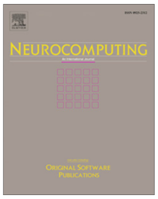
Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Delay-aware model-based reinforcement learning for continuous control

Baiming Chen [a,*], Mengdi Xu [b], Liang Li [a], Ding Zhao [b]

[a] Tsinghua University, Beijing 100084, China
[b] Carnegie Mellon University, Pittsburgh, PA 15213, USA

ARTICLE INFO

Article history:
Received 17 April 2020
Revised 19 October 2020
Accepted 8 April 2021
Available online 13 April 2021
Communicated by Zidong Wang

Keywords:
Model-based reinforcement learning
Markov decision process
Continuous control
Delayed system

ABSTRACT

Action delays degrade the performance of reinforcement learning in many real-world systems. This paper proposes a formal definition of delay-aware Markov Decision Process and proves it can be transformed into standard MDP with augmented states using the Markov reward process. We develop a delay-aware model-based reinforcement learning framework that can incorporate the multi-step delay into the learned system models without learning effort. Experiments with the Gym and MuJoCo platforms show that the proposed delay-aware model-based algorithm is more efficient in training and transferable between systems with various durations of delay compared with state-of-the-art model-free reinforcement learning methods.

1. Introduction

Deep reinforcement learning has made rapid progress in games [1,2] and robotic control [3–5]. However, most algorithms are evaluated in turn-based simulators like Gym [6] and MuJoCo [7], where the action selection and actuation of the agent are assumed to be instantaneous. Action delay, although prevalent in many areas of the real world, including robotic systems [8–10], communication networks [11] and parallel computing [12], may not be directly handled in this scheme.

Previous research has shown that delays would not only degrade the performance of the agent but also induce instability to the dynamic systems [13–15], which is a fatal threat in safety-critical systems like connected and autonomous vehicles (CAVs) [16]. For instance, it usually takes more than 0.4 s for the hydraulic automotive brake system to generate the desired deceleration [10], which could make a huge impact on the planning and control modules of CAVs [17]. The control community has proposed several methods to address this problem, such as using Smith predictor [18,19], Artstein reduction [20,21], finite spectrum assignment [22,23], and H_∞ robust controller [24,25]. Most of these methods depend on accurate models [26,13], which is usually not available in the real-world applications.

Recently, DRL has offered the potential to resolve this issue. The problems that DRL solves are usually modeled as Markov Decision Process (MDP). However, ignoring the delay of agents violates the Markov property and results in partially observable MDPs, or POMDPs, with historical actions as hidden states. From [27], it is shown that solving POMDPs without estimating hidden states can lead to arbitrarily suboptimal policies. Travnik et al. [28] also showed that the traditional MDP is problematic with delays. To retrieve the Markov property, the delayed system was reformulated as an augmented MDP problem such as the work in [29,30]. While the problem was elegantly formulated, the computational cost increases exponentially as the delay increases, which limits the application of this framework. To reduce the computational cost, Walsh et al. [28] proposed a model-based approach to compensating the delay by learning a dynamics model to predict the future state. However, they mainly focused on discrete tasks and could suffer from the curse of dimensionality when discretizing state and action space for continuous control tasks [31]. In addition, the mechanism of model-based reinforcement learning in delayed systems is not fully studied. Most recently, Ramstedt & Pal [32] proposed an off-policy model-free algorithm known as Real-Time Actor-Critic to address the delayed problem by adapting Q-learning to state-value-learning. Though performing well in 1-step delayed systems, this method could still suffer from inefficient learning when used in multi-step delayed systems. The learned policy is also not transferable is the delay step changes with most model-free DRL algorithms. Another direction to address the delay issue is to utilize robust learning for sim-to-real adaptation with domain randomization [33–35] and adversarial learning [36], which can be formalized as a two-player zero-sum game [37,38]. However, most works in this area focus on the noise of physical parameters [33,34] or destabilizing forces [36] instead of

[*] Corresponding author.
E-mail address: cbm17@mails.tsinghua.edu.cn (B. Chen).

https://doi.org/10.1016/j.neucom.2021.04.015
0925-2312/

time-delay. It is also worth mentioning that introducing robustness will result in degraded performance due to conservatism [39].

The review of the literature indicates the lack of an efficient algorithm framework to utilize DRL in time-delayed systems for real-world robotic control tasks. Current methods are either computationally expensive [32,30] or suboptimal [36,34] in tasks with continuous action space. We argue that this results from the underutilization of the dynamics of time-delayed systems. In other words, the mechanism of model-based DRL in delayed systems has not been comprehensively identified and formulated. In this paper, we propose a general delay-aware DRL framework to solve continuous control tasks with both high efficiency and optimality. Our key insight is that the dynamics of time-delayed systems can be explicitly divided into two parts: the known part caused by delays, and the unknown part inherited from the original delay-free systems. Based on this finding, we design an efficient algorithm framework where the multi-step delay is directly incorporated into the learned system models without learning effort, such that the unnecessary computational cost is saved. The main contributions of this paper are listed below:

- We formally define the Delay-Aware MDP and prove that it can be converted to standard MDP via the Markov reward process so that it can be solved within the reinforcement learning architecture.
- We propose a general framework of delay-aware model-based reinforcement learning for continuous control tasks with high efficiency and transferability.
- By synthesizing the state-of-the-art modeling and planning algorithms, we develop the Delay-Aware Trajectory Sampling (DATS) algorithm which can efficiently solve delayed MDPs with minimal degradation of performance.

The rest of the paper is organized as follows. We first review the preliminaries in Section 2 including the definition of Delay-Aware Markov Decision Process (DA-MDP). In Section 3, we formally define the Delay-Aware Markov Reward Process (DA-MRP) and prove its solidity. In Section 4, we introduce the proposed framework of delay-aware model-based reinforcement learning for DA-MDPs with a concrete algorithm: Delay-Aware Trajectory Sampling (DATS). In Section 5.1, we demonstrate the performance of the proposed algorithm in challenging continuous control tasks on Gym and MuJoCo platforms.

2. Preliminaries

2.1. Delay-free MDP and reinforcement learning

The Delay-free MDP framework is suitable to model games like chess and go, where the state keeps still until a new action is executed. The definition of a delay-free MDP is [32]:

Definition 1. A Markov Decision Process (MDP) is characterized by a tuple with

(1) state space $\mathscr{S}$,
(2) action space $\mathscr{A}$,
(3) initial state distribution $\rho : \mathscr{S} \rightarrow \mathbb{R}$,
(4) transition probability $p : \mathscr{S} \times \mathscr{S} \times \mathscr{A} \rightarrow \mathbb{R}$,
(5) reward function $r : \mathscr{S} \times \mathscr{A} \rightarrow \mathbb{R}$.

In the framework of reinforcement learning, the problem is often modeled as an MDP, and the agent is represented by a policy π that directs the action selection, given the current observation. The goal of reinforcement learning is to find the optimal policy π^* that maximizes the expected cumulative discounted reward $\Sigma_{t=0}^{T}\gamma^t r(s_t, a_t)$. Throughout this paper, we assume that we know the reward function r and do not know the transition probability p.

