee \phi_2 | \neg \phi_1 | \phi_1 . \phi_2 | [\phi_1]^{[a,b]},$$

where

- x is either the true constant $\top$ or an atomic proposition from AP;
- $\wedge$, $\vee$, and $\neg$ are the conjunction, disjunction, and negation Boolean operators, respectively;
- . is the concatenation operator;
- H^d with d $\in \mathbb{Z}_{\geq 0}$ is the hold operator;
- $[]^{[a,b]}$ with $0 \leq a \leq b$ is the within operator.

The semantics of TWTL are defined according to the finite output words o over an AP, and $o(k)$ refers to the k^{th} element on o. For any $x \in AP$, the *hold* operator $H^d x$ indicates that x should be true (serviced) for d time units (i.e., $o \models H^d x$ if $o(t) = x \ \forall t \in [0,d]$). The *within* operator $[\phi]^{[a,b]}$ means that the satisfaction of ϕ is bounded to the time window $[a,b]$ (i.e., $o \models [\phi]^{[a,b]}$ if $\exists k \in (0, b-a)$ s.t. $o' \models \phi$ where $o' = o(a+k)\dots o(b)$). Finally, the concatenation of ϕ_i and ϕ_j (i.e., $\phi_i \cdot \phi_j$) designates that the first ϕ_i must be satisfied and then immediately after that ϕ_j must be satisfied.

The satisfaction of a TWTL formula is checked within bounded time. We define the *time-bound* of a TWTL formula ϕ (i.e., $||\phi||$) as the maximum time required to satisfy ϕ. The time-bound can be recursively determined as follows:

$$||\phi|| = \begin{cases} max(||\phi_1||, ||\phi_2||) & \text{if } \phi \in \{\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2\} \\ ||\phi_1|| & \text{if } \phi = \neg\phi_1 \\ ||\phi_1|| + ||\phi_2|| + 1 & \text{if } \phi = \phi_1 . \phi_2 \\ d & \text{if } \phi \in \{H^d x, H^d \neg x\} \\ b & \text{if } \phi = [\phi_1]^{[a,b]} \end{cases}$$

The syntax and semantics of TWTL enable one to define rich specifications. For example, *"service region R_1 for 4 time units within [0, 7] and service region R_2 for 1 time unit within [6, 12]"* can be defined as a TWTL formula as follows:

$$\phi_1 = [H^4 R_1]^{[0,7]} \wedge [H^1 R_2]^{[6,12]} \quad (1)$$

Moreover, a pick-up and delivery task can be defined as a TWTL formula:

$$\phi_2 = [H^{10} L_p]^{[0,40]} . [H^{10} L_d]^{[0,40]}, \quad (2)$$

which means that *"Within 40 time units, go to the pick-up location L_p and hold for 10 time units; immediately after that, within 40 time units, go to the delivery location L_d and hold there for 10 time units."* For this TWTL specification, the time-bound becomes $||\phi_2|| = 40 + 40 + 1 = 81$.

TWTL specifications can be temporally relaxed by defining some slack variables added to the time windows of the specification [36]. For example, the relaxed version of the previous pick-up and delivery task ϕ_2 becomes

$$\phi = [H^{10} L_p]^{[0,40+\tau_1]} . [H^{10} L_d]^{[0,40+\tau_2]}, \quad (3)$$

which implies that *"within $(40+\tau_1)$ time units, first go to L_p and wait there for 10 time units; and immediately after that within $(40+\tau_2)$, go to L_d and wait there for 10 time units."* The time-bound of this relaxed specification is parametric and calculated as $80 + \tau_1 + \tau_2 + 1$. Note that if $\tau_i > 0$, then the corresponding time window is extended (i.e., a task is allowed to be finished after its deadline). However, if $\tau_i < 0$, then the corresponding time window is shortened (i.e., a task is allowed to be finished earlier than its deadline). Overall, the temporal relaxation is formally defined as follows:

Definition 1 *($\tau-$ Relaxation of ϕ [12]). Let $\tau \in \mathbb{Z}^m$, where m is the number of within operators contained in ϕ. The $\tau-$relaxation of ϕ is a feasible TWTL formula $\phi(\tau)$, where each subformula of the form $[\phi_i]^{[a_i,b_i]}$ is replaced by $[\phi_i]^{[a_i,b_{i+\tau_i}]}$*

Definition 2 *(Temporal Relaxation). Given a TWTL specification ϕ, let $\phi(\tau)$ be a feasible relaxed formula with a relaxation vector $\tau \in \mathbb{Z}^m$. The temporal relaxation of $\phi(\tau)$ is defined as $|\tau|_{TR} = max_j(\tau_j)$.*

A deterministic finite-state automaton (DFA) is a graphical representation to encode the satisfactory cases of a TWTL specification. This representation mainly holds sufficient information to monitor the progress towards the TWTL satisfaction. Moreover, a DFA can also be constructed for a temporally relaxed TWTL specification.

Definition 3 *(Deterministic Finite-State Automaton)[12] A DFA is a tuple $\mathscr{A} = (\mathscr{Q}, q_{init}, AP, \delta, F_{\mathscr{A}})$ where,*

- *$\mathscr{Q}$ is a finite set of states;*
- *$q_{init} \in \mathscr{Q}$ is the initial state;*
- *AP is the input alphabet;*
- *$\delta : \mathscr{Q} \times AP \times \mathscr{Q} \mapsto \mathscr{Q}$ is the transition function;*
- *$F_{\mathscr{A}} \subseteq \mathscr{Q}$ is the set of accepting states.*

Consider a pick-up and delivery task expressed as a TWTL specification $\phi = [H^0\, pick-up]^{[0,3+\tau_1]}.[H^0\, delivery]^{[0,5+\tau_2]}$, which implies that go to pick-up region within $[0,3+\tau_1]$, and immediately after that go to the delivery zone within $[0,5+\tau_2]$. The corresponding DFA of this relaxed specification is shown in Fig. 1, where q_0 is the initial state, q_2 is the accepting state, and state transitions occur when pick-up and/or delivery are observed. Note that any path that starts from q_0 and ends at q_2 is a satisfactory case. For example, the drone can pick-up the package at the current time t and then deliver it at $t+1$. As depicted in Fig. 1, the path q_0, q_1, q_2 is a satisfactory case over the DFA. Furthermore, a drone that starts at a non-pick-up state (q_0) can spend the first two time steps to reach the pick-up location (staying at q_0 for two time steps), and then pick-up the package (moving to the state q_1), and then spend one time step to go to the delivery location (staying at q_1) and then deliver (moving to q_2). This case is also satisfactory, and we can encode it as a path over the DFA as follows: $q_0, q_0, q_0, q_1, q_1, q_2$.

