

Model-free Motion Planning of Autonomous Agents for Complex Tasks in Partially Observable Environments

Junchao Li^{1*†}, Mingyu Cai^{2†}, Zhen Kan³, Shaoping Xiao^{4†}

^{1*}Department of Mechanical Engineering, Iowa Technology of Institute, The University of Iowa, Seamans Center, 3131, Iowa City, 52242, IA, USA.

²Department of Mechanical Engineering, Lehigh University, 19 Memorial Dr W, Bethlehem, 18015, PA, USA.

³Department of Automation, University of Science and Technology of China, No. 443 Huangshan Road, Shushan District, Hefei, 230022, Anhui, China.

⁴Department of Mechanical Engineering, Iowa Technology of Institute, The University of Iowa, Seamans Center, 3131, Iowa City, 52242, IA, USA.

*Corresponding author(s). E-mail(s): junchao-li@uiowa.edu;
Contributing authors: mic221@lehigh.edu; zkan@ustc.edu.cn;
shaoping-xiao@uiowa.edu;

[†]These authors contributed equally to this work.

Abstract

Motion planning of autonomous agents in partially known environments with incomplete information is a challenging problem, particularly for complex tasks. This paper proposes a model-free reinforcement learning approach to address this problem. We formulate motion planning as a probabilistic-labeled partially observable Markov decision process (PL-POMDP) problem and use linear temporal logic (LTL) to express the complex task. The LTL formula is then converted to a limit-deterministic generalized Büchi automaton (LDGBA). The problem is redefined as finding an optimal policy on the product of PL-POMDP with LDGBA based on model-checking techniques to satisfy the complex task. We implement deep Q learning with long short-term memory (LSTM) to process the observation history and task recognition. Our contributions include the proposed

method, the utilization of LTL and LDGBA, and the LSTM-enhanced deep Q learning. We demonstrate the applicability of the proposed method by conducting simulations in various environments, including grid worlds, a virtual office, and a multi-agent warehouse. The simulation results demonstrate that our proposed method effectively addresses environment, action, and observation uncertainties. This indicates its potential for real-world applications, including the control of unmanned aerial vehicles (UAVs).

Keywords: motion planning, partially observable environments, complex tasks, linear temporal logic, reinforcement learning, recurrent neural networks

1 Introduction

The partially observable Markov decision process (POMDP) [1] provides a mathematical framework to model decision-making problems, including motion planning of autonomous agents, e.g., unmanned aerial vehicles (UAVs). It differs from Markov decision processes (MDPs) [2] that have been widely applied in robotics and autonomous systems, assuming the environment is fully observable. POMDPs are more realistic for real-world applications in which the agent (e.g., the robot) may lack enough information from perception and cannot completely identify the state of the environment. On the other hand, simple go-to-goal motion planning tasks have been extensively studied by conventional pathfinding techniques [3] and reinforcement learning (RL) methods [4]. However, complex tasks like surveillance missions are more relevant to real-world applications. Therefore, it is challenging for the agent to learn how to plan its motions to accomplish complex tasks in partially observable environments, especially considering environment and action uncertainties.

Over the past decade, researchers have attempted to solve POMDP problems (with simple go-to-goal tasks) by using various model-based reinforcement learning (RL) algorithms. Many modern solvers can handle large spatial domains with thousands of states [5]. A commonly-used approach includes point-based value iteration (PBVI) [6, 7] methods, consisting of model-based algorithms to approximately solve the POMDP problems by computing a value function over a finite subset of the belief space. It shall be noted that a belief state represents a probability distribution of the states where the agent can be. After each transition, the belief state needs to be updated by the transition and observation probabilities via the Bayesian approach [8]. Indeed, such model-based approaches transform a POMDP problem into an equivalent MDP problem with the corresponding belief state space.

Another solution to POMDP problems is the model-free RL approach, in which the agent doesn't know transition and observation probabilities. Consequently, the policy maps a sequence of observations (i.e., the observation history) to the selected action. Mnih *et al.* [9, 10] first introduced a deep Q-Learning (i.e., DQN) and tested it on several Atari 2600 games, in which the Q networks were trained to reach human-level performances. Particularly, their Q networks took the last four frames (in grayscale) as the input and utilized a convolutional neural network (CNN) [11] to extract the image

features for a fully-connected neural network to predict Q values (i.e., the state-action values). However, since the environment was modeled as MDPs in their work, this approach bypassed the issue of partial observability, and the last four frames allowed the agent to access limited past experiences.

Based on the above-mentioned works, Hausknecht and Stone [12] proposed adding a recurrent neural network (RNN) to the Q network architecture. They modeled the Atari 2600 games as POMDP problems and proposed a so-called deep recurrent Q-networks (DRQNs) by replacing the first post-convolutional fully-connected layer of Q networks with a long short-term memory (LSTM) [13], which took a single image at each time step as the input feature. Hence, this approach was able to integrate the entire observation history instead of utilizing the observation sequences with a fixed length (e.g., four frames of images as the input in [9]). In addition, Foerster *et al.* [14] extended the DRQN to handle multi-agent RL problems in partially observable environments by proposing a deep distributed recurrent Q-networks (DDRQN), where the last action was fed as the input to the Q network. Zhu *et al.* [15] developed a new network architecture called action-specific deep recurrent Q-network (ADRQN), in which an LSTM layer processed the action history and associated observations for the Q value computing. Some other similar works, including [16, 17], implemented recurrent neural networks (RNNs) for the control policy on POMDP problems with continuous state spaces.

It shall be noted that the works on POMDP problems mentioned above consider simple go-to-goal missions only. However, complex tasks have been included in MDP problems via formal languages [18], such as linear temporal logic (LTL). Generally, user-defined high-level specifications can be expressed as an LTL formula, which is then converted to an ω -automaton over infinite words with a Büchi or a Rabin acceptance condition [19]. Consequently, robotic motion planning problems can be solved via control synthesis for a product of MDP and automaton. Recently, this formal approach was employed to verify the task objectives when solving POMDP problems with certain temporal logic constraints. Chatterjee *et al.* [20] studied the undecidability of the qualitative model checking in an infinite-horizon POMDP. Their approach relied on exploring the entire belief space and was most suitable to the problems with small state spaces. They also concluded that it might be unable to acquire the optimal policy, ensuring the maximum satisfaction probability of the specific logic formula in POMDPs.

Other works like [21–23] proposed solving such problems by converting LTL specifications to a deterministic Rabin automaton (DRA), then constructing a product of POMDP and DRA. Specifically, Sharan *et al.* [21] and Ahmadi *et al.* [23] employed finite state controllers (FSCs) to limit the policy search via the value iteration method. Also, Bouton *et al.* [22] utilized the approximate POMDP solver, SARSOP [7], to search for an optimal policy on the finite belief state space of the product POMDP. However, those approaches are model-based RL methods, which require the agent to know the transition and observation probabilities. Such a requirement limits the applications in unknown environments. It shall be noted that the most related work that employs RNNs and model-checking for POMDP problems satisfying the temporal logic constraints was carried out by Carr *et al.* [24, 25]. Their proposed method

focused on constructing a policy-based procedure to iteratively improve the current policy by implementing RNN. However, their approach constructed an underlying MDP to map the original POMDP and utilized a model-based RL solver, SARSOP [7], to approximate the value function. Hence, as they stated, it was still a model-based approach.

On the other hand, It has been shown in [26] that a limit-deterministic Büchi automaton (LDBA) has more advantages than a DRA in model-free and model-based RL learning when solving MDP problems. Hasanbeig *et al.* [27] stated that converting the LTL specification into an LDBA might result in a smaller automaton state space than a DRA. Furthermore, unlike LDBAs, a limit-deterministic generalized Büchi automaton (LDGBA) has multiple accepting sets and enables the agent can visit all accepting sets infinitely often. In addition, the conversion of LDGBA avoids the sparsity of reward caused by LDBA, which may slow down the convergence in RL [28].

In this paper, we model the interactions between the agent and its partially observable surroundings as a probabilistic-labeled POMDP (PL-POMDP). Introducing probabilistic labels in a POMDP enables us to consider both static and dynamic events in the environment. One of the contributions of this work is converting the LTL formula to an LDGBA, which represents LTL specifications for a complex task in the considered POMDP problem. This has not been reported in previous works, according to the authors’ best knowledge. After generating a product of PL-POMDP and LDGBA, the original problem of finding a policy that satisfies LTL specifications in a PL-POMDP can be reformulated as finding an optimal policy to maximize the collected reward on the corresponding product POMDP. Another contribution is proposing a model-free RL approach to learn an optimal policy on the product POMDP. We implement an RNN into Q network architectures to process the information that the agent acquires: the observation history and the task recognition. The latter depends on whether the agent fully recognizes the LTL-induced automaton. Specifically, either the history of automation states or the history of the state labels is utilized, respectively,

This paper is organized as follows: Section 2 reviews the PL-POMDP definition and introduces deep Q-learning with RNN to solve a simple go-to-goal POMDP problem. Section 3 presents LTL, LDGBA, and product PL-POMDP. Then, the problem of PL-POMDP with LTL specifications is reformulated. Section 4 proposes the model-free approaches to solve product PL-POMDP problems and provides detailed algorithms. Finally, experiments and results are included in Section 5, followed by the discussions and conclusions, including future works.

2 Background

In this section, we first define the probabilistic-labeled POMDP (PL-POMDP) and then explain using DQN, a model-free RL method, to solve a POMDP problem with simple go-to-goal tasks. Finally, a simulation example is conducted for the demonstration. Our focus in this paper is extending the DQN for the POMDP problem with complex tasks, and the developed methodology will be detailed in Section 4.

2.1 PL-POMDP

The POMDP is usually adopted as a mathematical description of problems in which the agent cannot fully observe and completely identify its surroundings. We employ PL-POMDP with the consideration of static and dynamic events.

Definition 1 (PL-POMDP). *Considering the transition and observation uncertainties and the probabilistic labels of states, we can denote a PL-POMDP by a tuple $\mathcal{P} = (S, A, T, s_0, R, O, \Omega, \Pi, L, P_L)$, which consists of:*

- A finite set of states, $S = \{s_1, \dots, s_n\}$.
- A finite set of actions, $A = \{a_1, \dots, a_m\}$. Particularly, $A(s)$ is the set of actions available for the agent at the current state s .
- A transition probability function, $T : S \times A \times S \rightarrow [0, 1]$ when the agent moves from the current state $s \in S$ to the next state $s' \in S$ after executing an action $a \in A(s)$. There exists $\sum_{s' \in S} T(s, a, s') = 1$.
- An initial state, $s_0 \in S$.
- A reward function, $R : S \times A \times S \rightarrow R$. It shall be noted that reward function sometimes can be defined as $R(s')$, $R(s, a)$ or $R(a, s')$.
- A finite set of observations, $O = \{o_1, \dots, o_k\}$. At the current state s , $O(s)$ consists of possible observations the agent can perceive.
- An observation probability function, $\Omega : S \times A \times O \rightarrow [0, 1]$, represents the probability that the agent can perceive observation o at state $s' \in S$ after executing action $a \in A(s)$. This function follows $\sum_{o \in O(s')} \Omega(s', a, o) = 1$.
- A set of atomic propositions, Π .
- A labeling function, $L : S \rightarrow 2^\Pi$, outputs a set of all possible labels at state s . 2^Π is the power set of Π .
- A labeling probability function, $P_L(s, l)$, represents the probability of a label $l \in L(s)$ associated with a state $s \in S$. It satisfies $\sum_{l \in L(s)} P_L(s, l) = 1, \forall s \in S$.

