IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 13, 2020, accepted December 9, 2020, date of publication December 15, 2020,
date of current version December 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3044945

rocorl: Transferable Reinforcement Learning-Based Robust Control for Cyber-Physical Systems With Limited Data Updates

GWANGPYO YOO, MINJONG YOO, IKJUN YEOM, AND HONGUK WOO, (Member, IEEE)
Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Honguk Woo (hwoo@skku.edu)

This work was supported in part by the DNA+ Drone Technology Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT) under Grant 2020M3C1C2A01080819, in part by the ICT Creative Consilience Program supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP) under Grant IITP-2020-0-01821, in part by the Basic Science Research Program through the NRF funded by MSIT under Grant 2020R1A2C2009809, and in part by Samsung Electronics.

ABSTRACT Autonomous control systems are increasingly using machine learning technologies to process sensor data, making timely and informed decisions about performing control functions based on the data processing results. Among such machine learning technologies, reinforcement learning (RL) with deep neural networks has been recently recognized as one of the feasible solutions, since it enables learning by interaction with environments of control systems. In this paper, we consider RL-based control models and address the problem of temporally outdated observations often incurred in dynamic cyber-physical environments. The problem can hinder broad adoptions of RL methods for autonomous control systems. Specifically, we present an RL-based robust control model, namely **rocorl**, that exploits a hierarchical learning structure in which a set of low-level policy variants are trained for stale observations and then their learned knowledge can be transferred to a target environment limited in timely data updates. In doing so, we employ an autoencoder-based observation transfer scheme for systematically training a set of transferable control policies and an aggregated model-based learning scheme for data-efficiently training a high-level orchestrator in a hierarchy. Our experiments show that **rocorl** is robust against various conditions of distributed sensor data updates, compared with several other models including a state-of-the-art POMDP method.

INDEX TERMS Cyber-physical system, real-time data, reinforcement learning, model-based learning, stale observations.

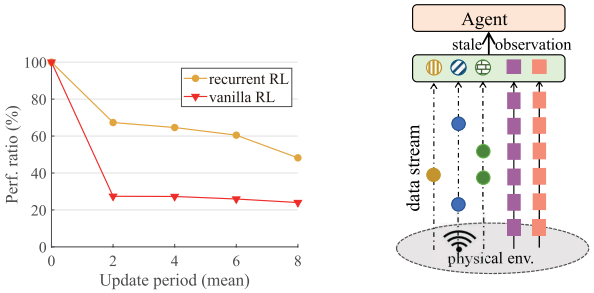
I. INTRODUCTION

Recently, deep reinforcement learning (RL) has gained much attention and been recognized as a practical solution to implement autonomous control functions for intelligent cyber-physical systems, e.g, vehicles [1], [2], robots [3], drones [4], and others. Such an RL-based control function normally relies on timely observations by several sensing mechanisms as part of system operations to acquire state information about its surroundings and make informed decisions about control actions in response to state changes. In a networked system environment, on the other hand, sensors might not be centralized but distributed, and thus they need to communicate with a controller to keep observations highly up-to-date. In this case, while an RL-based control function can be formulated upon real-time data models commonly used for navigation and tracking [5], [6], its underlying communication infrastructure might have inherent restriction in completely meeting data synchronization requirements of real-time data [7]. That is, a real-time data model necessitates continual updates of remote sensor data, but network resource constraints (e.g., limited bandwidth, intermittent connections, transmission delays) inherently limit the timeliness of those updates. This inconsistency between real-time data models

The associate editor coordinating the review of this manuscript and approving it for publication was Jie Tang.

and underlying network constraints can cause robustness issues for RL-based controls, particularly when observations are dependent on the timeliness of sensor data updates.