2.2. Delay-aware MDP

The delay-free MDP is problematic with agent delays and could lead to arbitrarily suboptimal policies [27]. To retrieve the Markov property, Delay-Aware MDP (DA-MDP) is proposed [30,32]:

Definition 2. A Delaye-Aware Markov Decision Process $DAMDP(E, n) = (\mathscr{X}, \boldsymbol{\mathscr{A}}, \boldsymbol{\rho}, \boldsymbol{p}, \boldsymbol{r})$ augments a Markov Decision Process $MDP(E) = (\mathscr{S}, \mathscr{A}, \rho, p, r)$, such that

(1) state space $\mathscr{X} = \mathscr{S} \times \mathscr{A}^n$ where n denotes the delay step,
(2) action space $\boldsymbol{\mathscr{A}} = \mathscr{A}$,
(3) initial state distribution[1]

$$\boldsymbol{\rho}(\boldsymbol{x_0}) = \boldsymbol{\rho}(\boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{a_{n-1}}) = \rho(\boldsymbol{s_0}) \prod_{\boldsymbol{i=0}}^{\boldsymbol{n-1}} \delta(\boldsymbol{a_i} - \boldsymbol{c_i}),$$

where $(c_i)_{i=1:n-1}$ denotes the initial action sequence,
(4) transition probability

$$\begin{aligned}
&\boldsymbol{p}(\boldsymbol{x_{t+1}}|\boldsymbol{x_t}, \boldsymbol{a_t}) \\
&= \boldsymbol{p}\left(\boldsymbol{s_{t+1}}, \boldsymbol{a_{t+1}^{(t+1)}}, \ldots, \boldsymbol{a_{t+n}^{(t+1)}} | \boldsymbol{s_t}, \boldsymbol{a_t^{(t)}}, \ldots, \boldsymbol{a_{t+n-1}^{(t)}}, \boldsymbol{a_t}\right) \\
&= \boldsymbol{p}\left(\boldsymbol{s_{t+1}} | \boldsymbol{s_t}, \boldsymbol{a_t^{(t)}}\right) \prod_{\boldsymbol{i=1}}^{\boldsymbol{n-1}} \delta\left(\boldsymbol{a_{t+i}^{(t+1)}} - \boldsymbol{a_{t+i}^{(t)}}\right) \delta\left(\boldsymbol{a_{t+n}^{(t+1)}} - \boldsymbol{a_t}\right),
\end{aligned}$$

(5) reward function

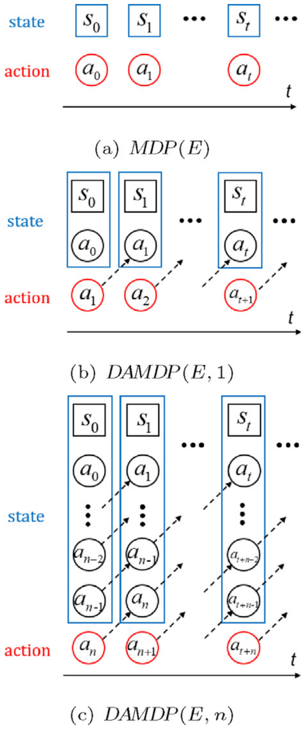
$$\boldsymbol{r}(\boldsymbol{x_t}, \boldsymbol{a_t}) = \boldsymbol{r}(\boldsymbol{s_t}, \boldsymbol{a_t}, \ldots, \boldsymbol{a_{t+n-1}}, \boldsymbol{a_t}) = \boldsymbol{r}(\boldsymbol{s_t}, \boldsymbol{a_t}).$$

The state vector of DA-MDP is augmented with an action sequence being executed in the next n steps where $n \in \mathbb{N}$ is the delay duration. The superscript of $a_{t_1}^{(t_2)}$ means that the action is one element of $\boldsymbol{x_{t_2}}$ and the subscript represents the action executed time. $\boldsymbol{a}_t$ is the action taken at time t in a DA-MDP but executed at time $t + n$ due to the n-step action delay, i.e. $\boldsymbol{a}_t = a_{t+n}$.

Policies interacting with the DA-MDPs, which also need to be augmented since the dimension of state vectors has changed, are denoted by bold $\boldsymbol{\pi}$. Fig. 1, which compares MDP and DA-MDP, shows that the state vector of DA-MDP is augmented with an action sequence being executed in the next n steps.

It should be noted that both action delay and observation delay could exist in real-world systems. However, it has been proved that from the point of view of the learning agent, observation and action delays form the same mathematical problem, since they both lead to the delay between the moment of measurement and the actual action [29]. For simplicity, we will focus on the action delay in this paper, and the algorithm and conclusions should be able to generalize to systems with observation delays. We divide the action delay into two main parts into action selection and action actuation. For action selection, the time length depends on the complexity of the algorithm and the computing power of the processor. System users can limit the action selection time by constraining the searching depth, as in AlphaGo [2]. For action actuation, on the other hand, the actuators (e.g., motors, hydraulic machines) also need time to respond to the selected action. For instance, it usually takes more than 0.4 s for the hydraulic automotive brake system to generate the desired deceleration [10]. The actuation delay is usually decided by the hardware.

[1] δ is the Dirac delta function. If $y \sim \delta(\cdot - x)$ then $y = x$ with probability one.



(a) $MDP(E)$

(b) $DAMDP(E,1)$

(c) $DAMDP(E,n)$

Fig. 1. Comparison between $MDP(E), DMDP(E,1)$ and $DMDP(E,n)$. $n \in \mathbb{N}$ denotes the action delay step. s_t denotes the observed state while a_t denotes the action executed, both at time t. Arrows represent how the action selected in the current time step will be included in the future state.

To formulate a delayed system into a DA-MDP, we must select a proper time step for discretely updating the environment. As shown in Fig. 1c, the action selected at the current time step $\boldsymbol{a}_t$ will be encapsulated in $\boldsymbol{x}_{t+1}$. Thus, $\boldsymbol{a}_t$ must be accessible at time $t+1$ since the agent needs it as the state, which requires the action selection delay to be at most one time step. We satisfy this requirement by making the time step of the DA-MDP larger than the action selection duration.

The above definition of DA-MDP assumes that the delay time of the agent is an integer multiple of the time step of the system, which is usually not true for many real-world tasks like robotic control. For that, Schuitema et al. [40] has proposed an approximation approach by assuming a *virtual* effective action at each discrete system time step, which could achieve first-order equivalence in linearizable systems with arbitrary delay time. With this approximation, the above DA-MDP structure can be adapted to systems with arbitrary-value delays.

3. Delay-aware markov reward process

Our first step is to show that an MDP with multi-step action delays can be converted to a regular MDP problem by state augmentation. We prove the equivalence of these two by comparing their corresponding Markov Reward Processes (MRPs). The delay-free MRP is:

Definition 3. A Markov Reward Process $(\mathscr{S}, \rho, \kappa, \bar{r}) = MRP(MDP(E), \pi)$ can be derived from a Markov Decision Process $MDP(E) = (\mathscr{S}, \mathscr{A}, \rho, p, r)$ with a policy π, such that

$$\kappa(s_{t+1}|s_t) = \int_{\mathscr{A}} p(s_{t+1}|s_t, a)\pi(a|s_t)\ da,$$

$$\bar{r}(s_t) = \int_{\mathscr{A}} r(s_t, a)\pi(a|s_t)\ da,$$

where κ is the sate transition probability and $\bar{r}$ is the state reward function of the MRP. E is the original environment without delays.

In the delay-free framework, at each time step, the agent selects an action based on the current observation. The action will immediately be executed in the environment to generate the next observation. However, if an action delay exists, the interaction manner between the environment and the agent changes, and a different MRP is generated. An illustration of the delayed interaction between agents and the environment is shown in Fig. 2. The agent interacts with the environment not directly but through an action buffer.

Based on the delayed interaction manner between the agent and the environment, the Delay-Aware MRP (DA-MRP) is defined as below.