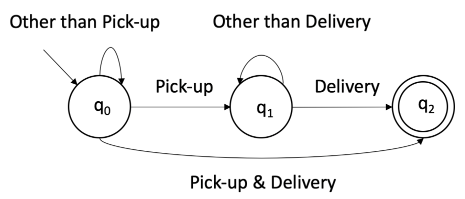


Fig. 1. DFA representation of $[H^0\, pick-up]^{[0,3+\tau_1]}.[H^0\, delivery]^{[0,5+\tau_2]}$

B. Reinforcement Learning

Markov decision process (MDP) is a suitable framework to model the decision-making under stochastic dynamics, and it is formally defined as follows:

Definition 4 *(Markov Decision Process) An MDP is a tuple $M = (S, A, P, R)$ where,*

- *S denotes the state-space;*
- *A is the set of actions;*
- *$P : S \times A \times S \mapsto [0,1]$ is a probabilistic transition relation;*
- *$R : S \mapsto \mathbb{R}$ is a reward function.*

A canonical learning problem in the reinforcement learning literature assumes that the transition probability function P is unknown. The goal is to find an optimal control policy $\pi : S \to A$ that maximizes the expected cumulative reward (i.e., $E\left[\sum_{k=0}^{T} r(s_{k+1})\right]$ or $E\left[\sum_{k=0}^{\infty} \gamma^k r(s_{k+1})\right]$) where *gamma* is the discount factor and $r(s_k)$ is the reward received at state s_k.

Q-learning [37] is a model-free reinforcement learning algorithm that can find the optimal policy through the agent's interaction with the environment. Consider that an agent's decision-making is modeled as an MDP. Starting from a state $s \in S$, the agent chooses an action $a \in A$, which takes it to state s' and leads to a reward $r \in R$. Accordingly, the following update rule is followed,

$$Q_{k+1}(s,a) = (1-\alpha_k) Q_k(s,a) + \alpha_k [r + \gamma \max_{a^* \in A} Q_k(s', a^*)] \tag{4}$$

where $\alpha \in (0,1]$ is the learning rate. It has been shown that if every action $a \in A$ is repeatedly chosen in each state s for an infinite number of times, and α decays appropriately, then $Q-$function converges to the optimal Q^* function with a probability of one [37]. Finally, the optimal policy becomes $\pi^*(s) = \arg\max_{a \in A} Q^*(s,a)$.

Note that Q-learning is a model-free approach that relies only on the real interactions with the environment. While Q-learning has theoretical guarantees for optimality, it requires sufficiently many interactions with the environment to converge to a good solution. Alternatively, there also exist model-based learning approaches, which construct the transition probabilities and reward function from a small set of interactions with the environment [38]. While model-based techniques converge to good solutions faster than model-free techniques, they suffer from the bias of the data used to create the model. Dyna-Q algorithm [38] is a hybrid learning method that integrates Q-learning with a model that is constructed internally. Overall, Dyna-Q relies on both real interactions with the environment and hypothetical interaction through the internal model. Alg. 1 shows the pseudo-code of Dyna-Q, where lines 1-6 are the same steps from Q-learning, line 7 is the update of the internal model form the observed experiences, and lines 8-12 are to sample randomly only from the state-action pairs that have previously experienced.

Algorithm 1: Dyna-Q [38]

```
1 : Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A(s)$ ;
2 : Do forever:
3 :     $s \leftarrow$ current state;
4 :     $a \leftarrow \varepsilon - greedy$;
5 :     Take action $a$, observe the reward $r$ and the next state $s'$;
6 :     $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$;
7 :     $Model(s,a) \leftarrow r, s'$ ;
8 :     Repeat $n_{samples}$ times:
9 :         $s \leftarrow$ random previously observed state;
10 :        $a \leftarrow$ random action previously taken in s;
11 :        $r, s' \leftarrow Model(s,a)$;
12 :        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) - Q(s,a)]$;
```

IV. Problem Statement

System and Environmental Model: In this paper, we model the high-level decision-making of a drone as an MDP, which is a suitable mathematical framework for stochastic decision-making. We assume that the drone can select an action from a set of motion primitives, and the consequence of an action is uncertain. To illustrate this, Fig. 2 shows the consequence of various actions. For example, if the intended action is North (N), there is a high probability that the drone will go up (following blue arrow) but there is a small probability that it can move along NE, NW, or stay (following a red arrow). Overall, the blue arrow arrows indicate the most probable state transition upon taking a particular action while the red arrows indicate the other states that can be unintentionally reached. In this paper, we assume that the agent does not know the true probability distribution of an action.

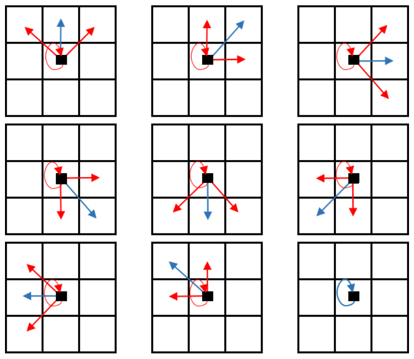


Fig. 2. Blue arrows indicate the intended action, red arrows indicate the uncertainty in the intended action. Intended actions from the top left corner to right bottom corner : N, NE, E, SE, S, SW, W, NW, Stay

We consider an agent moving over a discretized $m \times n$ environment, and each discrete state is an MDP state, $s_i \in S$, $i = \{0, 1, ..., m \times n - 1\}$. The set of actions at each state is $A = \{N, NE, E, SE, S, SW, W, NW, Stay\}$. The probability of transition between the states is denoted by $P : S \times A \times S \mapsto [0,1]$. Finally, each state has a corresponding reward (i.e., $r(s_i) \in \mathbb{R}$ for all s_i) that is according to the value of information gathered via aerial monitoring.

Mission Definition: In this paper, we use relaxed TWTL specification to define pick-up and delivery tasks that can be defined in the following form:

$$\phi_{task} = [H^{t_1}\, Pick]^{[0, T_1+\tau_1]}.[H^{t_2}\, Deliver]^{[0, T_2+\tau_2]}, \quad (5)$$

which means that "go to the pick-up location and stay there for t_1 time units within $[0, T_1 + \tau_1]$, and immediately after that, go to the delivery zone and stay there for t_2 time units in the following $T_2 + \tau_2$ time steps. Note that the time-bound of this relaxed task is $||\phi_{task}|| = T_1 + \tau_1 + T_2 + \tau_2 + 1$ and we will enforce that $T_1 + \tau_1 + T_2 + \tau_2 + 1 = T_1 + T_2 + 1$. Here, $T_1 + T_2 + 1$ corresponds to the time-bound of the specification with no relaxation.

The reason we consider a relaxed specification but enforcing its time-bound to the time-bound of the non-relaxed specification is because of the following reason. Due to uncertainties and disturbances in logistic, air traffic, etc., a package delivery drone might experience delays in real-life scenarios. Consider a case where the drone is assigned to a delivery task with a time-bound of 41, and it is expected to complete the pick-up within the first 20 time units. However, it can take more than 20 time units (say 25 time units) because of the unexpected impediments. In this case, the drone needs to compensate for the time lost during the pick-up. Therefore, it will automatically regenerate a new trajectory to finish the delivery part within 15 time units. As a result, the pick-up and delivery task will still be finished within 41 time units because the time-bound is constrained. Namely, the drone is imposed that any delay during pick-up needs to be compensated for during the delivery. Likewise, the extra time can be spent for the delays during the delivery part when the pick-up part takes less time than expected.