When the agent interacts with its environment, it has a (transition) probability $T(s, a, s')$ to move from state $s \in S$ to state $s' \in S$ after choosing and executing an action $a \in A(s)$. At the next state s' , the agent has an (observation) probability $\Omega(s', a, o)$ to perceive an observation $o \in O(s')$. In addition, the agent receives a reward as feedback from the environment based on the reward function $R(s, a, s')$.

The states' labels are atomic propositions representing event occurrences at specific states. In addition, the labeling probability function can characterize a static event l at state s if $L(s) = \{l\}$ and $P_L(s, l) = 1$ or a dynamic event otherwise. The labels indicate goal states to handle simple go-to-goal tasks. When considering a complex task, we can use formal language to formulate the task regarding labels and then convert the formula to a finite state automaton. The labels are also input symbols for the induced automaton. Therefore, the accomplishment of the complex task can be validated via model checking during the motion planning. We will provide the details of such an approach in Section 3.

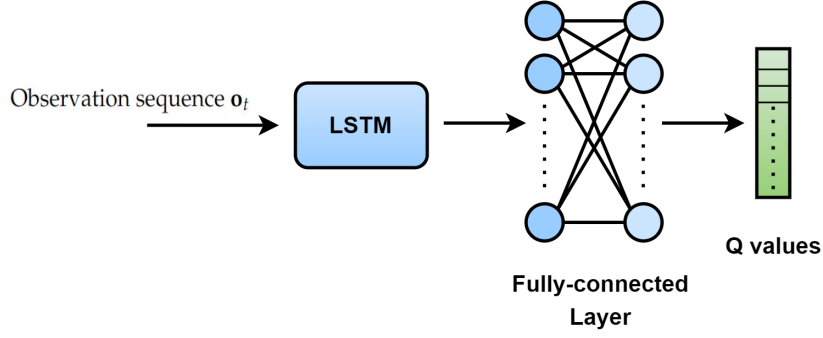


Fig. 1 Q network architecture.

2.2 DQN for POMDP problems

To solve a POMDP problem with unknown transition probability, DQN is employed to map a sequence of observations to Q values for action selection. We consider observation histories up to j previous time steps, so the sequence of observations is denoted as $\mathbf{o}_t = (o_{t-j}, o_{t-j+1}, o_{t-j+2}, \dots, o_t)$ with a length of $j + 1$. Consequently, the policy is a function of observation sequence. The agent's objective in deciding a course of action is maximizing the expected return as below, representing the total discounted rewards the agent can collect from the current time under a policy ξ .

$$U^\xi(s_t) = \mathbb{E}^\xi \left[\sum_{\tau=0}^{\infty} \gamma^\tau R(s_{t+\tau}, a, s_{t+\tau+1}) \middle| s_t \right] \quad (1)$$

where s_t is the agent's state at time t , and $\gamma \in [0, 1]$ is the discount factor to balance the importance between immediate and future rewards. Then, the optimal policy can be found as $\xi^*(\mathbf{o}_t) = \operatorname{argmax}_\xi U^\xi(s_0)$.

We implement an LSTM to process the observation sequence in the Q network architecture as shown in Figure 1. LSTM is one type of RNN that can model temporal dependencies between observations by introducing feedback loops in the network architecture. After processing the observation at each time step, the output is fed back into the network as input (together with the following observation) for the next time step. This allows the network to capture information about the order and timing of observations in the sequence. One-dimensional convolutional neural networks (CNNs) and deep neural networks (DNNs) can also be used to process time-series data. However, they don't explicitly model the temporal dependencies between observations, and DNNs cannot process input sequences of variable length. In the next subsection, we use a simple go-to-goal simulation example to demonstrate the advantages of RNN in Q networks over CNN and DNN.

DQN usually has two Q networks: an evaluation Q-network $Q_E(\mathbf{o}_t, a_t; \theta_E)$ and a target Q-network $Q_T(\mathbf{o}_t, a_t; \theta_T)$, where θ_E and θ_T are the network weights, respectively. The evaluation Q-network Q_E is usually updated at each iteration by randomly



Fig. 2 A 10 x 10 grid world

selecting a batch of data samples from a so-called replay memory [29] during the learning process. At the same time, the target Q-network Q_T keeps fixed weights until copying from the evaluation Q-network Q_E once in a while, i.e., $\theta_T = \theta_E$.

At each step (e.g., time t) during the learning process, the target Q-network (Q_T) predicts Q values based on the sequence of observations $\mathbf{o}_t$ so that the agent can choose an action a_t via the ϵ -greedy technique [4]. After executing the selected action and perceiving an observation o_{t+1} , a new sequence of observation $\mathbf{o}_{t+1} = (o_{t-j+1}, o_{t-j+2}, o_{t-j+3}, \dots, o_{t+1})$ is generated. Consequently, an experience is formed as $e_t = (\mathbf{o}_t, a_t, r_t, \mathbf{o}_{t+1})$, where $r_t = R(s_t, a_t, s_{t+1})$. The experience is recorded as one data sample in the replay memory D , where the data samples are randomly selected to update the new Q values for Q network updating. The equation used to update the Q value associated with a_t is defined below.

$$Q_{new}(\mathbf{o}_t, a_t) = Q_E(\mathbf{o}_t, a_t; \theta_E) + \alpha \left[r_t + \gamma \max_{a_{t+1}} Q_T(\mathbf{o}_{t+1}, a_{t+1}; \theta_T) - Q_E(\mathbf{o}_t, a_t; \theta_E) \right] \quad (2)$$

where α is the learning rate.

2.3 A go-to-goal example

We take a simple go-to-goal example in a grid world to demonstrate DQN, i.e., a model-free RL method, to solve a POMDP problem. Figure 2 illustrates a 10×10 grid world, in which the blue area labeled as ‘a’ and the green area labeled as ‘b’ are the initial and goal state, respectively. The block states, i.e., obstacles, are labeled with ‘B’. The agent, e.g., a mobile robot in the grid world, must move from the initial state to the goal state. It can take four actions at each state: *up*, *left*, *down*, and *right*. Due to the action uncertainty, the agent has a probability of 0.9 to take the selected action. Otherwise, it takes two side-way actions with equally weighted probabilities. In addition, the agent will remain in the current state if the next state is outside the grid world or the obstacles. After reaching the next state, the agent can observe this state with a probability of 0.9 and adjacent states with a total probability of 0.1 uniformly distributed. The discount factor γ is set as 0.98.

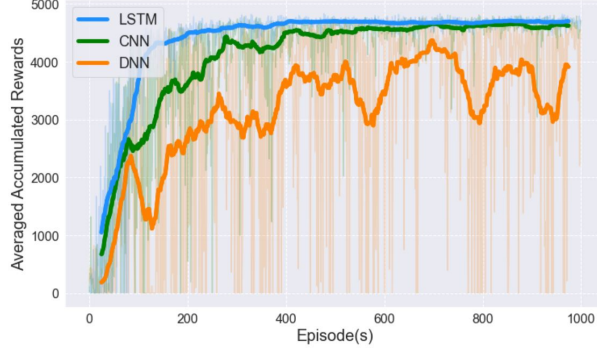


Fig. 3 The comparison of accumulated rewards by using Q networks with LSTM, CNN, and DNN.

It shall be noted that the observations are states in this grid world example. Therefore, the observation sequence is pre-processed by the technique of one-hot encoding since the row and column indices of a state are ordinal data. The generated 1-D vector as the input is fed into the LSTM layer. The extracted feature is then passed to two fully-connected layers with 16 neurons, respectively, to predict the corresponding Q values. The rectified linear unit (ReLU) is utilized as the activation function in the Q networks. The learning process for this problem includes 500 steps per episode for 1,000 episodes. Two Q networks, the evaluation network Q_E and the target network Q_T , are randomly initialized. The training batch size for the evaluation network is 32, and the target network is updated by copying the weight coefficients of Q_E every 50 steps.

We also test two other Q network architectures by implementing CNN and deep neural network (DNN). CNN-based Q networks use a 2D convolutional layer (filter=6, kernel size=(3,3), strides=1) followed by another 2D convolutional layer (filter=12, kernel size=(2,2), strides=1) before the fully connected layers. The DNN uses the fully connected layers only. Figure 3 compares the averaged accumulated rewards collected by the agent every ten episodes (the trend lines represent the simple moving average (SMA) of rewards every 50 episodes). The results illustrate that Q networks with LSTM can achieve faster convergence and a higher accumulated reward.

An optimal policy can be derived from the Q networks after convergence. Figure 4 displays a path generated from the derived policy for the agent moving on the grid world to accomplish this simple go-to-goal task. The start state is marked as the large purple solid circle, and the reached states are marked as light red dots. The brighter red dot and the bend of the black route indicate this state has been visited more than once. It shall be noted that even if the policy converges and approaches the optimal, the generated path may not. This is because of observation and action uncertainties.

3 Problem Definition

In this study, we introduce a framework for solving a PL-POMDP problem with LTL specifications by transforming the LTL formula into an LDGBA that represents the task variables and safety constraints of the POMDP and generating a product of

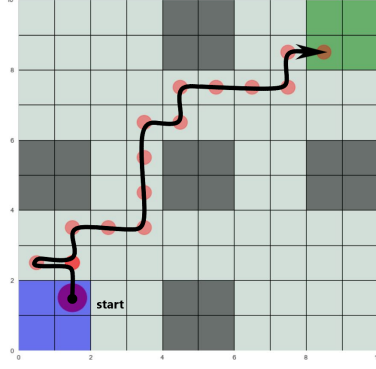


Fig. 4 A path generated from the derived policy.

POMDP and LDGBA. The problem of satisfying the given LTL constraints in a POMDP is equivalent to the problem of reaching (Büchi) accepting states in the product POMDP.

3.1 Linear temporal logic (LTL)

Linear temporal logic [18] is a logical formalism for linear-time properties, representing the relation between state labels in sequential executions. In addition to the Boolean connectors, LTL extends propositional logic by adding some temporal operators, including two basic ones $\bigcirc$ (pronounced “next”) and $\mathcal{U}$ (pronounced “until”). This study assumes that $a \in \Pi$ is an atomic proposition, and ϕ, ϕ_1 and ϕ_2 are single LTL formulas. Then, LTL formulas can be formed according to the following grammar [30]:

$$\phi := \text{True} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1\mathcal{U}\phi_2 \quad (3)$$

where negation ($\neg$) and conjunction ($\wedge$) are the Boolean operators. Formula $\bigcirc\phi$ is true at the current time if ϕ is true the next time. In addition, Formula $\phi_1\mathcal{U}\phi_2$ is true at the current time if ϕ_2 is true for some future time and ϕ_1 is true at all times until that future time.