Definition 4. A Delay-Aware Markov Reward Process $(\mathscr{X}, \boldsymbol{\rho}, \boldsymbol{\kappa}, \bar{\boldsymbol{r}}) = DAMRP(MDP(E), \boldsymbol{\pi}, n)$ can be derived from a Markov Decision Process $MDP(E) = (\mathscr{S}, \mathscr{A}, \rho, p, r)$ with a policy $\boldsymbol{\pi}$ and n-step action delay, such that

(1) state space

$$\mathscr{X} = \mathscr{S} \times \mathscr{A}^n,$$

(2) initial state distribution

$$\boldsymbol{\rho}(\boldsymbol{x_0}) = \boldsymbol{\rho}(\boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{a_{n-1}}) = \rho(\boldsymbol{s_0}) \prod_{\boldsymbol{i=0}}^{\boldsymbol{n-1}} \delta(\boldsymbol{a_i} - \boldsymbol{c_i})$$

where $(c_i)_{i=1:n-1}$ denotes the initial action sequence,
(3) state transition probability

$$\begin{aligned}
&\boldsymbol{\kappa}(\boldsymbol{x_{t+1}}|\boldsymbol{x_t}) \\
&= \boldsymbol{\kappa}\left(\boldsymbol{s_{t+1}}, \boldsymbol{a^{(t+1)}_{t+1}}, \ldots, \boldsymbol{a^{(t+1)}_{t+n}}|\boldsymbol{s_t}, \boldsymbol{a^{(t)}_{t}}, \ldots, \boldsymbol{a^{(t)}_{t+n-1}}\right) \\
&= \boldsymbol{p}(\boldsymbol{s_{t+1}}|\boldsymbol{s_t}, \boldsymbol{a_t}) \prod_{\boldsymbol{i=1}}^{\boldsymbol{n-1}} \delta\left(\boldsymbol{a^{(t+1)}_{t+i}} - \boldsymbol{a^{(t)}_{t+i}}\right) \pi\left(\boldsymbol{a^{(t+1)}_{t+n}}|\boldsymbol{x_t}\right),
\end{aligned}$$

(4) state-reward function

$$\bar{\boldsymbol{r}}(\boldsymbol{x_t}) = \bar{\boldsymbol{r}}(s_t, a_t, \ldots, a_{t+n-1}) = r(s_t, a_t),$$

With Def. 1–4, we are ready to prove that DA-MDP is a correct augmentation of MDP with delay, as stated in Theorem. 1.

Theorem 1. *A policy $\boldsymbol{\pi} : \mathscr{A} \times \mathscr{X} \to \mathbb{R}$ interacting with $DAMDP(E,n)$ in the delay-free manner produces the same Markov Reward Process as $\boldsymbol{\pi}$ interacting with $MDP(E)$ with n-step action delays, i.e.*

$$DAMRP(MDP(E), \boldsymbol{\pi}, n) = MRP(DAMDP(E,n), \boldsymbol{\pi}). \tag{1}$$

Proof. For any $MDP(E) = (\mathscr{S}, \mathscr{A}, \rho, p, r)$, we need to prove Eq. 1 by comparing the elements (i.e., state space, initial distribution, transition probability and state-reward function) of the above two MRPs. Referring to Def. 2 and 3, for $MRP(DMDP(E,n), \boldsymbol{\pi})$, we have

(1) state space $\mathscr{S} \times \mathscr{A}^n$,
(2) initial distribution

$$\begin{aligned}
\boldsymbol{\rho}(\boldsymbol{x_0}) &= \boldsymbol{\rho}(\boldsymbol{s_0}, \boldsymbol{a_0}, \ldots, \boldsymbol{a_{n-1}}) \\
&= \rho(\boldsymbol{s_0}) \prod_{\boldsymbol{i=0}}^{\boldsymbol{n-1}} \delta(\boldsymbol{a_i} - \boldsymbol{c_i}),
\end{aligned}$$

(3) transition probability

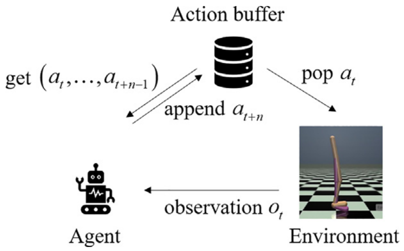


Fig. 2. Interaction manner between a delayed agents and the environment. The agent interacts with the environment not directly but through an action buffer. At time t, the agent get the observation o_t from the environment as well as a future action sequences $(a_t,\dots,a_{t+n-1})$ from the action buffer. The agents then decide their future action a_{t+n} and store them in the action buffer. The action buffer then pops actions a_t to be executed to the environment.

$$\begin{aligned}\kappa(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) &= \int_{\mathscr{A}} \boldsymbol{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{a}_t)\pi(\boldsymbol{a}|\boldsymbol{x}_t)\ \boldsymbol{da}\\ &= \int_{\mathscr{A}} \boldsymbol{p}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\prod_{i=1}^{n-1}\delta\left(\boldsymbol{a}_{t+i}^{(t+1)}-\boldsymbol{a}_{t+i}^{(t)}\right)\delta\left(\boldsymbol{a}_{t+n}^{(t+1)}-\boldsymbol{a}\right)\ \pi(\boldsymbol{a}|\boldsymbol{x}_t)\ \boldsymbol{da}\\ &= \boldsymbol{p}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\ \prod_{i=1}^{n-1}\delta\left(\boldsymbol{a}_{t+i}^{(t+1)}-\boldsymbol{a}_{t+i}^{(t)}\right)\pi\left(\boldsymbol{a}_{t+n}^{(t+1)}|\boldsymbol{x}_t\right),\end{aligned}$$

(4) state-reward function

$$\begin{aligned}\bar{r}(\boldsymbol{x}_t) &= \int_{\mathscr{A}} \boldsymbol{r}(\boldsymbol{x}_t,\boldsymbol{a})\pi(\boldsymbol{a}|\boldsymbol{x}_t)\ \boldsymbol{da}\\ &= \int_{\mathscr{A}} \boldsymbol{r}(\boldsymbol{s}_t,\boldsymbol{a}_t)\ \pi(\boldsymbol{a}|\boldsymbol{x}_t)\ \boldsymbol{da}\\ &= \boldsymbol{r}(\boldsymbol{s}_t,\boldsymbol{a}_t)\int_A \pi(\boldsymbol{a}|\boldsymbol{x}_t)\ \boldsymbol{da}\\ &= \boldsymbol{r}(\boldsymbol{s}_t,\boldsymbol{a}_t).\end{aligned}$$

Since the expanded terms of $MRP(DMG(E,n),\boldsymbol{\pi})$ match the corresponding terms of $DAMRP(MG(E),\boldsymbol{\pi},n)$ (Def. 4), Eq. 1 holds.
□

4. Delay-aware model-based reinforcement learning

Theorem. 1 shows that instead of solving MDPs with action delays, we can alternatively solve the corresponding DA-MDPs. From the transition function of a $DAMDP(E,n)$ with multi-step delays

$$\boldsymbol{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{a}_t) = \boldsymbol{p}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\prod_{i=1}^{n-1}\delta\left(\boldsymbol{a}_{t+i}^{(t+1)}-\boldsymbol{a}_{t+i}^{(t)}\right)\delta\left(\boldsymbol{a}_{t+n}^{(t+1)}-\boldsymbol{a}_t\right), \tag{2}$$

we see that the dynamics is divided into the unknown original dynamics $p(s_{t+1}|s_t,a_t)$ and the known dynamics $\prod_{i=1}^{n}\delta\left(a_{t+i}^{(t+1)}-a_{t+i}^{(t)}\right)\delta(a_{t+n}-\boldsymbol{a}_t)$ caused by the action delays. Thus, solving DA-MDPs with standard reinforcement learning algorithms will suffer from the curse of dimensionality if assuming a completely unknown environment. In this section, we propose a delay-aware model-based reinforcement learning framework to achieve high computational efficiency.