Problem Statement: A reinforcement learning problem under dynamically arriving TWTL tasks has been introduced in [13], which is formally defined as follows:

Problem 1 *[13] Given an MDP $M = (S, A, P, R)$ modeling the high-level decision-making of a drone, the transition probabilities P is initially unknown and $r(s(t)) \in \mathbb{R}$ is the reward observed at state $s(t) \in S$. Let Φ_{tasks} be a set of TWTL specifications defining the potential pick-up and delivery tasks where each task has a time-bound of T. The order of arriving tasks are initially unknown to the drone and each task arrives to the environment in periods of T. The objective is to learn the optimal control policy $\pi^* : S \rightarrow A$ that maximizes the expected sum of rewards while also ensuring that, at each episode, the resulting trajectories satisfy the corresponding TWTL formula with a probability of more than a desired threshold $Pr_{des} \in (0,1)$, i.e.,*

$$\begin{aligned} \pi^* = \arg\max_{\pi} E^{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r(s(t))\Big] \quad & \textit{such that} \\ Pr\big(o_n \models \phi_n(\tau_n)\big) \geq Pr_{des}, \quad & \forall \phi_n(\tau_n) \in \Phi_{tasks}, \\ ||\phi_n(\tau_n)|| \leq T, \quad & \forall n \in \{1, ..., N_{episode}\}, \end{aligned} \quad (6)$$

where $\phi_n(\tau_n)$ represents a task arriving at each episode, $o_n = l(s(1)), ..., l(s(T))$ is the output word of the episode in which $s_n = s(1)s(2)\ldots s(T)$ is the state trajectory of episode n and $l(.) : S \rightarrow AP$ is the labeling function, and $N_{episode}$ is the total number of episodes.

While we use the same problem formulation as in [13], in this paper, we propose a novel and efficient solution to it.

V. Proposed Method

In this section, we build a framework, where we combine the Dyna Q-learning algorithm and the definition of the TWTL, to solve Problem 1. First, we construct a product MDP between an MDP and a DFA. A product MDP encodes the feasible movements over the physical environment as well as keeping the track of the TL constraint satisfaction. Each state in a product MDP contains information on both MDP and DFA states. We formally define it as follows:

Definition 5 *(Product MDP) Given the DFA of a TWTL formula $\mathscr{A} = (\mathscr{Q}, q_{init}, AP, \delta, F_{\mathscr{A}})$, an MDP $M = (S, A, P, R)$, a set of atomic propositions AP, and a labelling function $l : S \mapsto AP$, a product MDP is a tuple $\mathscr{P} = M \times \mathscr{A} = (S_{\mathscr{P}}, p_{init}, A, \Delta, R_{\mathscr{P}}, F_{\mathscr{P}})$ where,*

- *$S_{\mathscr{P}} = S \times \mathscr{Q}$ is a finite set of states;*
- *$p_{init} = S \times \{q_{init}\} \in S_{\mathscr{P}}$ is the set of initial state;*
- *A is the set of actions;*
- *$\Delta : S_{\mathscr{P}} \times A \times S_{\mathscr{P}} \mapsto [0,1]$ is the probabilistic transition relation such that $\Delta(p,a,p') = P(s,a,s')$ and $\delta(q,l(s)) = q'$ for an action $a \in A$ and two product MDP states $p = (s,q) \in S_{\mathscr{P}}$ and $p' = (s',q') \in S_{\mathscr{P}}$;*
- *$R_{\mathscr{P}} : S_{\mathscr{P}} \mapsto \mathbb{R}$ is the reward function such that $R_{\mathscr{P}}(p) = R(s)$ for $p = (s,q) \in S_{\mathscr{P}}$;*
- *$F_{\mathscr{P}} = (S \times F_{\mathscr{A}}) \subseteq S_{\mathscr{P}}$ is the set of accepting states.*

During a mission, it is crucial to track the remaining time because we consider time-bounded missions. However, a product MDP does not accommodate any information about the remaining mission time. To address this, we define a time-product MDP that also holds the time information. Therefore, a time-product MDP state encodes a physical state, the TL task progression, and the current time. The formal definition of a time-product MDP is given as follows:

Definition 6 *(Time-Product MDP) Given a product MDP $\mathscr{P} = (S_{\mathscr{P}}, p_{init}, A, \Delta, R_{\mathscr{P}}, F_{\mathscr{P}})$ and a time set $\mathscr{T} = \{1,..,t,..,T\}$, a time-product MDP is a tuple $\mathscr{P}^{\mathscr{T}} = \mathscr{P} \times \mathscr{T} = (S^{\mathscr{T}}_{\mathscr{P}}, p^{\mathscr{T}}_{init}, A, \Delta^{\mathscr{T}}, R^{\mathscr{T}}_{\mathscr{P}}, F^{\mathscr{T}}_{\mathscr{P}})$ where,*

- *$S^{\mathscr{T}}_{\mathscr{P}} = S_{\mathscr{P}} \times \mathscr{T}$ is a finite set of states;*
- *$p^t = (p,t) \in S^{\mathscr{T}}_{\mathscr{P}}$ is a time-product MDP state at the time instance $t \in \mathscr{T}$;*
- *$p^{\mathscr{T}}_{init} = p_{init} \times \mathscr{T}$ is the set of initial states where $p^{\mathscr{T}}_{init} \subseteq S^{\mathscr{T}}_{\mathscr{P}}$;*
- *A is the set of actions;*
- *$\Delta^{\mathscr{T}} : S^{\mathscr{T}}_{\mathscr{P}} \times A \times S^{\mathscr{T}}_{\mathscr{P}} \mapsto [0,1]$ is the probabilistic transition relation such that $\Delta^{\mathscr{T}}(p^t, a, p^{t+1}) = \Delta(p,a,p')$ for an action $a \in A$ and two time-product MDP states $p^t = (p,t)$ and $p^{t+1} = (p', t+1) \in S^{\mathscr{T}}_{\mathscr{P}}$;*
- *$R^{\mathscr{T}}_{\mathscr{P}} : S^{\mathscr{T}}_{\mathscr{P}} \mapsto \mathbb{R}$ is the reward function such that $R^{\mathscr{T}}_{\mathscr{P}}(p^t) = R_{\mathscr{P}}(p)$ for $p^t = (p,t) \in S^{\mathscr{T}}_{\mathscr{P}}$;*
- *$F^{\mathscr{T}}_{\mathscr{P}} = (F_{\mathscr{P}} \times \mathscr{T}) \subseteq S^{\mathscr{T}}_{\mathscr{P}}$ is the set of accepting states.*

Overall, a time-product MDP state contains all the required information to achieve constrained RL during a mission, i.e., $p^t = (p,t) = (q,s,t)$ where p^t is a time-product MDP state at the current time t, p is the corresponding product MDP state, s is the corresponding MDP state, and q is the corresponding DFA state.