Other commonly-used temporal operators are $\Diamond$ (pronounced “eventually”) and $\Box$ (pronounced “always”). Formula $\Diamond\phi$ ensures that ϕ will be true eventually in the future, while $\Box\phi$ is true from now on forever. They can be derived as follows:

$$\begin{aligned} \text{eventually : } \Diamond\phi &\equiv \text{True } \mathcal{U}\phi \\ \text{always : } \Box\phi &\equiv \neg(\Diamond\neg\phi) \end{aligned} \quad (4)$$

When using $\models$ to represent the satisfaction relationship, we can interpret the semantics of an LTL formula over words as below. A word is an infinite sequence $\mathbf{w} = w_0w_1 \dots$

with $w_i \in 2^\Pi$ where Π is a set of atomic propositions for all $i \geq 0$.

$$\begin{aligned}
\mathbf{w} &\models \text{True} \\
\mathbf{w} &\models \alpha && \Leftrightarrow \alpha \in L(\mathbf{w}[0]) \\
\mathbf{w} &\models \phi_1 \wedge \phi_2 && \Leftrightarrow \mathbf{w} \models \phi_1 \text{ and } \mathbf{w} \models \phi_2 \\
\mathbf{w} &\models \neg\phi && \Leftrightarrow \mathbf{w} \not\models \phi \\
\mathbf{w} &\models \bigcirc\phi && \Leftrightarrow \mathbf{w}[1:] \models \phi \\
\mathbf{w} &\models \phi_1 \mathcal{U} \phi_2 && \Leftrightarrow \exists t \text{ s.t. } \mathbf{w}[t:] \models \phi_2, \forall t' \in [0, t), \mathbf{w}[t'] \models \phi_1
\end{aligned} \tag{5}$$

3.2 Limit-deterministic generalized Büchi automaton (LDGBA)

The previous subsection describes that a user-specified complex task can be formulated via LTL. Then, we can convert the LTL formula into an automaton, including LDGBA [31], to evaluate task satisfaction via model checking [18].

Definition 2 (LDGBA). *An LDGBA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ consists of a finite set of states Q , a finite alphabet (i.e., a finite set of input symbols) $\Sigma = 2^\Pi$ where Π is a set of atomic propositions, a transition function $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$, an initial state $q_0 \in Q$, and a set of accepting sets $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_f\}$ where $\mathcal{F}_i \subseteq Q$, $\forall i \in \{1, \dots, f\}$. Furthermore, the state set Q can be decomposed into deterministic and non-deterministic sets, i.e., Q_D and Q_N , respectively. They satisfy the following requirements.*

- $Q_D \cup Q_N = Q$ and $Q_D \cap Q_N = \emptyset$.
- The state transitions in Q_D are total, i.e., $|\delta(q, \alpha)| = 1$.
- The transitions in Q_D are restricted within it, i.e., $\delta(q, \alpha) \subseteq Q_D$ for every state $q \in Q_D$ and $\alpha \in \Sigma$.
- The ϵ -transitions do not take the input symbols and are only valid from $q \in Q_N$ to $q' \in Q_D$.
- The accepting sets, consisting of accepting states, are only defined in the deterministic set. In other words, $\mathcal{F}_i \subseteq Q_D$ for every $\mathcal{F}_i \in \mathcal{F}$.

A run of an LDGBA, subject to an input word $\mathbf{w} = w_0 w_1 \dots$, can be expressed as $\mathbf{q} = q_0 q_1 \dots$, according to the transition function $\delta(q_i, w_i) = q_{i+1}$. Let $\text{inf}(\mathbf{q})$ represent the infinite portion of $\mathbf{q}$. Theoretically, $\mathbf{q}$ satisfies the LDGBA acceptance condition, i.e., the LDGBA accepts the word $\mathbf{w}$, if there exists $\text{inf}(\mathbf{q}) \cap \mathcal{F}_i \neq \emptyset, \forall i \in \{1, \dots, f\}$. We suggest readers check Owl [19] for more details about automaton generation. This study aims to solve the POMDP problems with LTL constraints as defined below.

Problem 1. *Given a PL-POMDP $\mathcal{P}$ and a complex task expressed via an LTL formula. The objective is to find a policy $\xi^*(\mathbf{o}_t)$, where $\mathbf{o}_t$ denotes a sequence of observations on $\mathcal{P}$, that can complete the task by satisfying the acceptance condition of the LTL-induced LDGBA.*

3.3 Product POMDP

Definition 3 (Product POMDP). Given a PL-POMDP $\mathcal{P} = (S, A, T, s_0, R, O, \Omega, \Pi, L)$ and an LDGBA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$, the product PL-POMDP (simply named as the product POMDP) is defined by $\mathcal{P}^\times = \mathcal{P} \times \mathcal{A} = (S^\times, A^\times, T^\times, s_0^\times, R^\times, O, \Omega^\times, \mathcal{F}^\times)$, consisting of the following components.

- A finite set of labeled states, $S^\times = S \times Q$ or $s^\times = \langle s, q \rangle \in S^\times$ where $s \in S$ and $q \in Q$.
- A finite set of actions, $A^\times = A \cup \{\epsilon\}$.
- A transition function, $T^\times = S^\times \times A^\times \times S^\times \rightarrow [0, 1]$, and

$$T^\times(s^\times, a^\times, s^{\times'}) = \begin{cases} T(s, a^\times, s') & q' = \delta(q, l), l \in L(s') \text{ and } a^\times \in A \\ 1 & a^\times \in \{\epsilon\} \text{ and } q' \in \delta(q, \epsilon) \text{ and } s' = s, \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

where $s^{\times'} = \langle s', q' \rangle$.

- An initial state $s_0^\times = \langle s_0, q_0 \rangle \in S^\times$ where $s_0 \in S$ and $q_0 \in Q$.
- A reward function $R^\times = S^\times \times A^\times \times S^\times \rightarrow \mathcal{R}$, and

$$R^\times(s^\times, a^\times, s^{\times'}) = \begin{cases} R(s, a^\times, s') & a^\times \in A, l \in L(s'), q' = \delta(q, l) \in \mathcal{F}_i, \text{ and } \mathcal{F}_i \in \mathcal{F} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

- An observation function $\Omega^\times = S^\times \times A^\times \times O \rightarrow [0, 1]$ is

$$\Omega^\times(s^{\times'}, a^\times, o) = \Omega(s', a^\times, o) \quad (8)$$

If $a^\times \in A$. Otherwise, if $a^\times \in \{\epsilon\}$, the agent stays at the same state s , i.e., $s' = s$, but $q' = \delta(q, \epsilon)$, and no observation is perceived.

- A set of accepting sets $\mathcal{F}^\times = \{\mathcal{F}_1^\times, \mathcal{F}_2^\times, \dots, \mathcal{F}_f^\times\}$ where $\mathcal{F}_i^\times = \{\langle s, q \rangle | s \in S; q \in \mathcal{F}_i\}$ and $i = 1, \dots, f$.

A random path on the product POMDP, represented by $(s_0, q_0)(s_1, q_1) \dots$, is an integration of a path $s_0 s_1 \dots$ on the PL-POMDP and a path $q_0 q_1 \dots$ on the LDGBA. Similar to (1), the expected return starting from the initial state under a policy $\xi^\times$ on the product POMDP can be written as

$$U^{\xi^\times}(s_0^\times) = \mathbb{E}^{\xi^\times} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t^\times, a_t^\times, s_{t+1}^\times) \middle| s_{t=0}^\times = s_0^\times \right] \quad (9)$$

It shall be noted that the product POMDP $\mathcal{P}^\times$ can be viewed as a PL-POMDP $\mathcal{P}$ with the augmented state space, which includes automaton state space. Therefore, the product POMDP accounts for the temporal logic specifications represented by LDGBA $\mathcal{A}$. Any feasible path on $\mathcal{P}^\times$ shares the intersections between an accessible path over the original PL-POMDP $\mathcal{P}$ and a word accepted by the LTL-induced automaton $\mathcal{A}$.

For example, a path $\sigma^{\xi^\times} = (s_0, q_0)(s_1, q_1) \dots$ can be generated by the derived policy $\xi^\times$ on the product POMDP $\mathcal{P}^\times$. If there exists $\inf(\sigma^{\xi^\times}) \cap \mathcal{F}_i^\times \neq \emptyset, \forall i = 1, \dots, f$, where $\mathcal{F}^\times$ captures the acceptance conditions of LDGBA $\mathcal{A}$, this path is accepted. In other words, the run $q_0 q_1 \dots$ satisfies the LDGBA acceptance condition, or the LDGBA accepts the corresponding word.

Therefore, according to many previous studies on solving MDP problems with LTL specifications via the product MDP [30, 32–34], an optimal policy $\xi^{\times*}(\mathbf{o}_t, \mathbf{q}_t)$ on the product POMDP $\mathcal{P}^\times$ is equivalent to the optimal policy $\xi^*(\mathbf{o}_t)$ on the PL-POMDP $\mathcal{P}$ while satisfying LTL specifications. $\mathbf{o}_t$ is the observation history while $\mathbf{q}_t$ is the corresponding history of transitioned automaton state. It shall be noted that it is assumed that the agent receives labels, i.e., input symbols in automata, as part of the feedback. Therefore, if the agent is fully aware of the task, i.e., LTL-induced automaton, the history of transitioned automaton states can be derived. Then, we can reformulate Problem 1 as follows.

Problem 2. A product POMDP $\mathcal{P}^\times = \mathcal{P} \times \mathcal{A}$, defined in Section 3.3, is constructed from a PL-POMDP $\mathcal{P}$ describing the partially observable environment and an LDGBA $\mathcal{A}$ expressing LTL specifications ϕ for a complex task. The objective is to find a policy $\xi^{\times*}(\mathbf{o}_t, \mathbf{q}_t)$, where $\mathbf{o}_t$ and $\mathbf{q}_t$ denote the sequences of observations and transitioned-automaton states, respectively, over $\mathcal{P}^\times$ so that the expected return (9) is maximized.

On the other hand, if the agent is unaware of the task, the agent cannot derive automaton state transitions based on the label feedback. Consequently, the policy on the product POMDP is a function of the observation history and the perceived label history as $\xi^{\times*}(\mathbf{o}_t, \mathbf{l}_t)$. Then, the above problem formulation can be corresponding revised.