As mentioned, RTAC [32] has been proposed to deal the delay problem. However, we will show that this method is only efficient for 1-step delayed systems. When $n=1$ for $DMDP(E,n)$, any transition (s_t,a_t,s_{t+1}) in the replay buffer is always a valid transition in the Bellman equation with the state-value function as

$$\begin{aligned}&v^{\pi}_{DA-MDP(E,n)}(x_t)\\ &= r(s_t,a_t)+\mathbb{E}_{\mathbb{s}_{\mathbb{t}+1}}\left[\mathbb{E}_{a_{\mathbb{t}}}\left[\mathbb{v}^{\boldsymbol{\pi}}_{DA-MDP(E,n)}(\mathbb{s}_{\mathbb{t}+1},\mathbb{a}_{\mathbb{t}+1},\dots,\mathbb{a}_{\mathbb{t}+\mathbb{n}-1},a_{\mathbb{t}})\right]\right],\end{aligned}$$

where $a_t\sim\boldsymbol{\pi}(\cdot|x_t)$, and $s_{t+1}\sim p(\cdot|s_t,a_t)$. However, when considering the multi-step delay, i.e., $n\geqslant 2$, it is challenging to use off-policy model-free reinforcement learning because augmented transitions need to be stored and we only learn the effect of an action on the state-value function after n-step updates of the Bellman equation. Also, the dimension of the state vector x increases with the delay step n, resulting in the exponential growth of the state-space.

Another limitation of model-free methods for DA-MDPs is that it can be difficult to transfer the learned knowledge (e.g., value functions, policies) when the action delay step n changes because the input dimensions of the value functions and policies depend on the delay step n. The agent must learn again from scratch whenever the system delay changes, which is usual in real-world systems.

The problems of model-free methods have motivated us to develop model-based reinforcement learning (MBRL) methods to combat the action delay. MBRL tries to solve MDPs by learning the dynamics model of the environment. Intuitively, we can inject our knowledge into the learned model without leaning effort. Based on this intuition, in this paper, we propose a delay-aware MBRL framework to solve multi-step DA-MDPs which can efficiently alleviate the aforementioned two problems of model-free methods. From Eq. 2, the unknown part is exactly the dynamics that we learn in MBRL algorithms for delay-free MDPs. In our proposed framework, only $p(s_{t+1}|s_t,a_t)$ is learned and the dynamics caused by the delay is combined with the learned model by adding action delays to the interaction scheme. As mentioned, the learned dynamics model is transferable between systems with different delay steps, since we can adjust the interaction scheme based on the delay step (See Section 5.3 for an explanation of the transfer performance).

The proposed framework of delay-aware MBRL is shown in Algorithm 1. In the **for** loop, we are solving a planning problem, given a dynamics model with an initial action sequence. For that, the learned model is used not only for the optimal control but also for the state prediction to compensate for the delay effect. By iteratively training, we gradually improve the model accuracy and obtain better planning performance and, especially in high-reward regions.

Algorithm 1 Delay-Aware Model-Based Reinforcement Learning

```
Input: action delay step $n$, initial actions $(a_i)_{i=0,\dots,n-1}$, and task horizon $T$
Output: learned transition probability $\widetilde{p}$
Initialize replay buffer $\mathbb{D}$ with a random policy.
forEpisode $k=1$ to $K$do
  Train a dynamics model $\widetilde{p}$ given $\mathbb{D}$.
  Optimize action sequence $a_{n+1:T}$ with initial actions $(a_i)_{i=0,\dots,n-1}$ and estimated system dynamics $\widetilde{p}$
  Record experience: $\mathbb{D}\leftarrow\mathbb{D}\cup(s_t,a_t,s_{t+1})_{t=0:T}$.
end for
```

4.1. Delay-aware trajectory sampling

Recently, several MBRL algorithms have been proposed to match the asymptotic performance of model-free algorithms on challenging benchmark tasks, including probabilistic ensemble with trajectory sampling (PETS) [41], model-based policy optimization (MBPO) [42], model-based planning with policy networks (POPLIN) [43], etc. In this section, we will combine the state-of-the-art PETS algorithm with the proposed delay-aware MBRL framework to generate a new method for solving DA-MDPs. We name the method as the Delay-Aware Trajectory Sampling (DATS). The complete algorithm is shown in Algorithm 2.

Algorithm 1: Delay-Aware Trajectory Sampling

```
Input: action delay step $n$, initial actions $(a_i)_{i=0,\dots,n-1}$, task
  horizon $T$, planning horizon $m$
Output: learned transition probability $\widetilde{p}$
Initialize transition buffer $\mathbb{D}$ with a random policy.
for Episode $k = 1$ to $K$ do
  Train a probabilistic dynamics model $\widetilde{p}$ given $\mathbb{D}$.
  Initialize action buffer $\mathbb{A} = (a_i)_{i=0,\dots,n-1}$
  for Time $t = 0$ to $T - n$do
    Observe $s_t$
    for Sampled $a_{t+n:t+n+m} \sim \mathrm{CEM}(\cdot)$do
      Concatenate $a_{t+n:t+n+m}$ with $a_{t:t+n-1}$
      Propagate state particles $s_\tau$ using $\widetilde{p}$.
      Evaluate actions as $\sum_{\tau=t}^{t+n+m} r(s_\tau, a_\tau)$
      Update $\mathrm{CEM}(\cdot)$ distribution.
    end for
    Pick the first action $a_{t+n}$ from optimal action sequence
    and store in $\mathbb{A}$
  end for
  Record experience: $\mathbb{D} \leftarrow \mathbb{D} \cup (s_t, a_t, s_{t+1})_{t=0:T}$.
end for
```

In DATS, the dynamic model is represented by an ensemble of probabilistic neural networks that output Gaussian distributions which helps model the aleatoric uncertainty [41]. The negative log likelihood is used as the loss function. Suppose the output Gaussian distribution of the learned transition probability $\widetilde{p}$ parameterized by θ given the state-action pair at time t is:

$$\widetilde{p}_\theta(s_{t+1}|s_t, a_t) = \mathcal{N}\left(\mu_\theta(s_t, a_t), \Sigma_\theta(s_t, a_t)\right),$$

then the loss is calculated as:

$$\mathrm{loss}(\theta) = \sum_{t=1}^{N} \left[\mu_\theta(s_t, a_t) - s_{t+1}\right]^\top \Sigma_\theta^{-1}(s_t, a_t) \left[\mu_\theta(s_t, a_t) - s_{t+1}\right] + \log\det \Sigma_\theta(s_t, a_t),$$

where N is the sample size. The use of the ensemble can help incorporate the epistemic uncertainty of the dynamic model and approximate the Bayesian posterior [44,45]. We use ensembles of M bootstrap models and denotes the parameter of the m-th model with θ_m, then the transition probability model is calculated by the mean of the bootstrap models: $\widetilde{p}_\theta = \frac{1}{M}\sum_{m=1}^{M} \widetilde{p}_{\theta_m}$. Each bootstrap model is trained with its unique dataset $\mathbb{D}_m$ generated from the whole dataset $\mathbb{D}$ by random sampling. Throughout this paper, we set $M = 5$ since it is sufficient for all experiments.