The proposed method consists of offline and online parts. In the offline part, we construct the time-product MDPs for each pick-up and delivery task $\phi_i \in \Phi_{tasks}$. We utilize Alg. 2, proposed in [14], to quantify the worst case probability of constraint satisfaction in the remaining k time steps from a given time-product automaton state. The feasible next states are determined for each action a that can be taken at each state p^t (line 8 in Alg. 2). An action a is pruned from the feasible action set of p^t, i.e., $Act(p^t)$, if the worst case satisfaction probability of any state in the set of potential next states is smaller than the desired probability (lines 12-13 in Alg. 2). Alg. 2 relies on the assumption in [14, Assumption 1]. More specifically, given some $\varepsilon \in [0,1]$, it is assumed that 1) the transitions that occur with a probability of at least $1-\varepsilon$ are known, and 2) each feasible transition can be reverted in the next time step with a probability of at least $1-\varepsilon$. Note that knowing the transitions that occur with a probability of at least $1-\varepsilon$ does not require knowing the actual transition probabilities. Instead, it only requires knowing which transitions are sufficiently likely (depending on ε) to occur for each state-action pair. For example, for a mobile agent with the actions as in Fig. 2, suppose that each action (e.g., "move up") results in moving to the desired cell with a probability of 0.9 or to another adjacent cell with a probability of 0.1. While the actual values of these probabilities are unknown, some prior information may indicate that, for each action, the only transition that occurs with a probability of at least 0.7 ($\varepsilon = 0.3$) is moving to the desired cell. In that case, any feasible transition can also be reverted with of probability of at least 0.7 by choosing to move back to the previous location in the next time step.

Algorithm 2: Offline construction of the pruned time-product MDP [14]

```
Input: $T$ (episode length), $\varepsilon$ (estimated motion uncertainty)
Input: $M$ (MDP), $\Phi$ (temporal logic task), $Pr_{des}$ (desired satisfaction probability)
Output: $\mathscr{P}^{\mathscr{T}}$ (pruned time-product MDP)
1 : Create FSA of $\phi$, $\mathscr{A} = (\mathscr{Q}, q_{init}, AP, \delta, F_{\mathscr{A}})$;
2 : Create product MDP, $\mathscr{P} = M \times \mathscr{A} = (S_{\mathscr{P}}, p_{init}, A, \Delta, R_{\mathscr{P}}, F_{\mathscr{P}})$;
3 : Create time-product MDP, $\mathscr{P}^{\mathscr{T}} = \mathscr{P} \times \{0,\ldots,T-1\} = (S^{\mathscr{T}}_{\mathscr{P}}, p^{\mathscr{T}}_{init}, A, \Delta^{\mathscr{T}}, R^{\mathscr{T}}_{\mathscr{P}}, F^{\mathscr{T}}_{\mathscr{P}})$;
4 : Calculate the distance-to-$F_{\mathscr{P}}$, i.e., $d^{\varepsilon}(p^t)$ for all $p^t \in S^{\mathscr{T}}_{\mathscr{P}}$;
5 : Initialization: $Act(p^t) = A$ for all $p^t \in S^{\mathscr{T}}_{\mathscr{P}}$
6 :  for each non-accepting state $p^t \in S^{\mathscr{T}}_{\mathscr{P}} \setminus F^{\mathscr{T}}_{\mathscr{P}}$
7 :     for each action $a \in Act(p^t)$
8 :        Find $N(p^t,a) = \{(r,t+1) | \Delta(p,a,r) > 0\}$ (states reachable from $p^t$ under $a$);
9 :        $d_{max} = \max_{r^{t+1} \in N(p^t,a)} d^{\varepsilon}(r^{t+1})$;
10 :        $k = T - t - 1$ (the remaining episode time);
11 :        $i_{max} = \lfloor \frac{k-1-d_{max}}{2} \rfloor$;
12 :        if ($i_{max} \geq 0$ and $\sum_{i=0}^{i_{max}} \frac{(k-1)!}{(k-1-i)!i!} \varepsilon^i (1-\varepsilon)^{k-1-i} < Pr_{des}$) or ($i_{max} < 0$)
13 :           $Act(p^t) = Act(p^t) \setminus \{a\}$ ;
14 :        end if
15 :     end for
16 :  end for
17 : $\mathscr{P}^{\mathscr{T}} = (S^{\mathscr{T}}_{\mathscr{P}}, p^{\mathscr{T}}_{init}, Act : S^{\mathscr{T}}_{\mathscr{P}} \to A, \Delta^{\mathscr{T}}, R^{\mathscr{T}}_{\mathscr{P}}, F^{\mathscr{T}}_{\mathscr{P}})$;
```

We introduce Alg. 3 based on Alg. 2 to create a set of time-product MDPs corresponding to the tasks in the set Φ_{task} because each of them has a different DFA. While it is not known in which order the tasks are arriving, with the knowledge of Φ_{task} set, we construct the time-product MDPs in an offline fashion. Lines 1-3 in Alg. 2 construct the product MDPs, DFAs, and time-product MDPs for each task in Φ_{task} by using the off-the-shelf tool proposed in [12]. Line 4 calculates the distances corresponding to each time-product MDP, which is the measure of the task progress at any given time-product MDP state as defined in [14, Def. 5]. Then, lines 6-19 of Alg. 2 are called to create the pruned action spaces of each state of the time-product MDPs.

```
Algorithm 3: Offline construction of the time-product MDPs
Input: $T$ (episode length), $\varepsilon$ (estimated worst case motion uncertainty), $M$ (MDP)
Input: $\Phi_{tasks}$ (pick-up and delivery tasks), $Pr_{des}$ (desired satisfaction probability)
Output: $\mathscr{P}^{\mathscr{T}}_{\phi_i}$ (pruned time-product MDPs) for every $\phi_i \in \Phi_{tasks}$
1 : Create corresponding DFAs for every task $\phi_i \in \Phi_{tasks}$ : $\mathscr{A}_{\phi_i} = (\mathscr{Q}_{\phi_i}, q_{init,\phi_i}, AP_{\phi_i}, \delta_{\phi_i}, F_{\mathscr{A}_{\phi_i}})$;
2 : Create corresponding product MDPs: $\mathscr{P}_{\phi_i} = M \times \mathscr{A}_{\phi_i} = (S_{\mathscr{P}_{\phi_i}}, p_{init,\phi_i}, A_{\phi_i}, \Delta_{\phi_i}, R_{\mathscr{P}_{\phi_i}}, F_{\mathscr{P}_{\phi_i}})$;
3 : Create time-product MDPs, $\mathscr{P}^{\mathscr{T}}_{\phi_i} = \mathscr{P}_{\phi_i} \times \{0,\ldots,T-1\} = (S^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}, p^{\mathscr{T}}_{init,\phi_i}, A_{\phi_i}, \Delta^{\mathscr{T}}_{\phi_i}, R^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}, F^{\mathscr{T}}_{\mathscr{P}_{\phi_i}})$;
4 : Calculate the distance-to-$F_{\mathscr{P}_{\phi_i}}$, i.e., $d^{\varepsilon}_{\phi_i}(p^t_{\phi_i})$ for all $p^t_{\phi_i} \in S^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}$ based on [14];
5 : Initialization: $Act_{\phi_i}(p^t_{\phi_i}) = A_{\phi_i}$ for all $p^t_{\phi_i} \in S^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}$ and $\phi_i \in \Phi_{tasks}$
6 : for $i_{\phi_i} = 1 : N_{\Phi_{tasks}}$
7 :     Iterate over lines 6 - 19 of Algorithm 2;
8 :     $\mathscr{P}^{\mathscr{T}}_{\phi_i} = (S^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}, p^{\mathscr{T}}_{init,\phi_i}, Act_{\phi_i} : S^{\mathscr{T}}_{\mathscr{P}_{\phi_i}} \to A_{\phi_i}, \Delta^{\mathscr{T}}_{\phi_i}, R^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}, F^{\mathscr{T}}_{\mathscr{P}_{\phi_i}})$;
9 : end for
```