Figure 1 briefly depicts a performance degradation pattern in our RL-based control tests in which agents with neural networks utilize sensor data updates (i.e., periodic updates with variable intervals) to make decisions about control operations. In this simulation test, we configure variable periods of data updates from sensors to an RL agent to generate temporally outdated data input (stale observations) for the agent, as abstracted in Figure 1(b). We first test an agent with *vanilla* RL (the red line in Figure 1(a)) that is trained without any specific consideration on stale observations but tested with stale observations. The simulation environment will be detailed in Section V. The "Perf. ratio" on the Y-axis represents the relative performance degraded from the "normal" case where no stale observation exists. Its definition is in Eq. (44). The low performance of *vanilla* RL, less than 30% of the normal case, indicates the negative effect of stale observations on RL-based controls. Considering temporal features of sensor data, we also evaluate several different types of RL policy network structures which are known to be effective for partial observation problems. *Recurrent* RL (the orange line in Figure 1(a)) shows the test result when an LSTM (long short term memory)-based RL agent exploits a sequence of sensor data as input to its recurrent policy network. As shown, the longer the update periods (on the X-axis), the lower the performance. This result is contrary to our expectation that some effective rules for handling stale observations could be learned by this seemingly proper network structure that makes use of a sequence of observations for state estimation. It turns out that *recurrent* RL barely shows robust performance when having limited data updates, e.g., where the mean $\geq$ 2 on the X-axis. We will provide our analysis on this result in Section II-A.



(a) Effect of limited updates (b) Stale observations

FIGURE 1. The effect of limited data updates on the control function operated by RL: in (a), the X-axis denotes mean values of variable data update periods in discrete timesteps, where the standard deviation sets to be 2.5 for lognormal distribution, and the Y-axis denotes the performance ratio calculated by Eq. (44). In (b), the observations are acquired from sensor data streams with variable update periods, thus inherently containing stale information, i.e., represented by circle data.

In this paper, motivated by the test results aforementioned, we address the problem of stale observations for RL-based controls in a networked environment with random periods of sensor data updates. Such a control process with limited data updates is not an unusual structure, since geographically distributed sensing mechanisms are often part of a cyber-physical system; numerous sensing and actuation components are connected on a network. We consider a self-operation device as a target cyber-physical system, for which controls are governed by an RL agent and observations contain temporally outdated, stale information.

To do so, we employ a hierarchical learning structure by which a set of low-level transferable policies are systematically trained and then their learned knowledge can be transferred to a complex target environment having limited data updates. In generating a set of low-level policy variants that can collectively handle stale observations in a hierarchy, we leverage the autoencoder structure to reduce the difference of state estimates between different environments with respect to variable staleness. We also exploit a high-level orchestrator, which can be seen as a meta policy that learns rules for continuously selecting the most appropriate low-level policy from several over time. Furthermore, for tackling the issue of sample inefficiency induced by the meta policy on the higher level which has only fewer data sampling points than controlling timesteps, we take an efficient model-based learning approach.

The contributions of our paper are as follows.

- We address the stale observation problem for RL-based controls of a network-constrained cyber-physical system. To do so, we employ transfer learning driven through a hierarchical model that consists of a high-level orchestrator and a set of low-level transferable control policies (in Section II). We name our proposed model **rocorl** (robust control using RL).
- We present the hierarchical policy training schemes of **rocorl**: *autoencoder-based observation transfer* that facilitates the systematic generation of transferable control policies upon stale observations (in Section III), and *aggregated policy learning* that accelerates model-based learning for the orchestrator upon insufficient rollout samples (in Section IV).
- We conduct a case study with the Airsim simulator [8] for autonomous drone operation in edge computing. The case study demonstrates robust performance and wide applicability of our approach for adopting RL in a networked environment, e.g., showing up to 11% performance improvement over a state-of-the-art RL method in the configuration of variable update periods (in Section V).

II. OVERALL SYSTEM

In this section, we explain our assumption on limited updates of spatio-temporal sensor data in a target networked environment, describe the problem of stale observations caused by limited data updates, and then briefly present our approach to the problem.

A. STALE OBSERVATION PROBLEM

In a cyber-physical system with distributed sensors, data updates are considered periodic or sporadic. Temporally outdated observations are often inherent and they cannot be completely avoided especially due to underlying network limitations. Throughout this paper, we refer to this situation, temporally outdated observations incurred by variable periods of sensor data updates, as the stale observation problem for RL-based control systems, which makes it hard to train an end-to-end RL model.