4 Methodology

In this study, we propose model-free RL approaches (i.e., DQNs) on product POMDPs to synthesize optimal motion planning for the agent in a partially observable environment subject to LTL specifications. As discussed and demonstrated in Section 2, RNNs have the advantage of being included in Q networks for solving POMDP problems because they can capture relative temporal dependencies in the observation history. In the proposed methods, we extend the RNN-enhanced Q networks to process the perceived observations and the recognition of complex tasks.

We consider two scenarios, depending on whether the agent acknowledges the assigned task. If the agent is explicitly assigned the task, it has full knowledge of the LTL-induced automaton, including the transition function. Consequently, once it reaches a product POMDP state, i.e., an augmented state, it can derive the associated automaton state, although the POMDP state is not fully observable. It shall be noted the state labels are distinguished from the observations. We assume that the agent can acquire state labels correctly via feedback from the environment. Therefore, in this case, in addition to the observation history, our method takes the identified automaton state as another input feature to Q networks. On the other hand, if the agent is not explicitly assigned the task and has no knowledge of the automaton’s

transition function, (POMDP) state labels are taken as an additional input feature to Q networks.

4.1 Automaton state sequence as additional input

Suppose the agent is fully aware of the task. In that case, the automaton is fully knowledgeable, and the automaton states (i.e., q states) can be induced from the acquired labels according to the automaton transition function. Consequently, in addition to the observation history, automaton states can be directly used as input to the Q networks. Indeed, we utilize a sequence of q states with a length of k , $\mathbf{q}_t = \bar{q}_1 \dots \bar{q}_k$, where $\bar{q}$ denotes the induced q state, representing the automation evolution by times step t corresponding to the agent’s transitions on the PL-POMDP. It shall be noted that the sequences of observations $\mathbf{o}_t = o_{t-j} \dots o_t$ have a length of $j + 1$ and are generated based on the observation history from time step $t - j$ to the current time t . However, since the automaton space is usually smaller than the POMDP space, and the automaton state transition doesn’t occur at each time step, it is not practical to generate the history of the automaton transition by recording the automaton state at each time step. Instead, we use First in, first out (FIFO) to track the automaton transitions, and each transition is recorded only once. Consequently, $\mathbf{q}_t$ may maintain the same for several consecutive steps until the next automaton state is reached. In addition, $\mathbf{o}_t$ and $\mathbf{q}_t$ usually don’t have the same length.

After each transition $a_t^\times \in A$, the observed o_{t+1} and induced automaton state q_{t+1} are used to generate the new sequences of observations $\mathbf{o}_{t+1}$ and q states $\mathbf{q}_{t+1}$. Together with the previous sequences, a new experience can be written as $(\mathbf{o}_t, \mathbf{q}_t, a_t^\times, r_t^\times, \mathbf{o}_{t+1}, \mathbf{q}_{t+1})$, which is recorded as one data sample in the replay memory. If there is no new automation transition, $\mathbf{q}_{t+1} = \mathbf{q}_t$. It shall be noted that the ϵ -transition is one of the available actions in the product POMDP as defined in Section 3.3 if it exists in the LTL-induced LDGBA. In our approaches, no observation is perceived after an ϵ -transition because the agent remains at the same POMDP state. However, the automaton state is changed and recorded in the corresponding automaton state sequence for the next time step.

Figure 5 illustrates the architecture of the Q networks in our model-free RL approaches. The sequences of observations $\mathbf{o}_t$ and automaton states $\mathbf{q}_t$ at time step t are pre-processed by one-hot-encoding before entering LSTMs, respectively. Since these two sequences don’t have the same length, two separate LSTMs are adopted to extract the hidden states, which are then concatenated and flattened for the fully-connected layers to estimate Q values.

Similar to a general DQN, our DQN on the product POMDP also has two identical Q networks: the evaluation network $Q_E^\times(\mathbf{o}_t, \mathbf{q}_t, a_t^\times; \theta_E^\times)$ and the target network $Q_T^\times(\mathbf{o}_t, \mathbf{q}_t, a_t^\times; \theta_T^\times)$ where $a^\times \in A$. The target network is utilized for the next action selection and Q value prediction. On the other hand, the evaluation network is trained every M steps by a batch of data samples randomly selected from the replay memory. After every K steps, the target network is updated by copying the weight coefficients of the evaluation networks. Using two neural networks can prevent the bootstrapping of the DQN with a single neural network.

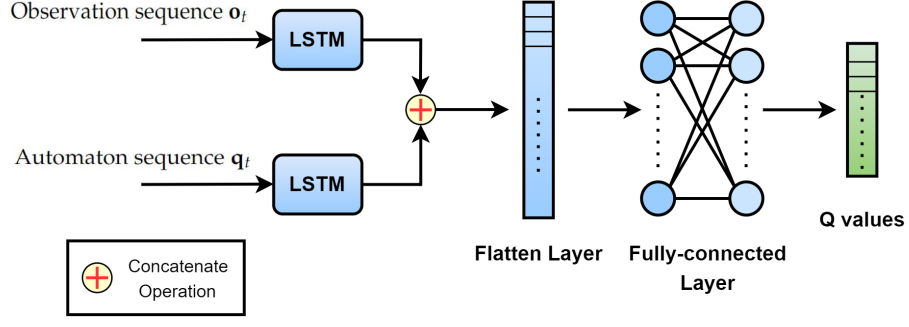


Fig. 5 The architecture of Q networks taking $\mathbf{o}_t$ and $\mathbf{q}_t$ as input.

The output of each data sample in the selected batch is a Q value, which can be updated as the equation below.

$$Q_{\text{new}}^{\times}(\mathbf{o}_t, \mathbf{q}_t, a_t^{\times}) = Q_E^{\times}(\mathbf{o}_t, \mathbf{q}_t, a_t^{\times}; \theta_E^{\times}) + \alpha \left[r_t^{\times} + \gamma \max_{a_{t+1}^{\times}} Q_T^{\times}(\mathbf{o}_{t+1}, \mathbf{q}_{t+1}, a_{t+1}^{\times}; \theta_T^{\times}) - Q_E^{\times}(\mathbf{o}_t, \mathbf{q}_t, a_t^{\times}; \theta_E^{\times}) \right] \quad (10)$$

where $r_t^{\times} = R^{\times}(s_t^{\times}, a_t^{\times}, s_t^{\times'})$, α is the learning rate, and γ is the discount factor. Therefore, each data sample has the input features $\mathbf{o}_t$ and $\mathbf{q}_t$ and the output target $Q_{\text{new}}^{\times}$ to train and update the evaluation Q network, Q_E . In addition, the following loss function is used to update the weight coefficients of the evaluation Q-network.

$$\mathcal{L}(\theta_E^{\times}) = \mathbb{E}_{(\mathbf{o}_t, \mathbf{q}_t, a_t^{\times}, r_t^{\times}, \mathbf{o}_{t+1}, \mathbf{q}_{t+1}) \sim U(D)} \left[\left(r_t^{\times} + \gamma \max_{a_{t+1}^{\times}} Q_T^{\times}(\mathbf{o}_{t+1}, \mathbf{q}_{t+1}, a_{t+1}^{\times}; \theta_T^{\times}) - Q_E^{\times}(\mathbf{o}_t, \mathbf{q}_t, a_t^{\times}; \theta_E^{\times}) \right)^2 \right] \quad (11)$$

Algorithm 1 demonstrates training the evaluation Q network during the agent interactions with the environment. Once converged, the Q network can predict the state-action value (i.e., Q value) function to derive the optimal policy for the studied POMDP problem with temporal logic specifications.

4.2 Label sequence as the additional input

If the agent is unaware of the task, i.e., the LTL-induced automaton is not knowledgeable, the state labels need to be the additional input for Q networks. We modified the Q network architecture in Figure 5 by replacing the automaton state sequence with a label sequence (i.e., a sequence of input symbols) in addition to the observation sequence. As mentioned above, state labels differ from observations and can be precisely received by the agent as feedback. Consequently, the sequence of the collected experience becomes $(\mathbf{o}_t, \mathbf{l}_t, a_t^{\times}, r_t^{\times}, \mathbf{o}_{t+1}, \mathbf{l}_{t+1})$, where $\mathbf{l}_t$ is the sequence of labels with a length of k received by the time step t . Similar to the scenario of utilizing q states, the labels corresponding to the POMDP states can be sparse. Hence, we utilize the same FIFO method (as described in Section 4.1) to generate the label sequence that only

Algorithm 1 Deep Recurrent Q-Network for Product POMDP Problems.

```
1: Initialize LTL formula  $\phi$ , POMDP  $\mathcal{P}$ .
2: Convert  $\phi$  to an LDGBA  $\mathcal{A}$ .
3: Construct the product POMDP  $\mathcal{P}^\times = \mathcal{P} \times \mathcal{A}$ .
4: Initialize the evaluation network  $Q_E^\times$ , the target network  $Q_T^\times$ , the replay memory
    $D$ , the length of observation sequence  $j + 1$ , the empty  $q$  state sequence  $\mathbf{q}_0$  with
   length of  $k$ , the learning rate  $\alpha$ , the discount factor  $\gamma$ , the total number of episodes
    $E$ , the total number of steps  $N$ , the batch size  $M$ , and the  $Q_T^\times$  update steps  $K$ .
5: while The current episode  $e$  in  $E$  do
6:   Randomly select a start state  $s_0^\times$ .
7:   while The current step  $i$  in  $N$  do
8:     Select a random action  $a_i^\times$  if  $i < j + 1$ ; otherwise, select an action via the
      $\epsilon$ -greedy technique.
9:     Obtain the observation  $o_{t+1}$  and induce the automaton state  $q_{t+1}$ .
10:    Generate  $\mathbf{o}_{t+1}$  and  $\mathbf{q}_{t+1}$ .
11:    Collect the rewards  $r_i^\times$ .
12:    Store the experience  $\langle \mathbf{o}_i, \mathbf{q}_i, a_i^\times, r_i^\times, \mathbf{o}_{i+1}, \mathbf{q}_{i+1} \rangle$  in  $D$ .
13:    if  $i > 0$  and  $i \% M = 0$  then
14:      Randomly select  $M$  data samples as  $U(D)$  from the replay memory.
15:      Compute  $Q_{new}^\times$  for each data sample.
16:      Train  $Q_E^\times$  by the batch of samples.
17:    end if
18:    if  $i > 0$  and  $i \% K = 0$  then
19:      Pass the weights of  $Q_E^\times$  to  $Q_T^\times$ .
20:    end if
21:  end while
22: end while
23: Training end and save the evaluation network  $Q_E^\times$ 
```

stores a label once it is received. On the other hand, Algorithm 1 can be corresponding revised.

5 Simulations and Results

We evaluate our approaches on three simulations with discrete POMDP domains. We first perform simulations over a partially observable grid world with two different tasks, considering ϵ -transition in the automaton and static/dynamic events in the PL-POMDP. Then, we test the approaches in an office scenario where we utilize two different observation settings. Finally, we also conduct a preliminary application of the proposed approach to a multiagent RL case. The simulations are programmed via Python 3.9 and Rabinizer 4. They are completed on a desktop with a 3.20 GHz eight-core CPU and 32 GB RAM. Part of the source codes and supplementary materials are provided ¹.