In the online part of the proposed method, we introduce Alg. 4, where we implement a Dyna Q-learning algorithm under persistent TWTL constraints. In line 3, a task ϕ_i from the set of Φ_{tasks} is randomly assigned at the beginning of each episode. At any time-product MDP state $p^t_{\phi_i}$, the agent selects an action based on the policy π^{ε}_{GO} if the feasible action set $Act(p^t_{\phi_i})$ is empty (line 7). Here, the policy π^{ε}_{GO} results in the selection of an action that leads to the most progress from the current state to the constraint satisfaction under the most likely state transitions (i.e., minimizing the distance to the set of accepting states under ε-stochastic transitions as defined in [14, Def. 6]). Otherwise, the agent selects an action from $Act(p^t_{\phi_i})$ (line 9). Accordingly, the agent observes the reward r and the next state $p^{t+1}_{\phi_i}$ to update the Q-table using the equation (4) (lines 11-12). Then, the agent updates the model (line 13) and does the planning based on the model (lines 14-19). Under this learning algorithm, the Q-table converges to its true value as the number iterations goes to infinity (e.g., see [37] for the convergence proof).

VI. Simulation Results and Discussion

In this section, we discuss the simulation results, implemented using Python 2.7, on a PC with an Intel i7-7700HQ CPU at 2.8 GHz processor and 16.0 GB RAM. It is assumed that it takes one time unit to move from a cell to an adjacent cell. In [13], the authors propose a model-free RL approach to solve Problem 1. They present a case study where an agent learns the optimal policy for a monitoring mission while it persistently completes the arriving pick-up and delivery tasks. In this paper, we propose a hybrid RL approach to improve the sample efficiency. We conduct extensive simulations to reveal the performance difference between Q-learning and Dyna-Q architecture. We also investigate the effect of the action uncertainty on the reward optimization, and the scalability of the algorithm.

```
Algorithm 4: Dyna-Q learning under persistent TWTL constraints
Input: Pruned time-product MDPs $\mathscr{P}^{\mathscr{T}}_{\phi_i} = (S^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}, p^{\mathscr{T}}_{init,\phi_i}, Act_{\phi_i} : S^{\mathscr{T}}_{\mathscr{P}_{\phi_i}} \to A_{\phi_i}, \Delta^{\mathscr{T}}_{\phi_i}, R^{\mathscr{T}}_{\mathscr{P}_{\phi_i}}, F^{\mathscr{T}}_{\mathscr{P}_{\phi_i}})$
for every $\phi_i \in \Phi_{tasks}$
Output: $\pi$ (policy maximizing the sum of rewards under TWTL constraints)
1 : Initialization: Initial $Q-$table
2 : for $ep = 1 : N_{episode}$
3 :     Assign a random task $\phi_i$ from $\Phi_{tasks}$;
4 :     $p^t_{\phi_i} = (p_{init,\phi_i}, 0)$;
5 :     for $t = 0 : T-1$
6 :         if $Act(p^t_{\phi_i}) = \emptyset$
7 :             $a = \pi^{\varepsilon}_{GO}(p^t_{\phi_i})$;
8 :         else
9 :             Select an action $a$ from $Act_{\phi_i}(p^t_{\phi_i})$ via $\varepsilon-$greedy or $\pi$;
10 :        end if
11 :        Take the action $a$, observe the reward $r$, and the next state $p^{t+1}_{\phi_i} = (p_{\phi_i}, t+1)$;
12 :        $Q(p^t_{\phi_i}, a) = (1-\alpha_{ep})Q(p^t_{\phi_i}, a) + \alpha_{ep}\left[r + \gamma \max_{a'} Q(p^{t+1}_{\phi_i}, a')\right]$;
13 :        $Model(p^t_{\phi_i}, a') \leftarrow r, p^{t+1}_{\phi_i}$ ;
14 :        for $= 1 : N_{samples}$
15 :            $p^{t_r}_{\phi_i} \leftarrow$ random previously observed state;
16 :            $a_r \leftarrow$ random action previously taken in $p^{t_r}_{\phi_i}$;
17 :            $r_r, p^{t_r+1}_{\phi_i} \leftarrow Model(p^{t_r}_{\phi_i}, a_r)$;
18 :            $Q(p^{t_r}_{\phi_i}, a_r) = (1-\alpha_{ep})Q(p^{t_r}_{\phi_i}, a_r) + \alpha_{ep}\left[r_r + \gamma \max_{a'_r} Q(p^{t_r+1}_{\phi_i}, a'_r)\right]$;
19 :        end for
20 :        $\pi(p^t_{\phi_i}) = \arg\max_a, Q(p^t_{\phi_i}, a))$;
21 :        $p^t_{\phi_i} = p^{t+1}_{\phi_i}$;
22 :    end for
23 : end for
```

A. Randomly Generated Pick-up and Delivery Locations

The objective of this section is to explore the effects of the different environments on the performance of the Dyna-Q algorithm. We create a 7×7 environment with obstacles and reward regions as in Fig. 3 (a). Given two pick-up and two delivery locations (p_1, p_2, d_1, and d_2), we define a set of pick-up and delivery missions consisting of the combinations of these locations as in $\Phi_{tasks} = \{\phi_{p_1d_1}, \phi_{p_1d_2}, \phi_{p_2d_1}, \phi_{p_2d_2}\}$. For example, $\phi_{p_1d_1}$ encodes a pick-up and delivery mission where the pick-up and delivery locations are p_1 and d_1, respectively. Accordingly, we randomly generate pick-up and delivery locations to create ten different sets of pick-up and delivery missions, i.e., $\Phi_{all} = \{\Phi_{tasks_1}, \ldots, \Phi_{tasks_n}, \ldots, \Phi_{tasks_{10}}\}$ for $n = \{1, \ldots, 10\}$. Here, each Φ_{tasks_n} contains four pick-up and delivery tasks $\{\phi_{p_1d_1}, \phi_{p_1d_2}, \phi_{p_2d_1}, \phi_{p_2d_2}\}$ for randomly generated p_1 and d_1. Thus, Φ_{all} contains ten sets of four randomly generated pick-up and delivery tasks. Based on the previously defined family of TWTL specifications for pick-

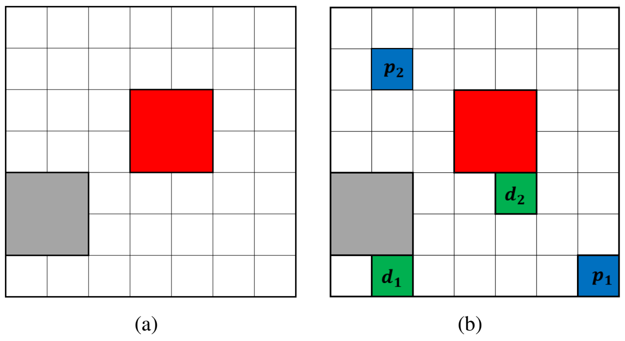


Fig. 3. A grid environment where blue, green, red cells respectively, represent the pick-up regions, the delivery regions, and the obstacles. Gray cells are the regions where monitoring is rewarded. (a) Simulation environment with obstacles and reward regions, (b) Φ_{tasks_2}, an example of randomly generated pick-up and delivery pairs.