The planning of action sequences applies the concept of model predictive control (MPC) with the cross-entropy method (CEM) for elite selection of the sampled action sequences. In the most inner **for** loop of Algorithm 2, with the current state s_t, we first propagate state particles with the same action sequence $a_{t:t+n-1}$ to make various estimates of the future state s_{t+n} , and then use sampled action sequences $a_{t+n:t+n+m}$ to predict $s_{t+n+1:t+n+1+m}$ for each particle. In this way, the uncertainty of the learned model is considered in both state-prediction and planning phases, which improves the robustness of the algorithm.

Model-based methods have a natural advantage when dealing with multi-step DA-MDPs when compared with model-free methods. With model-free methods, the effect of an action on the state-value function can only be learned after n-time updates of the Bellman equation. The agent implicitly wastes both time and effort to learn the known part of system dynamics caused by action delay since it does not understand the meaning of the elements in the state vectors. As mentioned, the advantage of model-based methods is that they incorporate delay effect into the system dynamics without extra learning (see Section 5.2 for a performance comparison between model-free and model-based methods).

5. Experiments

5.1. Reinforcement learning in delayed systems

Experiments are conducted across four OpenAI Gym/Mujoco [6,7] environments for continuous control: `Pendulum`, `Cartpole`, `Walker2d` and `Ant` as shown in Fig. 3. The details of the environments are described below [46]. The reward functions used in the experiments are shown in Table 1.

`Pendulum`. A single-linked pendulum is fixed on the one end, with an actuator located on the joint. In this version of the problem, the pendulum starts in a random position, and the goal is to swing it up to keep it upright. Observations include the joint angle θ_t and the joint angular velocity $\dot{\theta}_t$. The reward penalizes position and speed deviations from the upright equilibrium and the magnitude of the control input.

`Cartpole`. A pole is connected to the cart through an unactuated joint, and the cart moves along a frictionless track. Control the system by applying a real-number force to the cart. The pole starts upright, and the goal is to prevent it from falling over. Let θ_t be the angle of the pole away from the upright vertical position, and x_t be the position where the cart leaves the center of the rail at time t. The 4-dimensional observation at time t is $(x_t, \theta_t, \dot{x}_t, \dot{\theta}_t)$. A reward of $+1$ is provided for every timestep that the pole remains upright.

`Walker2d`. Walker2d is a 2-dimensional bipedal robot, consisting of 7 rigid links, including a torso and 2 legs. There are 6 actuators, 3 for each leg. The observations include the (angular) position and speed of all joints. The reward is the x direction speed plus the penalty for the distance to a target height and the magnitude of control input. The goal is to walk forward as fast as possible while keeping the standing height with minimal control input.

`Ant`. Ant is a 3-dimensional 4-legged robot with 13 rigid links, including a torso and 4 legs. There are 8 actuators at the joints, 2 for each leg. The observations include the (angular) position and speed of all joints. The reward is the x direction speed plus penalty for the distance to a target height and the magnitude of control input. The goal is to walk forward as fast as possible, and approximately maintain the normal standing height with minimal control input.

Among the 4 continuous control tasks, the tasks of `Walker2d` and `Ant` are considered more challenging than `Pendulum` and `Cartpole` indicated by the dimension of dynamics.

In experiments, we add delays manually by revising the interaction framework between the agents and the environments if needed.

To show the advantage of DATS, we use 6 algorithms:

- **DDPG** ($n = 0$): Deep deterministic policy gradient [31] is a model-free reinforcement learning algorithm for continuous control. Only the performances at the maximum time step are visualized.
- **SAC** ($n = 0$): Soft actor-critic [47] is a state-of-the-art model-free reinforcement learning algorithm serving as another model-free baseline. Only the performances at the maximum time step are visualized.
- **PETS** ($n = 0$): The PETS algorithm [41] is implemented in the non-delayed environment without action delays, providing the performance upper bound for algorithms in delayed environments.

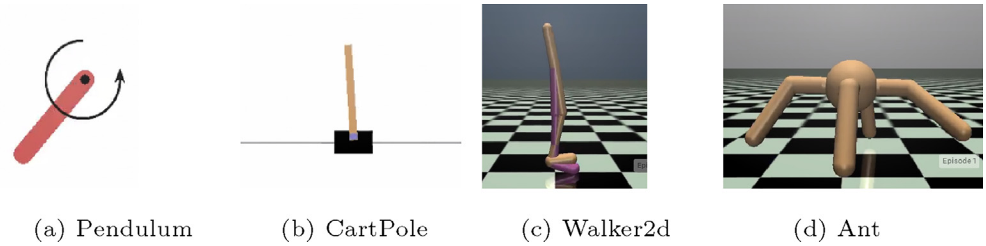

(a) Pendulum (b) CartPole (c) Walker2d (d) Ant

Fig. 3. Benchmark environments.

Table 1
Reward Functions.

Environment	Reward Function
Pendulum	$-\cos\theta_t - 0.1\dot{\theta}_t^2 - 0.001a_t^2$
CartPole	$\cos\theta_t - 0.01x_t^2$
Walker2d	$\dot{x}_t - 0.1\|a_t\| - 3.0 \times (z_t - 1.3)^2$
Ant	$\dot{x}_t - 0.1\|a_t\| - 3.0 \times (z_t - 0.57)^2$

- **PETS** $(n = 1)$: The PETS algorithm is blindly implemented in the 1-step delayed environment without modeling action delays, which makes it delay-unaware.
- **W-PETS** $(n = 1)$: The PETS algorithm is augmented to solve DA-MDPs with $n = 1$. However, it inefficiently tries to learn the **whole** dynamics $p(x_{t+1}|x_t, a_t)$ as shown in Eq. 2 including the known part caused by actions delays.
- **DATS** $(n = 1)$: DATS is our proposed method as in Algorithm 2. It incorporates the action delay into the framework and only learns the unknown original dynamics $p(s_{t+1}|s_t, a_t)$ as shown in Eq. 2.

Each algorithm is run with 10 random seeds in each environment. Fig. 4 shows the algorithmic performances. As the model-free baseline, DDPG and SAC are not as efficient as PETS in the four environments when there are no delays. Providing the performance upper bound, PETS $(n = 0)$ achieves the best performance in all tasks. However, when the action delay exists, the delay-unaware algorithm PETS $(n = 1)$ has the worst performance in all experiments. It fails to learn policies for challenging tasks that need accurate transition dynamics for planning in `Walker2d` (Fig. 4c) and `Ant` (Fig. 4d). W-PETS achieves similar performance with PETS $(n = 0)$ in `Pendulum` and `Cartpole`. But its performance also degrades a lot when the task gets more difficult since it has to learn the dynamics of the extra n dimensions of states caused by the n-step action delays (Fig. 4c and 4d). DATS performs the same as PETS $(n = 0)$ for the four tasks, i.e., action delays do not affect DATS.

The reason why DATS in delayed environment matches the asymptotic performance of PETS in the non-delayed environment is that the quality and quantity of transitions (s_t, a_t, s_{t+1}) used for model training in DATS are almost the same with PETS, despite the action delay. The slight difference is due to the distribution shift caused by the predefined initial actions, which has minimal influence on the overall performance if the task horizon is long enough compared to the action delay step.

Our codes are available online[2] and can be directly used for any continuous control tasks on the OpenAI Gym/Mujoco platform. The reward functions are set consistent with the benchmark paper [46].The results on 8 representative environments are summarized in Table 2. The algorithms are trained with 10,000 timesteps in `Pendulum` and `Cartpole` and 200,000 timesteps in other environments. The results show the advantage of the proposed algorithm DATS in one-step-delayed $(n = 1)$ systems, as well as the sample-efficiency of model-based algorithms (DATS, PETS) compared to model-free ones (DDPG, SAC).