¹https://github.com/JunchaoLi001/Model-free-DRL-LSTM-on-POMDP_with-LDGBA

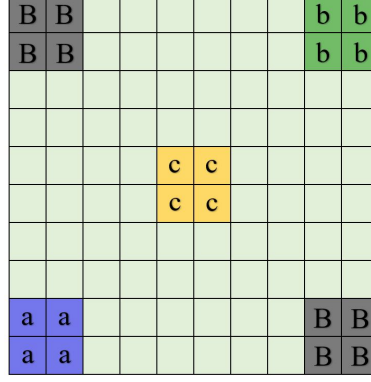


Fig. 6 A 10 x 10 grid world with trapping states

5.1 Grid world Simulations

We use a 10×10 grid world workspace, shown in Figure.6. Several states are labeled with ‘a’ in blue and ‘b’ in green, indicating two different events. The trapping states, labeled with ‘c’, indicate that the agent can never leave once entering them. Other parameters, like the transition probability and observation probability, are the same as defined in the simple go-to-goal case in Section 2.3. Two tasks are studied. Task 1 demonstrates that our approaches can handle LDGBA with ϵ -transitions. Task 2 considers two scenarios: static events and dynamic events, respectively. When assuming dynamic events, each event has a 90% probability of occurring at its labeled states and a 10% probability at the other labeled states. Both tasks are simulated for 15,000 episodes with 600 steps per episode, using the observation sequence with a length of $j + 1 = 5$, the automaton state or label sequence with a length of $k = 3$, batch size $M = 32$, and the number of steps to copy the evaluation Q network $K = 50$.

5.1.1 Task 1

The first task tested in the grid world requires the agent to visit states labeled ‘a’ or ‘b’ infinitely many times. The LTL formula is expressed below, and the induced LDGBA is shown in Figure 7.

$$\phi_1 = (\Box \Diamond a \mid \Box \Diamond b) \wedge \Box \neg c \quad (12)$$

It can be seen that the LDGBA contains ϵ -transitions that are included in the set of actions, $a^\times$, in the generated product POMDP. Figure 7 also shows that the ϵ -transitions can transition the automaton states from the initial state q_0 to either q_1 or q_2 , resulting in the agent to keep visiting only ‘a’ or ‘b’, respectively. Visiting ‘c’ will lead to a state (q_3) without an outlet as a ”trapping” state.

According to the definition of LDGBA in Section 3.2, ϵ -transitions don’t take any input symbols. They are only valid to enter the deterministic set of automaton states where the transitions are restricted. Therefore, after an ϵ -transition, the agent will be at the augmented states associated with either q_1 or q_2 to complete the task. Without

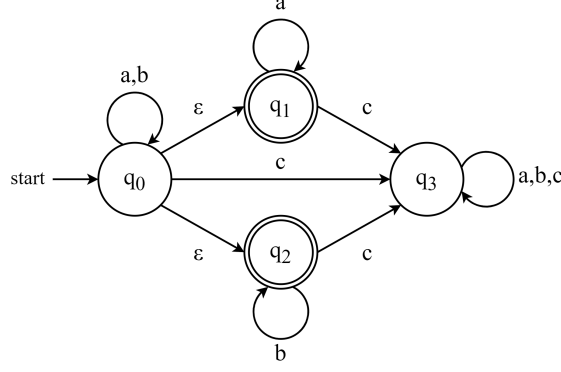


Fig. 7 The LDGBA of ϕ_1 .

the loss of generality, we give each episode a random probability for selecting the ϵ -transition, which only occurs once. Otherwise, the other actions, i.e., $a^\times \in A$, will be chosen based on the ϵ -greedy method. As mentioned in Section 3.3, the agent doesn't perceive observations right after taking ϵ -transitions, and Q networks only predict Q values of the actions other than ϵ -transitions.

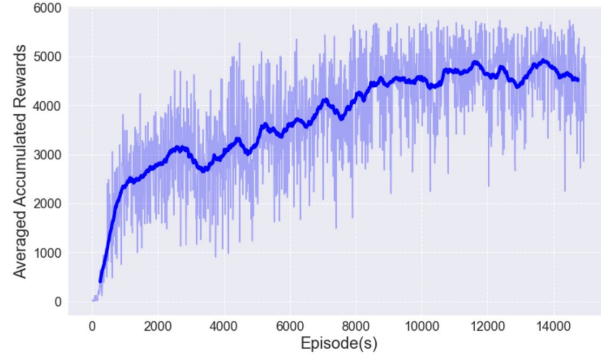


Fig. 8 The averaged accumulated rewards of task ϕ_1 .

In Task 1, we only consider static events, i.e., $P_L(s_a, 'a') = P_L(s_b, 'b') = 1$. Also, it is assumed that the agent is aware of the task. Therefore, the observation and q state histories are input to predict Q values via Q networks for action selection. Figure 8 shows the evolution of accumulated reward, averaged per 10 episodes with SMA 50 episodes. Since the agent may accidentally move into the 'trapping' state, averaged rewards better presents the trend of the rewards' convergence. The reward is set as 10 whenever the agent visits accepting states. After obtaining the optimal policy, we generate one path for the agent to accomplish the task as shown in Figure 9.

At first, the agent randomly selects actions, as shown in Figure 9(a), before generating the first observation and q state sequences. Then, Q values can be predicted,

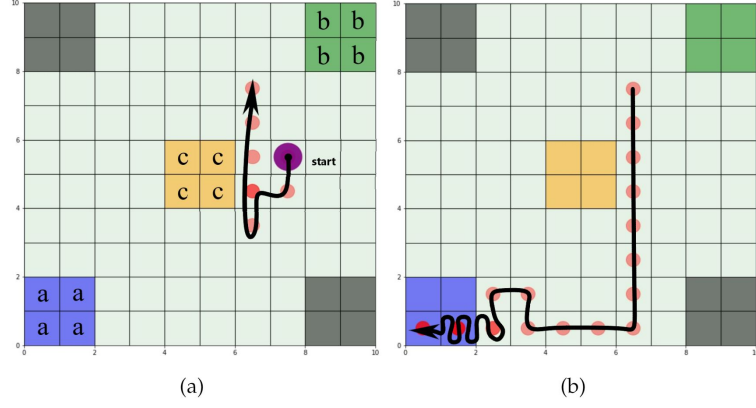


Fig. 9 A single round path of task ϕ_1 : (a) The path on q_0 . (b) The path on q_1 .

and the greedy action selection is applied unless an ϵ -transition is taken. After the ϵ -transition, the automaton state is transitioned from q_0 to q_1 , and the agent desires to visit the state labeled ‘a’ infinitely many times. This can be seen in Figure 9(b), where the agent moves down and then left to keep visiting state ‘a’. Due to the action uncertainty, the agent occasionally visits some states multiple times, indicated as bright red dots in Figure 9.

5.1.2 Task 2

The second task requires the agent to visit states labeled ‘a’ then ‘b’ in order infinitely many times, subject to dynamic events due to labeling uncertainty. The LTL formula is expressed as

$$\phi_2 = \Box \Diamond (a \wedge \Diamond b) \wedge \Box \neg c \quad (13)$$

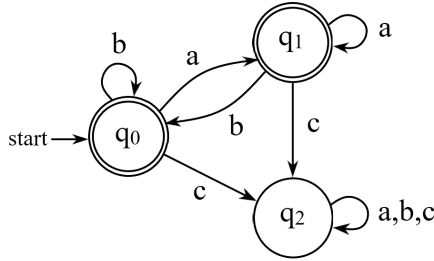


Fig. 10 The LDGBA of ϕ_2 .

In the LTL-induced LDGBA, directly utilizing the accepting sets may fail to find the deterministic policy as discussed in [27, 28]. Inspired by their works, we modify the automaton structure and reward function for easing the training process. Specifically, we augment the accepting states to separate transitions with the input symbols ‘a’ or

‘b’, respectively. Only single-labeled transitions are kept for demonstration, as shown in Figure 10 for Task 2. We then redesign the reward function (7), shown below, by adding a constraint to the reward function so that the agent can visit the accepting sets repeatedly.

$$R^\times(s^\times, a^\times, s^{\times'}) = \begin{cases} R(s, a^\times, s') & a^\times \in A, l \in L(s), q' = \delta(q, l) \in \mathcal{F}_i, \mathcal{F}_i \in \mathcal{F}, \text{ and } q' \neq q \\ 0 & \text{otherwise.} \end{cases} \quad (14)$$

where $q \neq q'$ prevents the repeated transitions at the same automaton accepting state by removing the rewards on the associated labeled POMDP states. After applying this constraint to the reward function, the derived optimal policy satisfies the desired surveillance task specification. An alternative approach can be implementing the frontier-tracking function introduced by M. Cai *et al.* [33].

For Task 2, we investigate the agent’s learning in the environment with dynamic or static events combined with two scenarios that depend on the agent’s awareness of the task. At first, to provide a detailed demonstration, we consider dynamic events and assume that the agent fully understands the task. The labeling uncertainty is introduced so that an event has a 90% probability of occurrence at its labeled states and the other labeled states otherwise. For example, at the states labeled ‘a’, event ‘a’ has a 90% probability of occurrence while event ‘b’ has a 10% probability. In addition, a q state sequence is utilized as the input of Q networks in addition to the observation sequence.

Figure 11(1) demonstrates a single round path generated from the learned policy for the agent to accomplish Task 2 when the agent is fully aware of the task. In Figure 11(a), the agent starts from the initial state (purple dot) and performs five random movements on q_0 until it generates the first observation sequence (along with the q state sequence) for decision-making. It then navigates around the trapping states and heads to the bottom left for the states labeled ‘a’ shown as the blue square. However, event ‘a’ doesn’t occur in the state when the agent first visits the blue area due to the labeling uncertainty. Therefore, the agent moves downward to the next state, where event ‘a’ occurs at the next time step. The bend of the black route at the top of the path is caused by motion uncertainty.

After visiting the state labeled ‘a’, the automaton transition happens from q_0 to q_1 . In Figure 11(b), the agent moves around the area of ‘a’ states then bypasses the trapping states in yellow to reach the states labeled ‘b’ in the top right corner. Again, event ‘b’ doesn’t occur on the agent’s first visit. Then, the agent moves one more step to the right, and event ‘b’ occurs. Consequently, the agent completes a single round to visit ‘a’ and then ‘b’. At the same time, the agent is back to the automaton state q_0 . Figure 11(c) shows the agent tries to move back to visit states labeled ‘a’ for the second round, but it keeps visiting ‘b’ states a few times before heading to ‘a’ states.