up and delivery tasks, a generic pick-up and delivery mission for each combination of a pick-up and delivery locations is defined as

$$\phi_{p_i d_j} = [H^1 \, p_i]^{[0,10+\tau_p]} . [H^1 \, d_j]^{[0,10+\tau_d]}, \quad i, j \in \{1, 2\}, \quad (7)$$

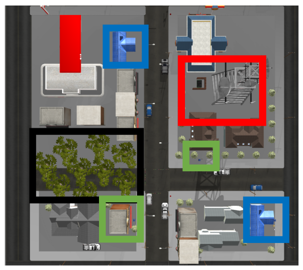
and we enforce the time-bound $||\phi_{p_i d_j}|| \leq 10 + 10 + 1 = 21$. The specification $\phi_{p_i d_j}$ with its time-bound restriction means that *"Within 21 time units, first go to a pick-up region p_i and hold there for 1 time unit. Then, go to a delivery region d_j and hold there for 1 time unit."*. A complete environment with a random example of pick-up and delivery pairs, Φ_{tasks_2} is shown in Fig. 3 (b).

Figure 4 demonstrates a realistic scenario, constructed using the ROS package in [39], corresponding to the abstracted 7×7 environment in Fig. 3 (b). While the package contains different drone dynamical models, we specifically use Asctec Firefly micro air vehicle (MAV) in our high-fidelity simulations (see [40] for specifications). In the considered scenario, the drone picks-up the package from a post-office (blue regions) and delivers it to a mailbox (green regions) while avoiding the flight near to the radio tower (red region). It also performs environmental monitoring around the oak tree region (black region) as it executes its pick-up and delivery mission. Our proposed algorithm generates the sequence of waypoints, and the drone follows these waypoints in the high-fidelity simulation.

We define a measure, $\beta_{@episode}$ as in (8) to compare the sample efficiency between the model-free and hybrid learning methods over a certain number of episodes. Higher $\beta_{@episode}$ values indicate that Dyna-Q performs better. If $\beta_{@episode} = 1$, then both algorithms have the same sample efficiency over the given number of episodes. For example, $\beta_{@10000}$ would indicate the relative sample efficiencies of Dyna-Q architecture and Q-learning over 10000 episodes.

$$\beta_{@episode} = \left(\frac{\text{episode reward using - Dyna-Q}}{\text{episode reward using - Q-learning}} \right)_{@episode} \quad (8)$$

We simulate each ten different task set in Φ_{all} by implementing both model-free and hybrid learning approaches over 100000 episodes. We repeat each simulation ten times to compare the average results. Important parameters used in the simulations are summarized in Table I.



(a)

(b)

Fig. 4. A small-city environment where blue, green, red rectangles respectively, represent the pick-up regions, the delivery regions, and the obstacles. Black rectangle is the region where monitoring is rewarded. (a) Bird's eye view of the environment (b) A snapshot taken during the pick-up.

In Table I, Pr_{des} is the desired probability of constraint satisfaction, ε is the algorithm parameter (we assume that we know the set of states reachable under a taken action with a probability greater than $1 - \varepsilon$), $N_{samples}$ is the number of samples used for the model-based learning, $N_{episodes}$ is the total number of episodes, and $\varepsilon - greedy$ is the parameter for ε-greedy policy followed in line 9 of Alg. 4. Note that since a task $\phi_{p_i d_j} \in \Phi_{tasks}$ is randomly assigned at each episode (Alg. 4, line 3), and Φ_{tasks} has in total four pick-up and delivery tasks, each task $\phi_{p_i d_j}$ are simulated over 25000 ($N_{episodes}/4$) episodes on average.

Pr_{des}	ε	$N_{samples}$	$N_{episodes}$	$\varepsilon - greedy$
0.85	0.05	10	100 000	0.1

TABLE I
SIMULATION PARAMETERS OF DYNA-Q VS Q-LEARNING

The simulation results are summarized in Table II. The results show that Dyna-Q performs better in all cases. On the one hand, we observe that $\beta_{@25000}$ is very close to 1 in the same cases which indicates that the performance of the algorithms is very close to each other in these cases. On the other hand, there are significant differences in some cases. For example, in Φ_{tasks_2} illustrated in Fig. 3, we observe that the sample efficiency of the algorithms are is close to each other for the tasks $\phi_{p_1 d_1}$ and $\phi_{p_2 d_1}$. The detailed results for this case are shown in Fig. 5. Considering the task $\phi_{p_2 d_1}$, the agent needs to visit both p_2 and d_1 to satisfy the task. The proposed algorithm provides a probabilistic satisfaction guarantee of the task. Therefore, the agent needs to pass through the states adjacent to p_2 and d_1. Since the same satisfaction guarantee is provided in both Dyna-Q and Q-learning, and d_1 is adjacent to a reward region, this leads the agent to quickly learn the locations of the reward regions regardless of the method used. Consequently, we observe that $\beta_{@25000}$ value is 1.02 for the task $\phi_{p_2 d_1}$.

We observe a similar condition in the case of $\phi_{p_1 d_1} \in \Phi_{tasks2}$ that also yields $\beta_{@25000}$=1.02. In other words, even when the agent randomly wanders at the initial episodes, it is highly possible that it will hit a reward region. Then, the value function will be updated accordingly. This will cause the sample efficiencies to be very similar in both methods. Likewise, we observe such conditions in other task sets (e.g., $\phi_{p_2 d_2}$ in Φ_{tasks_3} and $\phi_{p_1 d_2}$ in Φ_{tasks_9}). However, the benefit of using Dyna-Q is revealed when the reward is relatively far away from both the pick-up and delivery locations. For example, the task $\phi_{p_1 d_2} \in \Phi_{tasks_2}$ (see Fig. 3 (b)) has $\beta_{@25000} = 1.44$ that indicates Dyna-Q performing significantly better. In this case, the agent needs extra effort to discover the location of the reward region because p_1 and d_2 are very close to each other, i.e., it is likely that the task can be completed without visiting any reward regions when the agent has no information about the environment at the initial episodes. As a result, extra sampling offered by the Dyna-Q architecture accelerates the learning process and yields a higher $\beta_{@25000}$ value.