5.2. Model-based vs model-free

To show the advantage of the proposed delay-aware MBRL framework when dealing with multi-step delays, we compare the model-free algorithm RTAC [32] and the proposed model-based DATS. RTAC is suitable for solving DA-MDPs and is modified based on SAC, but as explained in Section 4, RTAC can avoid extra learning only when the action delay is exactly one-step.

We test them in the simple environment `Pendulum` and the complex environment `Walker2d` with various delay step n. The learning curves in Fig. 5. show that DATS outperforms RTAC in efficiency and stability. RTAC degrades significantly as the delay step increases, even for the simple task `Pendulum`, as shown in Fig. 5b. The reason is that with the original dynamics of `Pendulum` and `Walker2d` fixed, the extra dynamics caused by the action delay rapidly dominates the dimension of the state space of the learning problem as the delay step increases, and exponentially more transitions are needed to sample and learn.

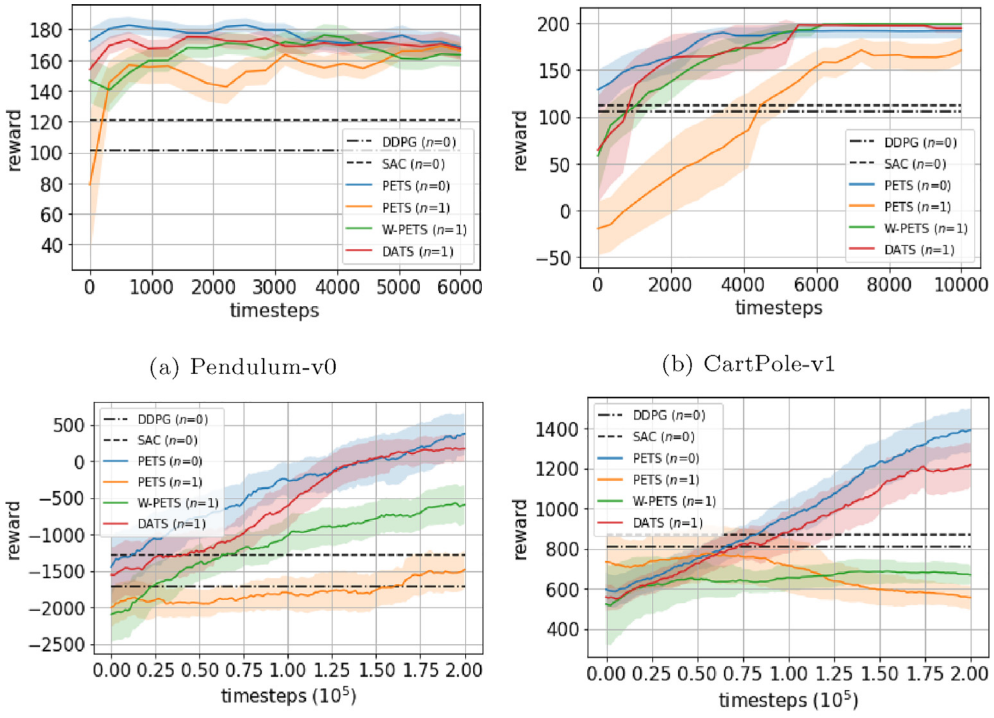
5.3. Transferable knowledge

In this section, we show the transferability of the knowledge learned by DATS. We first learn several dynamics models $\{\tilde{p}_i\}$ in `Pendulum` and `Walker2d` with DATS, where $i = 1, 2, 4, 8$ denotes the action delay step during training. The learned models are then tested in environments with n-step action delays $(n = 1, 2, 4, 8, 16)$. We train the dynamics model in each environment with the same amount of transitions (s_t, a_t, s_{t+1}): 2,000 for `Pendulum` and 200,000 for `Walker2d`. The planning method and hyper-parameters stay the same as those in Algorithm 2. RTAC provides the model-free baseline for each environment. Recall that since RTAC is a model-free algorithm, when changing the delay steps, it must learn from scratch.

The reward matrix in Table 3 shows that DATS performs well even when the delay step is twice larger than the maximum step during model-training $(n = 16)$ for `Pendulum` and `Walker2d`. We infer that the learned knowledge (dynamics in this case) is transferable, i.e., when the action delay of the system changes, the estimated dynamics are still useful by simply adjusting the known part of the dynamics caused by the action delay. On the other hand, RTAC performs poorly as the delay step increases since the dimension of the state space grows and the agent has to spend more effort to learn the delay dynamics. Notably, the learned knowledge of model-free methods cannot transfer when the delay step changes.

The results suggest that the transferability of DATS makes it suitable for Sim-to-Real tasks when there are action delays in real systems, and that the delay step during model training does not

[2] https://github.com/baimingc/delay-aware-MBRL.



(a) Pendulum-v0

(b) CartPole-v1

(c) Walker2d-v1

(d) Ant-v1

Fig. 4. Performances (means and standard deviations of rewards) of different MBRL algorithms in Gym environments. The environment is non-delayed for DDPG, SAC and PETS ($n = 0$) and is one-step-delayed for other algorithms. DATS is the proposed algorithm. The results indicate that the performance degradation resulting from the environment action delay is minimal when using DATS, while the delay-unaware algorithm PETS ($n = 1$) fails to learn good polices for challenging tasks like `Walker2d` and `Ant`.

Table 2
Final performance for Gym environments.

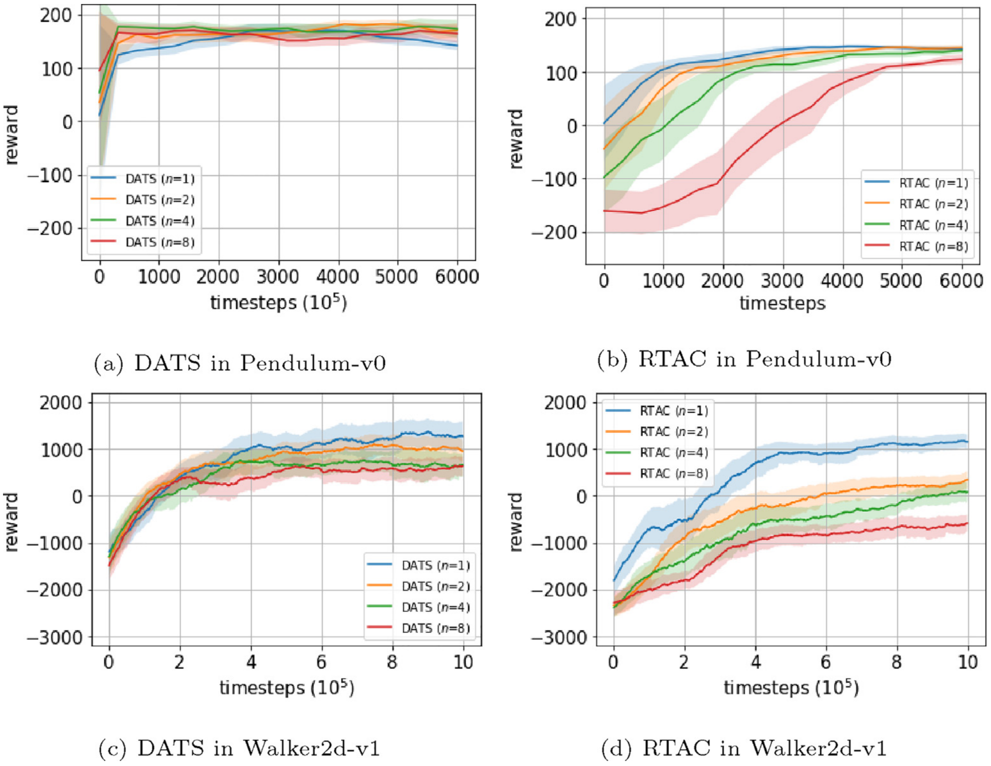
Environment	DDPG ($n = 0$)	SAC ($n = 0$)	PETS ($n = 0$)	PETS ($n = 1$)	W-PETS ($n = 1$)	DATS ($n = 1$)
Pendulum	103.1 ± 25.3	121.6 ± 28.1	188.7 ± 8.3	171.0 ± 11.4	180.1 ± 9.8	185.6 ± 9.6
CartPole	106.5 ± 29.6	117.3 ± 27.0	197.7 ± 1.3	180.0 ± 8.5	196.6 ± 2.6	198.1 ± 1.2
Walker2d	−1722.4 ± 363.0	-1257.5 ± 328.2	465.8 ± 182.2	-1510.7 ± 261.0	-515.4 ± 275.9	386.2 ± 224.1
Ant	813.3 ± 145.0	869.4 ± 120.8	1406.8 ± 96.0	575.2 ± 62.6	861.4 ± 49.5	1215.8 ± 114.7
Hopper	927.4 ± 429.8	729.4 ± 377.9	1124.9 ± 320.3	507.4 ± 441.2	846.2 ± 396.2	1194.1 ± 317.2
HalfCheetah	1277.1 ± 580.1	1780.7 ± 697.2	2686.3 ± 420.9	1281.8 ± 640.7	1840.3 ± 548.2	2404.0 ± 485.7
Swimmer	250.6 ± 22.4	227.9 ± 31.6	308.3 ± 37.1	160.3 ± 39.2	244.7 ± 26.8	285.3 ± 33.5
SlimHumanoid	1219.1 ± 719.4	1228.4 ± 592.8	2085.3 ± 671.5	451.9 ± 497.2	1588.2 ± 621.7	1894.5 ± 640.7