We also conduct the simulation in an environment with dynamic events when the agent is unaware of the task. The observation and label sequences are input to the Q networks in this case. Figure 11 (2) illustrates the path generated from the learned policy. We observe a similar phenomenon in which the agent makes a few attempts to visit ‘a’ states before leaving for ‘b’ states. However, after we generate more paths, we find that the agent occasionally visits the states labeled ‘b’ first because event ‘a’ has

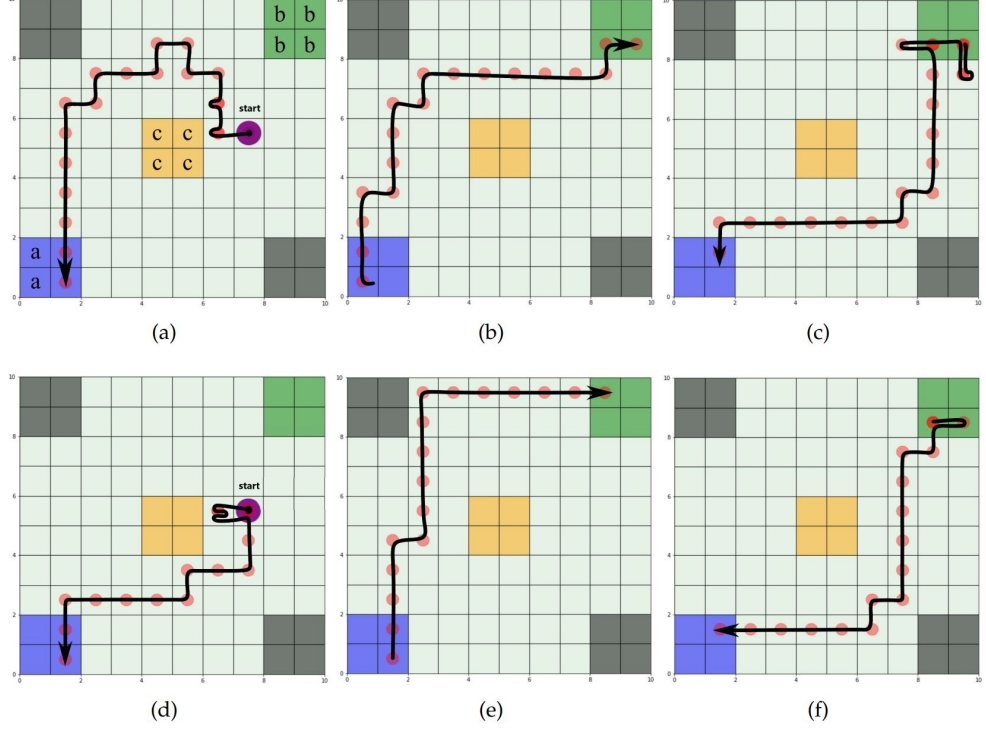


Fig. 11 A generated path for Task in ϕ_2 of dynamic event with labeling uncertainty:
(1) If it is fully aware of the task (q state sequence): **(a)** The path on q_0 . **(b)** The path on q_1 . **(c)** The path back on q_0 .
(2) If it is not aware of the task (label sequence): **(d)** The path on q_0 . **(e)** The path on q_1 . **(f)** The path back on q_0 .

10% probability of occurring on those states. This phenomenon is uniquely observed when the agent is unaware of the task.

Next, the events are assumed to be static, i.e., without labeling uncertainty. We consider both scenarios depending on whether the agent is fully aware of the task. After obtaining optimal policies for each scenario, we generate paths to demonstrate the agent accomplishing the task. Figure 12 **(1)** shows the agent planning the motion based on the observation history and the q state history. On the other hand, Figure 12 **(2)** demonstrates that the agent can accomplish the task following the policy regarding the observation history and the label-receiving history. By following both paths, the agent can complete the task, and the performances between the two approaches are similar. Comparing the dynamic event cases, we don't observe that the agent tries to visit 'a' or 'b' states multiple times before leaving for the other.

Figure 13 compares the evolution of the accumulated rewards of Task 2 for four cases discussed above. It can be observed that static event cases reach higher accumulated rewards than dynamic event cases. It may be due to the labeling uncertainty in

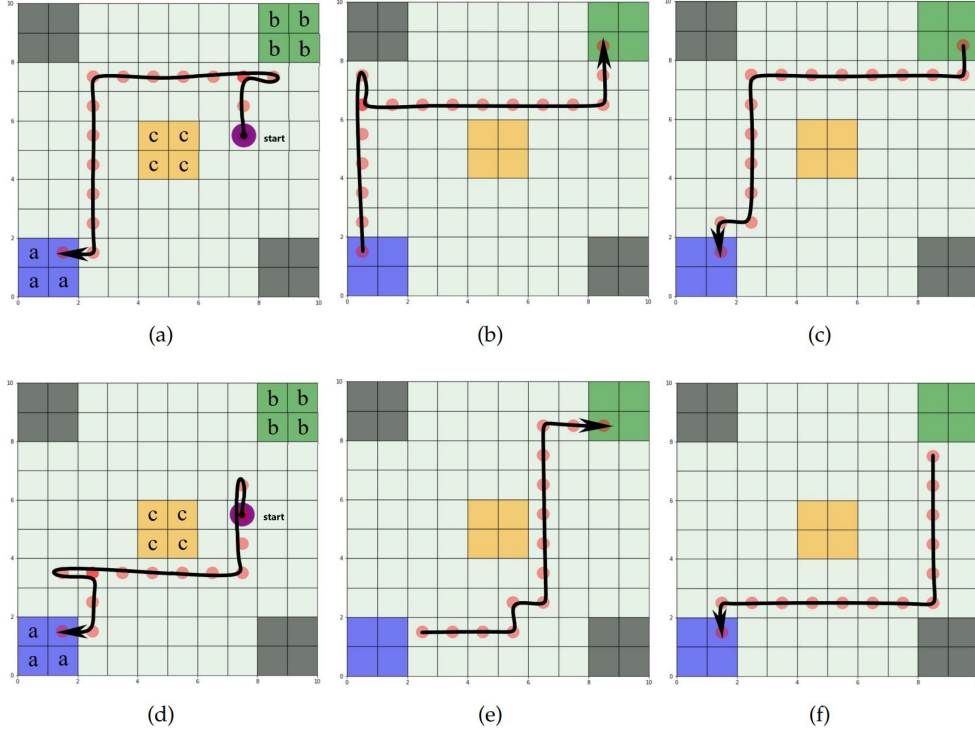


Fig. 12 A generated path for Task in ϕ_2 of static event:

- (1) If it is fully aware of the task (q state sequence): (a) The path on q_0 . (b) The path on q_1 . (c) The path back on q_0 .
(2) If it is not aware of the task (label sequence): (d) The path on q_0 . (e) The path on q_1 . (f) The path back on q_0 .

dynamic events. Also, regardless of the agent’s awareness of the task, the converged accumulated rewards are similar.

5.2 Pybullet TurtleBot Simulations

Figure. 14 shows a virtual office environment generated by PyBullet 3.0 [35]. In the office space, there are four office rooms ‘a’, ‘b’, ‘c’, and ‘d’, a storage room ‘S’, a printer room ‘Print’, and a supply station ‘Sply’ to recharge the TurtleBot, i.e., the agent. In addition, there are two big windows in Offices ‘a’ and ‘d’ and multiple doors in the office space. We discretize this office space into a four-by-four grid world to generate the POMDP model. Considering the motion uncertainties, we assume that the TurtleBot has a probability of 0.9 to successfully execute its navigation controller by moving along the desired direction. However, it can move to other possible directions, uniformly sharing a probability of 0.1. Moving toward the wall will keep the TurtleBot stay at the same location. Assuming the agent is fully aware of the assigned tasks, We test the proposed model-free RL approach with two different observation settings in this office

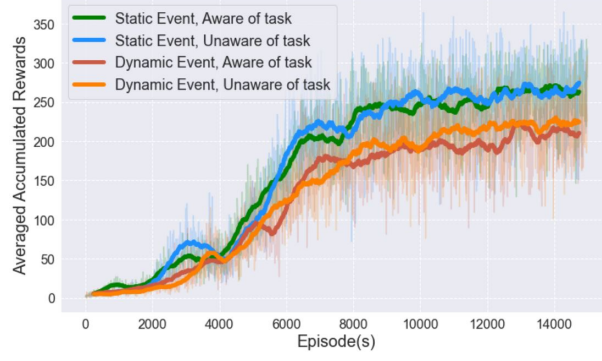


Fig. 13 Comparison of the reward evolution for task ϕ_2 .

scenario. All simulations are conducted via 10,000 episodes with 300 steps per episode. The other computation settings are the same as in Section 5.1. We map the office scenario for each simulation to a grid world in which the optimal policy is learned. Then, we apply the derived optimal policy to the virtual TurtleBot in the PyBullet platform to validate the task accomplishment.

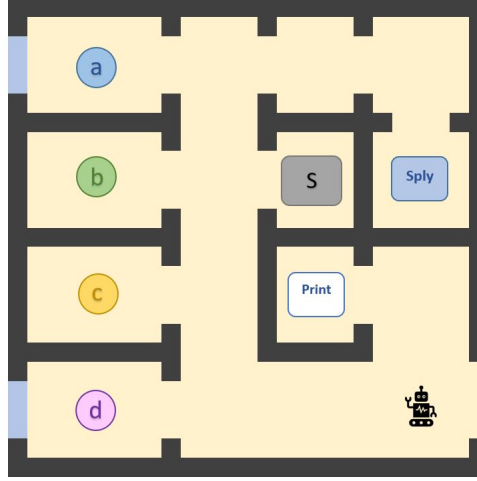


Fig. 14 The office environment.

5.2.1 Observation of the surroundings

This setting assumes that at the current state, the TurtleBot can collect the surrounding observations in all four directions, following a specific order from ‘North’, ‘West’, ‘South’ to ‘East’. The observation elements are ‘wall’, ‘hallway’, ‘door’, and ‘window’. The agent can only observe one element in each direction. It shall be noted that there

is only one observation at each state, i.e., $O(s', a, o) = 1$. However, the agent may perceive the same observation in two or more states. For example, offices ‘b’ and ‘c’ have the same observation: $o(\text{‘b’}) = o(\text{‘c’}) = \{\text{‘wall’ ‘wall’ ‘wall’ ‘door’}\}$. Consequently, the set of observation O in this POMDP problem consists of 13 distinct observations.