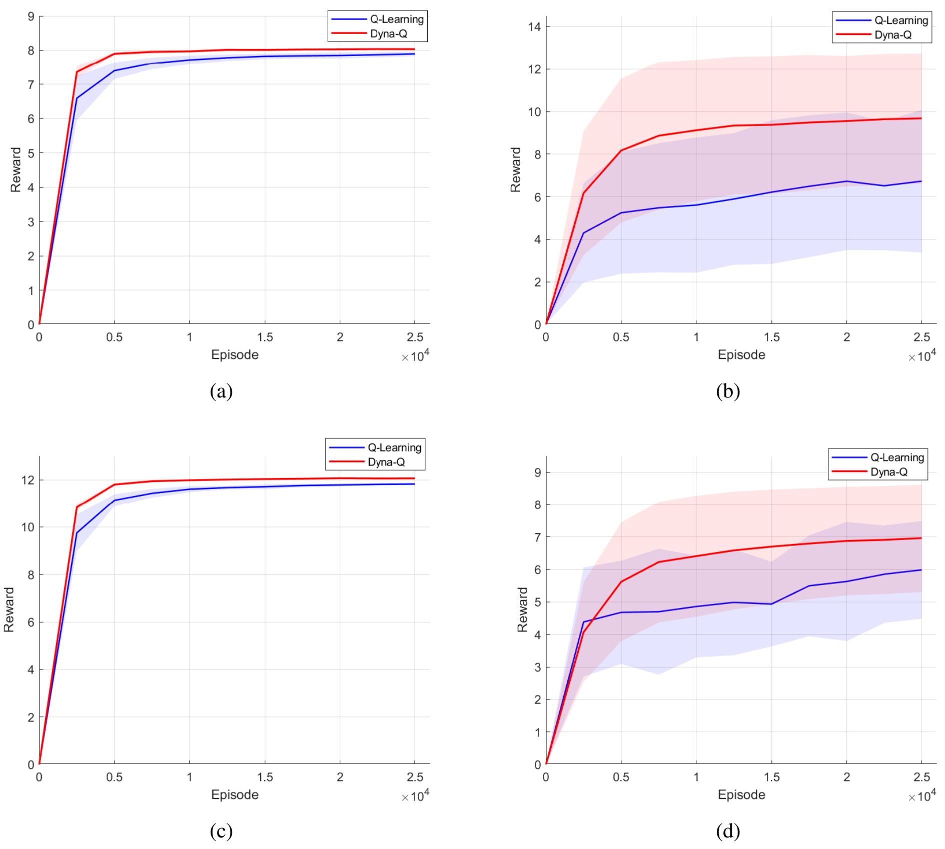


Fig. 5. Comparison of Dyna-Q and Q-learning for Φ_{tasks_2}. Pick-up and delivery tasks: (a) $\phi_{p_1 d_1}$, (b) $\phi_{p_1 d_2}$, (c) $\phi_{p_2 d_1}$, and, (d) $\phi_{p_2 d_2}$. The results are smoothed over 2500 episodes.

	Φ_{tasks_1}	Φ_{tasks_2}	Φ_{tasks_3}	Φ_{tasks_4}	Φ_{tasks_5}	Φ_{tasks_6}	Φ_{tasks_7}	Φ_{tasks_8}	Φ_{tasks_9}	$\Phi_{tasks_{10}}$
$\phi_{p_1 d_1}$	1.24	1.02	1.34	1.41	1.07	1.12	1.03	1.04	1.41	1.62
$\phi_{p_1 d_2}$	1.38	1.44	1.28	1.15	1.25	1.18	1.26	1.12	1.01	1.37
$\phi_{p_2 d_1}$	1.04	1.02	1.38	1.30	1.14	1.01	1.18	1.27	1.04	1.08
$\phi_{p_2 d_2}$	1.08	1.16	1.01	1.14	1.11	1.44	1.16	2.19	1.11	1.03

TABLE II

$\beta_{@25000}$ VALUES FOR EACH TASK SET AND CORRESPONDING PICK-UP AND DELIVERY MISSIONS

Size	$N_{samples}$	# of product MDP states of each task	# of episodes	# of tasks	Episode length	Avg. time to calculate each distance function	Total run time
200×200	10	79993	100 000	25	201	40 [mins]	1240 [mins]

TABLE III

SIMULATION PARAMETERS OF THE LARGE ENVIRONMENT

B. *Effect of the Algorithm Parameter ε*

The proposed algorithm probabilistically guarantees the successful completion of the pick-up and delivery task while optimizing the expected sum of rewards. This is achieved by calculating the worst case probability of satisfaction based on the algorithm parameter ε. Note that, in the case study, we assume that each action in Fig. 2 leads to following the blue arrow with a probability of $1-\varepsilon_{unc}$ and one of the red arrows with a probability of ε_{unc}. We assume that the value of ε_{unc} is unknown and $\varepsilon \geq \varepsilon_{unc}$ is an overestimation of it. When ε is much greater than ε_{unc}, our algorithm gets more cautious and gives initial priority to constraint satisfaction as it considers that the system has higher uncertainty.

In Fig. 6, we demonstrate some simulation results to show how different values of ε_{unc} affect the collected rewards. We randomly generate an environment with four pick-up and delivery location pairs with the parameters with $Pr_{des} = 0.75$, $N_{epsiodes} = 100000$, $N_{samples} = 10$, $\varepsilon_{unc} = 0.03$. We repeat each case ten times to compare the average results.

The results show that as ε increases, the collected rewards always decrease. This indicates that the agent gives more priority to constraint satisfaction to guarantee the probability

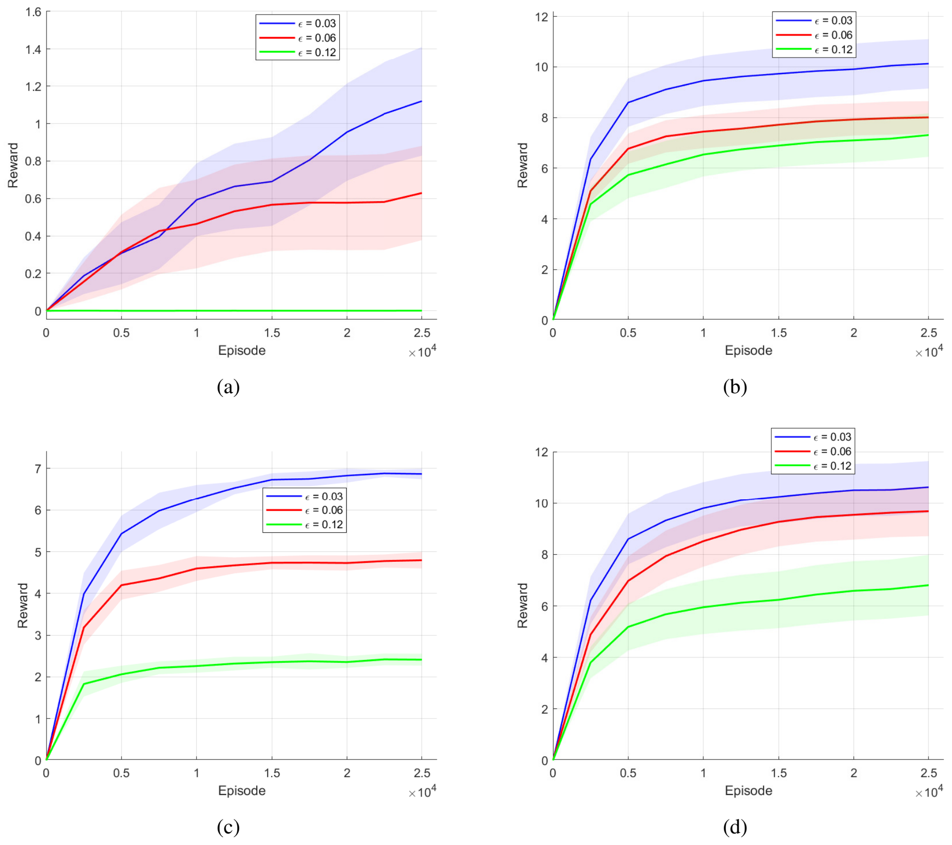


Fig. 6. Effect of the algorithm parameter ε on the collected rewards. Pick-up and delivery tasks of a randomly generated set Φ_{tasks}: (a) $\phi_{p_1 d_1}$, (b) $\phi_{p_1 d_2}$, (c) $\phi_{p_2 d_1}$, and, (d) $\phi_{p_2 d_2}$. Results are smoothed over 2500 episodes.

of satisfaction at least 75%. In other words, it eliminates exploration due to the overestimation of the uncertainty in actions, which in return, leads to the decline in the collected rewards. Therefore, the selection of algorithm parameter ε needs to be handled attentively.