have to be equal to the delay step in a real system. Therefore, if the delay steps of the real-world tasks are known and fixed, we can incorporate the delay effect with the original dynamics learned in the delay-free simulator, and obtain highly efficient Sim-to-Real transformations.

6. Conclusion and discussion

This paper proposed a general delay-aware MBRL framework which solves multi-step DA-MDPs with high efficiency and transferability. Our key insight is that the dynamics of DA-MDPs can be divided into two parts: the known part caused by delays, and the unknown part inherited from the original delay-free MDP. The proposed delay-aware MBRL framework learns the original unknown dynamics and incorporates the known part of the dynamics explicitly. We also provided an efficient implementation of delay-aware MBRL as DATS by combining a state-of-the-art modeling and planning method, PETS. The experiment results showed that the performance of PETS in instantaneous environments is similarly to the performance of DATS in delayed environments with respect to delay duration. Moreover, the learned dynamics by DATS is transferable when the time of action delay changes, thus making DATS the preferred algorithm for tasks in real-world systems.

There are two promising directions to extend this study. 1) The delay effect needs to be further explored in multi-agent systems, where the delay of one agent could spread to other coupled agents. For example, in tasks involving communications between agents, the action delay of a speaker would give rise to observation delays of all listeners subscribing to this speaker. 2) It is worth studying how to estimate the delay time if it is not known a priori. This problem is highly related to online system identification, but it is unclear how to efficiently incorporate it with DRL in delayed systems.



(a) DATS in Pendulum-v0

(b) RTAC in Pendulum-v0

(c) DATS in Walker2d-v1

(d) RTAC in Walker2d-v1

Fig. 5. Performances (means and standard deviations of rewards) of DATS and RTAC in Gym environments with different action delay steps. The model-based algorithm DATS outperforms the model-free algorithm RTAC in terms of efficiency and stability. RTAC degrades significantly as the delay step increases.

Table 3
Reward matrix of DATS and RTAC.

(a) Pendulum-v0

n	DATS				RTAC
	$\tilde{p}_1$	$\tilde{p}_2$	$\tilde{p}_4$	$\tilde{p}_8$	
1	154.10 ± 14.86	156.37 ± 13.29	**163.29** ± **16.03**	149.78 ± 13.778	121.36 ± 12.63
2	**163.92** ± **15.23**	162.93 ± 14.26	155.90 ± 16.11	160.07 ± 18.30	109.44 ± 12.58
4	160.39 ± 12.63	162.87 ± 16.21	**171.53** ± **10.85**	166.29 ± 14.22	80.15 ± 27.94
8	163.29 ± 15.53	151.20 ± 13.44	166.37 ± 13.32	**166.59** ± **10.59**	-110.28 ± 58.89
16	153.41 ± 17.35	**159.09** ± **19.88**	153.89 ± 14.22	149.90 ± 16.86	-122.98 ± 64.82

(b) Walker2d-v1

n	DATS				RTAC
	$\tilde{p}_1$	$\tilde{p}_2$	$\tilde{p}_4$	$\tilde{p}_8$	
1	471.34 ± 426.26	**524.76** ± **387.67**	496.13 ± 442.89	395.78 ± 409.98	-471.13 ± 896.28
2	549.73 ± 410.76	487.32 ± 334.49	**527.98** ± **477.19**	492.56 ± 490.01	-754.42 ± 722.79
4	**485.29** ± **438.98**	439.23 ± 529.39	248.60 ± 611.82	552.91 ± 410.76	-1252.47 ± 710.10
8	356.93 ± 431.58	438.82 ± 563.13	**482.09** ± **316.34**	247.97 ± 595.63	-1766.85 ± 404.28
16	292.38 ± 521.86	311.44 ± 409.80	**473.97** ± **309.81**	401.34 ± 634.12	-2173.87 ± 625.76

CRediT authorship contribution statement

Baiming Chen: Conceptualization, Methodology, Software, Validation, Writing - original draft, Funding acquisition. **Mengdi Xu:** Methodology, Formal analysis. **Liang Li:** Resources, Supervision. **Ding Zhao:** Resources, Writing - review & editing, Supervision, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by China Scholarship Council.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602.

[2] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529(7587) (2016) 484.

[3] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning, 2015, pp. 1889–1897.

[4] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking deep reinforcement learning for continuous control, in: International Conference on Machine Learning, 2016, pp. 1329–1338.

[5] J. Hwangbo, I. Sa, R. Siegwart, M. Hutter, Control of a quadrotor with reinforcement learning, IEEE Robot. Autom. Lett. 2 (4) (2017) 2096–2103.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540.

[7] E. Todorov, T. Erez, Y. Tassa, Mujoco: a physics engine for model-based control, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2012, pp. 5026–5033.

[8] T. Imaida, Y. Yokokohji, T. Doi, M. Oda, T. Yoshikawa, Ground-space bilateral teleoperation of ets-vii robot arm by direct bilateral coupling under 7-s time delay condition, IEEE Trans. Robot. Autom. 20 (3) (2004) 499–511.

[9] M. Jin, S.H. Kang, P.H. Chang, Robust compliant motion control of robot with nonlinear friction using time-delay estimation, IEEE Trans. Industr. Electron. 55 (1) (2008) 258–269.

[10] F.P. Bayan, A.D. Cornetto, A. Dunn, E. Sauer, Brake timing measurements for a tractor-semitrailer under emergency braking, SAE International Journal of Commercial Vehicles 2 (2009–01-2918) (2009) 245–255.

[11] S.B. Moon, P. Skelly, D. Towsley, Estimation and removal of clock skew from network delay measurements, in: IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320), vol. 1, IEEE, 1999, pp. 227–234.