Task 1:

Task 1 requires that the TurtleBot visits the printer room to collect the documents and then carries the documents to Office ‘a’ or ‘c’, repeatedly. At the same time, the TurtleBot shall always avoid entering the storage room ‘S’. Similar to equation (13), this task can be expressed as an LTL formula in equation (15). Figure 15(a) shows the induced LDGBA, and we also employ the redesigned reward in equation (14).

$$\varphi_{task1} = \Box \Diamond (Print \wedge \Diamond (a \mid c)) \wedge \Box \neg S \quad (15)$$

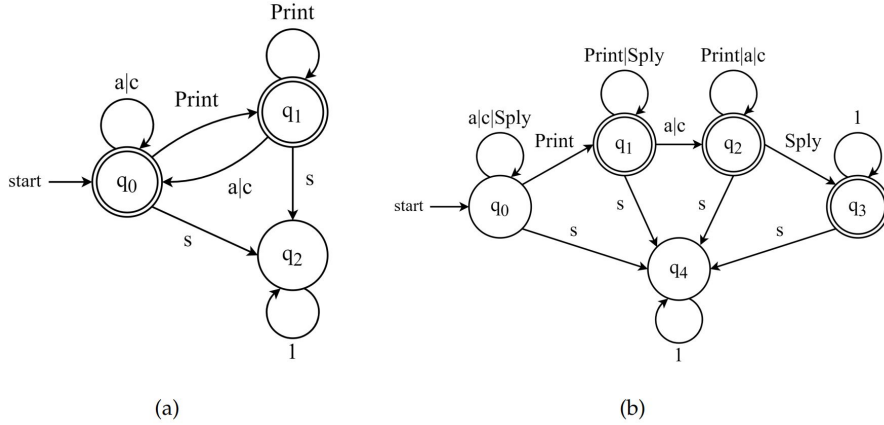


Fig. 15 The induced LDGBAs: (a) φ_{task1} . (b) φ_{task2} .

After the training process is converged, the optimal policy can be derived from Q networks. Figure 16(a) illustrates a generated path with which the TurtleBot can complete Task 1. It can be seen that after leaving the initial state, office ‘b’, the TurtleBot moves towards the printer room, indicated via a yellow path. It visits offices ‘b’ and ‘c’ more than once on the way to the printer room due to motion uncertainty. After the TurtleBot arrives at the printer room and collects the documents, a blue path demonstrates that it leaves the printer room and moves to office ‘c’ for the delivery.

Task 2:

Here, we extend Task 1 to a more complex task. Task 2 indicates that the TurtleBot must go to the supply station for recharge after delivering the documents and before repeating Task 1. The LTL formula of Task 2 can be expressed as equation (16),

and Figure 15(b) depicts the corresponding LDGBA, keeping the transitions with the single label only to simplify the illustration.

$$\varphi_{task2} = (\neg(a \mid c)\mathcal{U}Print) \wedge (\neg Sply\mathcal{U}(a \mid c)) \wedge (\Diamond Sply) \wedge (\Box \neg S) \quad (16)$$

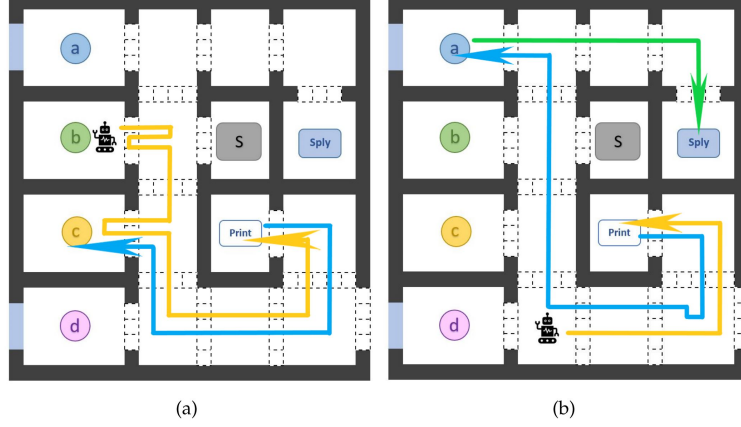


Fig. 16 Generated paths for the TurtleBot to accomplish tasks if it can observe surroundings in all four directions: (a) Task 1. (b) Task 2.

Figure 16(b) shows a path generated from the learned policy for the agent to accomplish Task 2. The TurtleBot starts outside office ‘d’. It then moves to the printer room, collects the documents, and leaves for office ‘a’, indicated as the yellow and blue routes, respectively. Finally, the green route shows that the agent arrives at the ‘Sply’ station for recharge after delivering the documents.

5.2.2 Observation of a single direction

We also consider another observation setting, in which the TurtleBot is supposed to observe only one direction randomly at each state. Consequently, the observation uncertainty increases significantly. We add a few items in the PyBullet office environment to enhance policy convergence, as shown in Figure 17. Therefore, the observable space includes ‘hallway’, ‘wall’, ‘door’, ‘window’, ‘table’, ‘paint on the wall’, and ‘flower by the wall.’ For example, the agent can observe ‘wall’, ‘table’, and ‘door’ with the probabilities of 50%, 25%, and 25%, respectively, in offices ‘b’ and ‘c’.

Figure 18 demonstrates the generated paths for the agent to accomplish the same tasks as in Section 5.2.1, respectively. Compared with the paths in Figure 16 with four directional observations, the single observation element provides an agent with less sense of the current state. Figure 18 also indicates that the agent encounters difficulty deciding the right moves. For example, the yellow path in Figure 18 (a) illustrates that the agent moves back and forth a few times in the hallway with multiple paints on the wall before finally heading to the printer room. It is mainly because of observation uncertainty in addition to motion uncertainty.

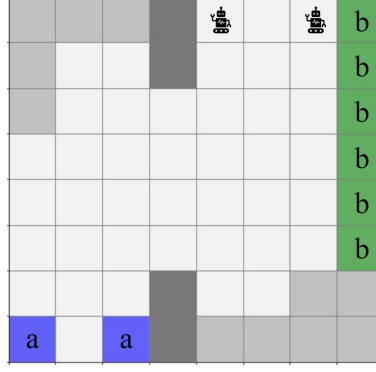


Fig. 19 The warehouse environment.

initial locations are shown in Figure 19. The observation and transition probabilities are set as the same as in Section 2.3.

We can formulate the team task via LTL as $\phi = \Box\Diamond(a \wedge \Diamond b)$, similar to the formula in equation (13). Also, we implement the Reward Redesign mentioned in Section 5.1.2 for task feasibility. Multi-agent reinforcement learning (MARL) studies how multiple agents interact in a common environment. According to agents' actual task requirements, MARL can be categorized as the following three broad classes [36]:

- Cooperative: Agents cooperate to achieve the team goal, in which no agent can perform the whole task alone.
- Competitive: Agents compete against each other.
- Mixed: Agents maximize the utility that may require cooperating and/or competing.

In addition, the multi-agent system can also be modeled in a centralized or decentralized framework, where a central policy or multiple independent policies can be learned by the agents [36], respectively.

In this case, the agents work in the same environment towards a common goal. This MARL problem can be categorized as a cooperative case. We model this problem as a simplified decentralized POMDP (Dec-POMDP) framework. Each agent has a set of actions A^i and a set of observations O^i , where i denotes the index of agents: $i \in \{1, 2\}$. However, since they are identical agents in the same state space, we define $A^1 = A^2$ and $O^1 = O^2$. Only static events are considered in this case.

There have been numerous previous works to solve MARL problems from various perspectives. Inspired by the work of Zhou *et al.* [37], we set up cooperative communication between the agents and implement it into our model-free RL algorithms. We assume the agents have full knowledge of the task. Two independent Q networks are initialized for each agent. For each Q network, the input consists of the observation sequence, the q state sequence, and the q state sequence from the other agent. The agents share the task recognition information through the messages between the agents in the same communication network. The corresponding Q networks are shown as Figure 20. Since two Q networks are trained independently to estimate Q values for each agent, we consider this approach a decentralized training and execution. The

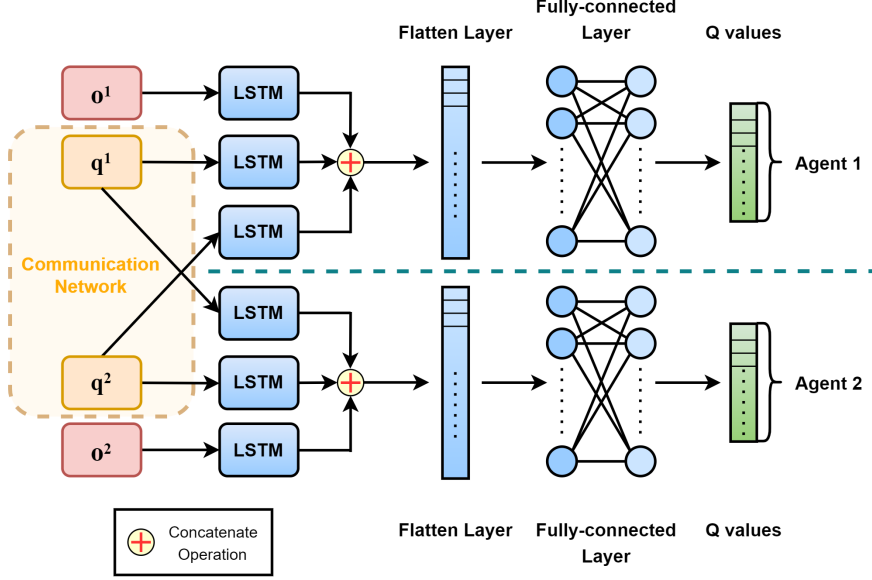


Fig. 20 Q network architectures for a MARL problem.

simulation took 30,000 episodes for 300 steps each. Figure 21 shows the trend of the accumulated reward *vs.* episode.

Figure 22 shows the derived paths for both agents, represented by red and yellow routes, respectively. At the beginning of the task in Figure 22 (a), both agents stay at their original positions to gather sufficient observations of the surrounding environment for decision-making. Shortly after, they leave for the loading zone in blue, and the yellow agent waits at a state on its way to prevent a collision with the other agent (red). Once the packages are loaded, the agents head towards the green zone to drop them off on the convey belts, as shown in Figure 22 (b). The second run starts right after the completion, shown in Figure 22 (c). It shall be noted that the yellow agent selects the action ‘*stay*’ a few times to wait for the other agent to pass because we prioritize the red agent. The generated paths show that the desired cooperative task is achieved even though separate policies are trained for the agents with limited communication.

6 Discussion

This study formulates the motion planning of autonomous agents in partially observable environments as a PL-POMDP problem. We also address high-level complex tasks by expressing them through LTLs and converting them to LDGBAs for model checking. Consequently, such a motion planning problem became equivalent to finding an optimal policy on the product of PL-POMDP and the induced LDGBA, and a model-free RL approach is proposed. We employ LDGBAs for model checking because they result in a smaller automaton state space than the corresponding DRAs. In addition, they have multiple accepting sets, enabling the agent to visit all accepting

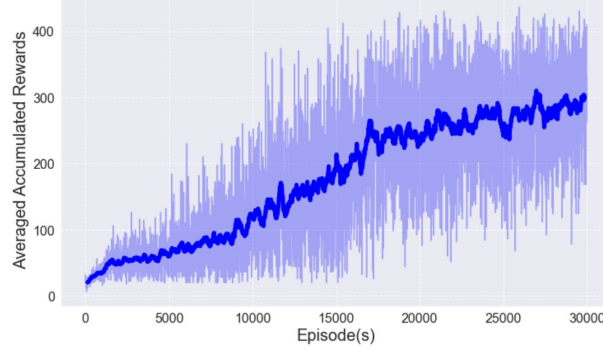


Fig. 21 The accumulated rewards of the MARL case.