C. *Scalability*

We formulate the problem using a discretized environment while the real environments are most often continuous. Discretization of a continuous environment may result in significantly large number of states depending on the fineness of the discretization. Therefore, we conduct a simulation in a large environment to investigate the scalability of the algorithm with the parameters summarized in Table III. We observe that the most of the computation cost originates from the calculation of the distance function (performed offline). We randomly created 25 pairs of pick-up and delivery locations. Each requires around 40 minutes, in total 1000 minutes, to calculate the distance functions. Recalling that the distance function is calculated off-line, and we run the simulations on a regular laptop, the total run time can significantly be reduced using supercomputers.

VII. Conclusion

We considered the problem of learning optimal routing policies for delivery drones to perform optimal aerial monitoring while successfully completing their pick-up and delivery tasks with a desired probability. We formulated this problem as a Markov decision process under temporal logic (TWTL) constraints encoding the pick-up and delivery tasks. We presented a modified Dyna-Q algorithm, which adjusts the set of allowable actions as needed based on a conservative estimation of the probability of satisfying the TWTL constraint in the remaining time. Accordingly, the proposed algorithm can be used to learn optimal routing policies with probabilistic guarantees on constraint satisfaction throughout the learning process. Furthermore, such a hybrid reinforcement learning algorithm achieves a faster learning rate compared to model-free methods (e.g., Q-learning) due to its improved sample efficiency. We demonstrated the performance of the proposed algorithm and compared it to a model-free approach [13] via extensive simulations.

As a future direction, we aim to extend our proposed approach to systems with continuous states and actions using methods such as deep Q-learning. Furthermore, we aim to extend our methods to dynamic environments, where the constraint satisfaction requires reaching the accepting states on a time-varying graph (time-product MDP) (e.g., [41]). We are also interested in extending our methods to multi-UAV systems where the vehicles are coupled through their objectives and constraints (e.g., [42], [43]).

References

[1] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, vol. 91, pp. 99–131, 2017.

[2] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic uav and ugv system for precision agriculture," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1498–1511, 2016.

[3] S. Seyedi, Y. Yazıcıoğlu, and D. Aksaray, "Persistent surveillance with energy-constrained uavs and mobile charging stations," *IFAC-PapersOnLine*, vol. 52, no. 20, pp. 193–198, 2019.

[4] D. Aksaray, Y. Yazıcıoğlu, E. Feron, and D. N. Mavris, "Message-passing strategy for decentralized connectivity maintenance in multiagent surveillance," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 3, pp. 542–555, 2016.

[5] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[6] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[7] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European journal of operational research*, vol. 59, no. 3, pp. 345–358, 1992.

[8] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE, 2009, pp. 2222–2229.

[9] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE, 2009, pp. 5997–6004.

[10] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *IEEE Conference on Decision and Control (CDC)*, 2016, pp. 6565–6570.

[11] D. Aksaray, K. Leahy, and C. Belta, "Distributed multi-agent persistent surveillance under temporal logic constraints," *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 174–179, 2015.

[12] C.-I. Vasile, D. Aksaray, and C. Belta, "Time window temporal logic," *Theoretical Computer Science*, vol. 691, pp. 27–54, 2017.

[13] A. S. Asarkaya, D. Aksaray, and Y. Yazıcıoğlu, "Persistent aerial monitoring under unknown stochastic dynamics in pick-up and delivery missions," in *AIAA Scitech 2021 Forum*, 2021, p. 1125.

[14] D. Aksaray, Y. Yazıcıoğlu, and A. S. Asarkaya, "Probabilistically guaranteed satisfaction of temporal logic constraints during reinforcement learning," *arXiv preprint arXiv:2102.10063*, 2021.

[15] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic, "Control synthesis from linear temporal logic specifications using model-free reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 349–10 355.

[16] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[17] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[18] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," *arXiv preprint arXiv:1807.01675*, 2018.

[19] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[20] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep rl for model-based control," *arXiv preprint arXiv:1802.09081*, 2018.

[21] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," in *International conference on machine learning*. PMLR, 2017, pp. 703–711.

[22] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *arXiv preprint arXiv:1707.01495*, 2017.

[23] J. Garcia and F. Fernández, "Safe exploration of state and action spaces in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515–564, 2012.

[24] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration in finite markov decision processes with gaussian processes," in *Advances in Neural Information Processing Systems*, 2016, pp. 4312–4320.

[25] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[26] R. Koppejan and S. Whiteson, "Neuroevolutionary reinforcement learning for generalized helicopter control," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009, pp. 145–152.

[27] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[28] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2460–2465.

[29] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, 2017, pp. 908–918.

[30] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration," in *IEEE Conference on Decision and Control (CDC)*, 2018, pp. 6059–6066.

[31] T. J. Perkins and A. G. Barto, "Lyapunov design for safe reinforcement learning," *Journal of Machine Learning Research*, vol. 3, no. Dec, pp. 803–832, 2003.

[32] Z. Li, U. Kalabić, and T. Chu, "Safe reinforcement learning: Learning with supervision using a constraint-admissible set," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 6390–6395.

[33] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3387–3395.

[34] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[35] M. Hasanbeig, A. Abate, and D. Kroening, "Cautious reinforcement learning with logical constraints," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 483–491.

[36] D. Aksaray, C.-I. Vasile, and C. Belta, "Dynamic routing of energy-aware vehicles with temporal logic constraints," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3141–3146.

[37] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[38] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[39] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular gazebo mav simulator framework," in *Robot operating system (ROS)*. Springer, 2016, pp. 595–625.

[40] F. d'Apolito and C. Sulzbachner, "System architecture of a demonstrator for indoor aerial navigation," *IFAC-PapersOnLine*, vol. 52, no. 25, pp. 316–320, 2019.

[41] Y. Yazıcıoğlu, "Absorption in time-varying Markov chains: Graph-based conditions," *IEEE Control Systems Letters*, vol. 5, no. 4, pp. 1127–1132, 2020.

[42] R. Bhat, Y. Yazıcıoğlu, and D. Aksaray, "Distributed path planning for executing cooperative tasks with time windows," *IFAC-PapersOnLine*, vol. 52, no. 20, pp. 187–192, 2019.

[43] R. Peterson, A. T. Buyukkocak, D. Aksaray, and Y. Yazıcıoğlu, "Decentralized safe reactive planning under TWTL specifications," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.