1) STALE OBSERVATION

In general, an RL-based control system operates upon real-time data streams that are managed by a set of networked modules, where the data streams continuously maintain observations for making informed decisions about device controls. As shown in Figure 2, we consider two types of real-time data according to the location of sensing mechanisms: *global* data e managed by an *edge* service and *local* data d managed by a *device*. That is, we refer to global data as near-edge, i.e., long-range low-resolution sensor data that can be accessed in real-time on the edge-side, while we refer to local data as near-device, i.e., short-range high-resolution sensor data that can be accessed in real-time on the device-side. We focus on the issue related to a network-constrained environment, and accordingly we presume that most sensor data cannot be both due to the limitation of timely edge-device synchronization. For a self-operation vehicle, sensor measurements of the vehicle can be near-device local data (e.g., lidar, radar, visual images), whilst long-range real-time map information around the vehicle can be near-edge global data (e.g., the location of moving obstacles or traffic information).

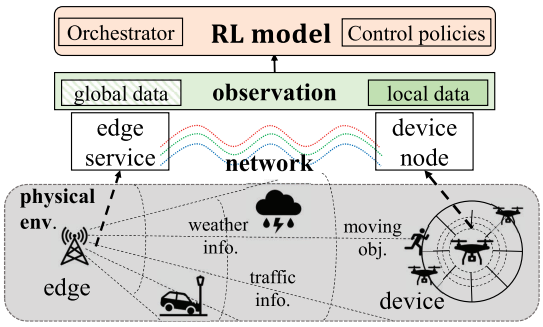


FIGURE 2. The overall structure of rocorl: we consider a networked system that consists of a self-operation device node and an edge service, where the device has its locally observed short-range high-resolution sensor data (local data), the edge service has its globally observed long-range low-resolution sensor (global data), and they communicate for conducting device control operations with network limitations.

In our notation, we represent (near-edge) global and (near-device) local data as

$$e,\ \mathcal{E} = \{e\}; \quad d,\ \mathcal{D} = \{d\}. \tag{1}$$

We assume that $\mathcal{E}$ and $\mathcal{D}$ are bounded regions on $\mathbb{R}^n$ for some $n \in \mathbb{N}$ without loss of generality. We also represent a state at discrete timestep $t \in \mathbb{N}_0$ as

$$s_t = (e_t, d_t), \quad s_t \in \mathcal{S} = \mathcal{E} \times \mathcal{D}. \tag{2}$$

Considering limited updates, we then formalize stale observations on the device-side as

$$\omega_t = (e_{t-i}, d_t), \quad \omega_t \in \Omega = \mathcal{E} \times \mathcal{D} \tag{3}$$

for some $i \in \mathbb{N}_0$ that specifies the elapsed period from the last update. Similarly, stale observations on the edge-side can be represented as $\omega_t = (e_t, d_{t-i})$. In most cases, we consider stale observations from the device perspective, since we concentrate on RL-based controls for a self-operation device that exploits its observations to make control decisions.

2) RL TRAINING ISSUE WITH STALE OBSERVATION

Here, we describe the problem of RL training upon stale observations. In principle, RL is a method that finds an optimal policy π,

$$\pi : \ \mathcal{S} \times \mathcal{A} \to [0, 1] \tag{4}$$

that maximizes the expected reward sum over an MDP (Markov decision process) with

$$(\mathcal{S}, \mathcal{A}, \mathcal{P}, R). \tag{5}$$

Note that $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition probability, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is an immediate expected reward function. Then, for a θ-parameterized policy π_θ, the objective function is represented as

$$J(\theta) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi_\theta \\ s_t, T \sim \mathcal{P}_{\pi_\theta}}} \left[\sum_{t=0}^{T} R(s_t, a_t) \right] \tag{6}$$

where $\mathcal{P}_{\pi_\theta}$ is a probability distribution of states $s \in \mathcal{S}$ induced by π_θ and $T \in \mathbb{N}_0$ denotes the episode timesteps.