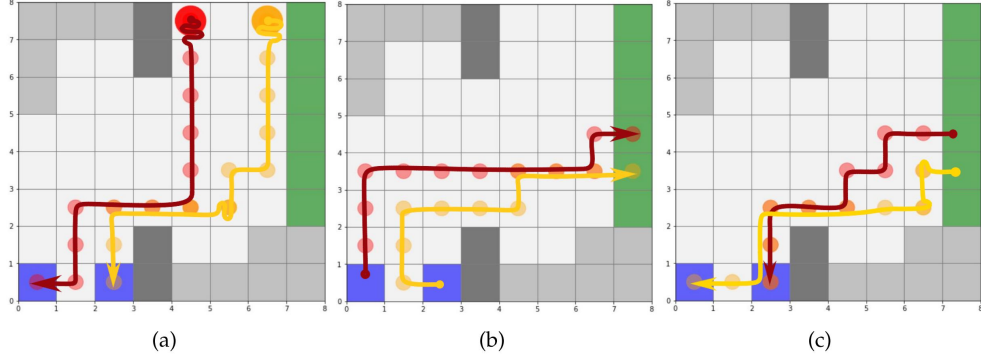


Fig. 22 The paths generated for the red and yellow agents to accomplish the assigned task.

sets infinitely often. With the implementation of the reward redesign, the modified LDGBA can address the sparsity of rewards caused by LDBA in RL for motion planning [28]. In the future, we plan to leverage the tracking-frontier function proposed in [33] to keep track of non-visited accepting sets in LDGBA. This could be particularly useful when more complicated surveillance tasks for UAVs are required in partially observable environments. By further developing this function, we aim to improve the performance and applicability of our proposed method.

Deep Q learning is employed in the proposed model-free RL approach to learning optimal policies on the product PL-POMDP. Specifically, LSTM is implemented into Q network architectures to process the agent’s observation history and task recognition. Using the induced automation state or perceived label history to represent task recognition depends on whether the agent is fully aware of the task. As the simulation examples demonstrate, either representation can enhance the agent’s learning in partially observable environments subject to complex tasks. We also investigate the performances of Q networks by using different lengths of the input sequences. Choosing too long or too short sequences can lead to an unstable training process. The sequence lengths used in this study provide the agent with sufficient information for

decision-making and avoid lengthy delays before initiating action selection and following the trained policy. It shall be noted that the length selection may depend on the solved problem. The proposed approach can be easily updated with other value-based or policy-based deep reinforcement learning (DRL) methods in future work.

We apply the proposed model-free RL approach to the motion planning of autonomous agents in discrete environments only in this study. Future works will extend the approach to POMDP problems with continuous state and action spaces. In addition, a general LDGBA can consist of non-deterministic and deterministic sets. Although most LDGBA corresponding to the tasks considered in our simulation examples have deterministic sets only, one example demonstrates that the proposed approach can handle ϵ -transitions, which are non-deterministic. Even if the initial state is in the non-deterministic set, the automaton state transitions are restricted in the deterministic set after an ϵ -transition. Because the accepting states are in the deterministic set only, the proposed approaches can still achieve optimal policies.

We also provide a preliminary simulation of a cooperative multi-agent system to demonstrate the proposed approach’s versatility and adaptability. Furthermore, by enabling the communication of task recognition between the agents, the acquired policies show a high level of collaboration that results in successful task completion. Although the current approach in the multi-agent simulation focuses on a simplified scenario by using a decentralized training and execution framework, which can often be non-stationary [36], it has the potential to address a system of multiple UAVs for cooperative missions. In the future, it may be more appropriate to use a centralized training and decentralized execution (CTDE) framework capable of handling a continuous state space. Moreover, the agents’ task recognition communication relies on their full knowledge of the tasks. An alternative approach can use local observations from other adjacent agents, so an additional neural network is needed to approximate the communication mechanism.

Acknowledgments. Li and Xiao would like to thank US Department of Education (ED#P116S210005) and NSF (#2226936) for supporting this research.

Declarations

- Funding: This research was funded by US Department of Education (ED#P116S210005) and NSF (#2226936).
- Competing interests: The authors declare no competing interests.
- Code availability: Code files are available at https://github.com/JunchaoLi001/Model-free_DRL_LSTM_on_POMDP_with_LDGBA

References

- [1] Kurniawati, H.: Partially observable markov decision processes and robotics. Annual Review of Control, Robotics, and Autonomous Systems **5** (2022) <https://doi.org/10.1146/annurev-control-042920-092451>

- [2] Kaufman, H., Howard, R.A.: Dynamic programming and markov processes. The American Mathematical Monthly **68** (1961) <https://doi.org/10.2307/2312519>
- [3] Perez, A., Platt, R., Konidaris, G., Kaelbling, L., Lozano-Perez, T.: Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In: 2012 IEEE International Conference on Robotics and Automation, pp. 2537–2542 (2012). IEEE
- [4] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press, Cambridge, Massachusetts (2018). Chap. 1,2,3
- [5] Shani, G., Pineau, J., Kaplow, R.: A survey of point-based pomdp solvers. Autonomous Agents and Multi-Agent Systems **27** (2013) <https://doi.org/10.1007/s10458-012-9200-2>
- [6] Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for pomdps. (2003)
- [7] Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces, vol. 4 (2009). <https://doi.org/10.15607/rss.2008.iv.009>
- [8] Icarte, R.T., Waldie, E., Klassen, T.Q., Valenzano, R., Castro, M.P., McIlraith, S.A.: Learning reward machines for partially observable reinforcement learning, vol. 32 (2019)
- [9] Mnih, V., Silver, D., Riedmiller, M.: Playing atari with deep q learning. Nips (2013)
- [10] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518** (2015) <https://doi.org/10.1038/nature14236>
- [11] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Communications of the ACM **60** (2017) <https://doi.org/10.1145/3065386>
- [12] Hausknecht, M., Stone, P.: Deep recurrent q-learning for partially observable mdps, vol. FS-15-06 (2015)
- [13] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9** (1997) <https://doi.org/10.1162/neco.1997.9.8.1735>
- [14] Jakob N. Foerster, N.d.F. Yannis M. Assael, Whiteson, S.: Learning to communicate to solve riddles with deep distributed recurrent q-networks (2016)
- [15] Pengfei Zhu, P.P. Xin Li, Miao, G.: On improving deep reinforcement learning

for pomdps (2017)

- [16] Nicolas Heess, T.P.L. Jonathan J Hunt, Silver, D.: Memory-based control with recurrent neural networks (2015)
- [17] Meng, L., Gorbet, R., Kulic, D.: Memory-based deep reinforcement learning for pomdps. (2021). <https://doi.org/10.1109/IROS51168.2021.9636140>
- [18] Baier, C., Katoen, J.-P.: Principles Of Model Checking vol. 950, (2008)
- [19] Křetínský, J., Meggendorfer, T., Sickert, S.: Owl: A library for ω -words, automata, and ltl, vol. 11138 LNCS (2018). https://doi.org/10.1007/978-3-030-01090-4_34
- [20] Chatterjee, K., Chmelík, M., Gupta, R., Kanodia, A.: Qualitative analysis of pomdps with temporal logic specifications for robotics applications, vol. 2015-June (2015). <https://doi.org/10.1109/ICRA.2015.7139019>
- [21] Sharan, R., Burdick, J.: Finite state control of pomdps with ltl specifications. (2014). <https://doi.org/10.1109/ACC.2014.6858909>
- [22] Bouton, M., Kochenderfer, M.J.: Point-based methods for model checking in partially observable markov decision processes. (2020). <https://doi.org/10.1609/aaai.v34i06.6563>
- [23] Ahmadi, R.. M.; Sharan, Burdick, J.W.: Stochastic finite state control of pomdps with ltl specifications (2020)
- [24] Carr, S., Jansen, N., Wimmer, R., Serban, A., Becker, B., Topcu, U.: Counterexample-guided strategy improvement for pomdps using recurrent neural networks, vol. 2019-August (2019). <https://doi.org/10.24963/ijcai.2019/768>
- [25] Carr, S., Jansen, N., Topcu, U.: Verifiable rnn-based policies for pomdps under temporal logic constraints, vol. 2021-January (2020). <https://doi.org/10.24963/ijcai.2020/570>
- [26] Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Omega-regular objectives in model-free reinforcement learning, vol. 11427 LNCS (2019). https://doi.org/10.1007/978-3-030-17462-0_27
- [27] Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G.J., Lee, I.: Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees, vol. 2019-December (2019). <https://doi.org/10.1109/CDC40024.2019.9028919>
- [28] Oura, R., Sakakibara, A., Ushio, T.: Reinforcement learning of control policy for linear temporal logic specifications using limit-deterministic generalized büchi automata. IEEE Control Systems Letters 4 (2020) <https://doi.org/10.1109/LCSYS.2020.2980552>

- [29] Lin, L.-J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* **8** (1992) <https://doi.org/10.1007/bf00992699>
- [30] Bozkurt, A.K., Wang, Y., Zavlanos, M.M., Pajic, M.: Control synthesis from linear temporal logic specifications using model-free reinforcement learning. (2020). <https://doi.org/10.1109/ICRA40945.2020.9196796>
- [31] Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic büchi automata for linear temporal logic, vol. 9780 (2016). https://doi.org/10.1007/978-3-319-41540-6_17
- [32] Cai, M., Hasanbeig, M., Xiao, S., Abate, A., Kan, Z.: Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE Robotics and Automation Letters* **6** (2021) <https://doi.org/10.1109/LRA.2021.3101544>
- [33] Cai, M., Xiao, S., Li, B., Li, Z., Kan, Z.: Reinforcement learning based temporal logic control with maximum probabilistic satisfaction, vol. 2021-May (2021). <https://doi.org/10.1109/ICRA48506.2021.9561903>
- [34] Mingyu, C., Xiao, S., Li, Z., Kan, Z.: Optimal probabilistic motion planning with potential infeasible ltl constraints. *IEEE Transactions on Automatic Control* (2021) <https://doi.org/10.1109/TAC.2021.3138704>
- [35] Coumans, E., Bai, Y.: PyBullet, a Python module for physics simulation for games, robotics and machine learning. <http://pybullet.org> (2016–2021)
- [36] Oroojlooy, A., Hajinezhad, D.: A review of cooperative multi-agent deep reinforcement learning. *Appl Intell* (2022) <https://doi.org/10.1007/s10489-022-04105-y>
- [37] Zhou, W., Li, J., Zhang, Q.: Joint communication and action learning in multi-target tracking of uav swarms with deep reinforcement learning. *Drones* **6**(11), 339 (2022)