[12] R. Hannah, W. Yin, On unbounded delays in asynchronous parallel fixed-point algorithms, J. Sci. Comput. 76 (1) (2018) 299–326.

[13] K. Gu, S.-I. Niculescu, Survey on recent results in the stability and control of time-delay systems, J. Dyn. Syst. Meas. Control 125 (2) (2003) 158–165.

[14] L. Dugard, E.I. Verriest, Stability and Control of Time-Delay Systems, vol. 228, Springer, 1998.

[15] L. Chung, C. Lin, K. Lu, Time-delay control of structures, Earthq. Eng. Struct. Dyn. 24 (5) (1995) 687–701.

[16] S. Gong, J. Shen, L. Du, Constrained optimization and distributed computation based car following control of a connected and autonomous vehicle platoon, Transp. Res. Part B: Methodol. 94 (2016) 314–334.

[17] J. Ploeg, N. Van De Wouw, H. Nijmeijer, Lp string stability of cascaded systems: application to vehicle platooning, IEEE Trans. Control Syst. Technol. 22 (2) (2013) 786–793.

[18] K.J. Astrom, C.C. Hang, B. Lim, A new smith predictor for controlling a process with an integrator and long dead-time, IEEE Trans. Autom. Control 39 (2) (1994) 343–345.

[19] M.R. Matausek, A. Micic, On the modified smith predictor for controlling a process with an integrator and long dead-time, IEEE Trans. Autom. Control 44 (8) (1999) 1603–1606.

[20] Z. Artstein, Linear systems with delayed controls: a reduction, IEEE Trans. Autom. Control 27 (4) (1982) 869–879.

[21] E. Moulay, M. Dambrine, N. Yeganefar, W. Perruquetti, Finite-time stability and stabilization of time-delay systems, Syst. Control Lett. 57 (7) (2008) 561–566.

[22] A. Manitius, A. Olbrot, Finite spectrum assignment problem for systems with delays, IEEE Trans. Autom. Control 24 (4) (1979) 541–552.

[23] S. Mondié, W. Michiels, Finite spectrum assignment of unstable time-delay systems with a safe implementation, IEEE Trans. Autom. Control 48 (12) (2003) 2207–2212.

[24] E.T. Jeung, J.H. Kim, H.B. Park, et al., Robust controller design for uncertain systems with time delays: Lmi approach, Automatica 32 (8) (1996) 1229–1231.

[25] L. Mirkin, On the extraction of dead-time controllers from delay-free parametrizations, IFAC Proc. Vol. 33 (23) (2000) 169–174.

[26] S.-I. Niculescu, Delay Effects on Stability: A Robust Control Approach, vol. 269, Springer Science & Business Media, 2001.

[27] S.P. Singh, T. Jaakkola, M.I. Jordan, Learning without state-estimation in partially observable markovian decision processes, Machine Learning Proceedings 1994, Elsevier (1994) 284–292.

[28] J.B. Travnik, K.W. Mathewson, R.S. Sutton, P.M. Pilarski, Reactive reinforcement learning in asynchronous environments, Front. Robot. AI 5 (2018) 79.

[29] K.V. Katsikopoulos, S.E. Engelbrecht, Markov decision processes with delays and asynchronous cost collection, IEEE Trans. Autom. Control 48 (4) (2003) 568–574.

[30] T.J. Walsh, A. Nouri, L. Li, M.L. Littman, Learning and planning in environments with delayed feedback, Auton. Agent. Multi-Agent Syst. 18 (1) (2009) 83.

[31] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971.

[32] S. Ramstedt, C. Pal, Real-time reinforcement learning, Advances in Neural Information Processing Systems (2019) 3067–3076.

[33] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, V. Vanhoucke, Sim-to-real: learning agile locomotion for quadruped robots, arXiv preprint arXiv:1804.10332.

[34] A. Rajeswaran, S. Ghotra, B. Ravindran, S. Levine, Epopt: learning robust neural network policies using model ensembles, arXiv preprint arXiv:1610.01283.

[35] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), IEEE, 2017, pp. 23–30.

[36] L. Pinto, J. Davidson, R. Sukthankar, A. Gupta, Robust adversarial reinforcement learning, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 2817–2826.

[37] Z. Cao, C.-T. Lin, Reinforcement learning from hierarchical critics, arXiv preprint arXiv:1902.03079.

[38] Z. Cao, K. Wong, Q. Bai, C.-T. Lin, Hierarchical and non-hierarchical multi-agent interactions based on unity reinforcement learning, in: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, 2020, pp. 2095–2097.

[39] H. Zhang, Y. Yu, J. Jiao, E.P. Xing, L.E. Ghaoui, M.I. Jordan, Theoretically principled trade-off between robustness and accuracy, arXiv preprint arXiv:1901.08573.

[40] E. Schuitema, L. Buşoniu, R. Babuška, P. Jonker, Control delay in reinforcement learning for real-time dynamic systems: a memoryless approach, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2010, pp. 3226–3231.

[41] K. Chua, R. Calandra, R. McAllister, S. Levine, Deep reinforcement learning in a handful of trials using probabilistic dynamics models, Advances in Neural Information Processing Systems (2018) 4754–4765.

[42] M. Janner, J. Fu, M. Zhang, S. Levine, When to trust your model: Model-based policy optimization, arXiv preprint arXiv:1906.08253.

[43] T. Wang, J. Ba, Exploring model-based planning with policy networks, arXiv preprint arXiv:1906.08649.

[44] I. Osband, C. Blundell, A. Pritzel, B. Van Roy, Deep exploration via bootstrapped dqn, in: Advances in Neural Information Processing Systems, 2016, pp. 4026–4034.

[45] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, Advances in Neural Information Processing Systems (2017) 6402–6413.

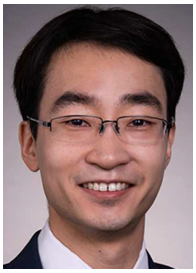
[46] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, J. Ba, Benchmarking model-based reinforcement learning, arXiv preprint arXiv:1907.02057.

[47] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor, arXiv preprint arXiv:1801.01290.

Baiming Chen received his B.S. degree from Tsinghua University in 2017. He is now working towards Ph.D. degree in mechanical engineering from the School of Vehicle and Mobility at Tsinghua University, Beijing, China. His research interests include autonomous driving, reinforcement learning and control, multi-agent systems.

Mengdi Xu is a PhD student in Mechanical Engineering, Carnegie Mellon University. She received her Master in Robotics degree from the Johns Hopkins University and Bachelor degree in Vehicle Engineering from Tsinghua University. Her research interests lie in Robotics, Cognitive Science, Machine Learning and design, with applications on human-robot interaction, medical robots and autonomous driving.

Liang Li received his Ph.D. degree from the Department of Automotive Engineering at Tsinghua University in 2008. Since 2017, he has been a tenured professor in Tsinghua University. From November 2011 to December 2012, he was a researcher with the Institute for Automobile Engineering, RWTH Aachen University, Aachen Germany. His research interests mainly include vehicle dynamics and control, adaptive and nonlinear system control, and hybrid vehicle develop and control. Dr. Li received the China Automotive Industry Science and Technology Progress Award for his achievements in hybrid electrical bus in 2012, and won the National Science Fund for Excellent Young Scholars of the Peoples Republic of China in 2014.



Ding Zhao received his Ph.D. degree in 2016 from the University of Michigan, Ann Arbor. He is currently an Assistant Professor at Department of Mechanical Engineering, Carnegie Mellon University. His research focuses on the intersection of robotics, machine learning, and design, with applications on autonomous driving, connected/smart city, energy efficiency, human-machine interaction, cybersecurity, and big data analytics.