In case that stale observations are involved, however, this objective function rarely optimizes the policy π_θ due to the fact that states s_t cannot be accurately estimated. In the RL research community, for the cases when the observation is partial or limited, i.e., a POMDP (partially observable MDP) where an RL agent is not able to observe true states from its environment, several methods including the recurrent policy [9], [10] have been investigated. In general, a POMDP is represented by

$$(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \Omega, \mathcal{O}) \tag{7}$$

where Ω is a set of observations, $\mathcal{O} : \mathcal{S} \to (\Omega \to [0, 1])$ is an observation probability, and the other elements are the same as in Eq. (5). Specifically, given a history h_t of observation and action samples, the recurrent policy (e.g., LSTM policy) parameterized by θ, say $\pi_\theta^{(r)}$, intends to infer an action from the history h_t.

$$\pi_\theta^{(r)}(h_t = (\omega_{t-i}, a_{t-i} \ldots, \omega_t)) \ \mapsto \ a_t \tag{8}$$

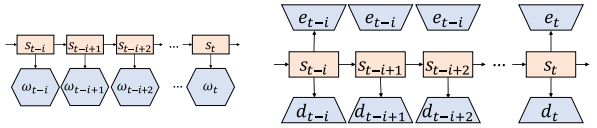
This work is based on a property of POMDPs such that the observation probability $\mathcal{O}$ is a fixed probability distribution that completely depends on the current state [9], [10].

$$\mathcal{O}(s_t) = \Pr[\omega_t | s_t] \ \text{ for } \omega_t \in \Omega \tag{9}$$

However, given stale observations, the assumption on a fixed observation distribution does not hold. Suppose that at i previous step, global data are updated from the edge service to the device. For the current state s_t, we have $\mathcal{O}(s_t) = \Pr[\omega_t = (e_{t-i}, d_t)|s_t]$ which is not fixed, according to Eq. (3); the distribution of observations ω_t is not completely determined by the current state s_t, i.e.,

$$\Pr[(e_{t-i}, d_t)|s_t] \neq \Pr[(e_{t-i}, d_t)|s_t, s_{t-i}] \tag{10}$$

since e_{t-i} is necessarily part of s_{t-i}. The difference between a typical POMDP and a process with stale observations is illustrated in their transition diagrams in Figure 3.



(a) POMDP case (b) Stale observation case

FIGURE 3. A typical POMDP and a process with state observations.

In the following, we show why this different temporal dependency leads to incorrect calculations of policy gradients when a conventional end-to-end RL model is applied for a process with stale observations. Let H be a random variable form of a history h. Then, we can rewrite the objective function in Eq. (6) in

$$J(\theta) = \int_H \Pr(H|\theta) R(H) dH \tag{11}$$

where $R(H)$ is the sum of rewards obtained during a history $h \sim H$. By taking the log derivative trick and Monte Carlo approximation in N-episodes of history samples, we obtain

$$\begin{aligned} \nabla_\theta J &= \mathbb{E}_H \left[\nabla_\theta \log \Pr[H] R(H) \right] \\ &\approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \Pr[H^{(i)}] R(H^{(i)}). \end{aligned} \tag{12}$$

In a POMDP, the log derivative for model parameters θ in $\nabla_\theta \log \Pr[h]$ computed by the observed history h can be acquired by realizing that the conditional probability of a particular history is the product of all observation and action samples [10]. Accordingly, the log derivative term becomes

$$\nabla_\theta \log \Pr[H^{(i)}] = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta^{(r)}(a_t | h_t^{(i)}) \tag{13}$$

where $\pi_\theta^{(r)}$ denotes our target θ-parameterized policy. This is valid with the fixed distribution assumption in Eq. (9). Combining Eq. (12) and (13), we obtain the estimated gradients of the objective function

$$\nabla_\theta J \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta^{(r)}(a_t | h_t^{(i)}) R_t^{(i)}. \tag{14}$$

While the gradient calculation in Eq. (14) is intended for optimizing $\pi_\theta^{(r)}$ for a typical POMDP, stale observations in Eq. (10) make Eq (13) invalid, thereby resulting in inaccurate gradient values when a gradient-based optimization method is used. This explains the performance degradation previously shown in Figure 1, and motivates us to reformulate the stale observation problem into a hierarchical form, as we will present in the following.

B. OVERALL APPROACH

To tackle the aforementioned issue of policy gradient methods related to stale observations, we exploit the hierarchical structure of RL processing as shown in Figure 4, where a set of low-level control policies are maintained, and over time, one of them is continuously selected and used for performing rollouts. We name this hierarchical RL model **rocorl**, where the selection is made by a high-level decision maker, called *orchestrator*.

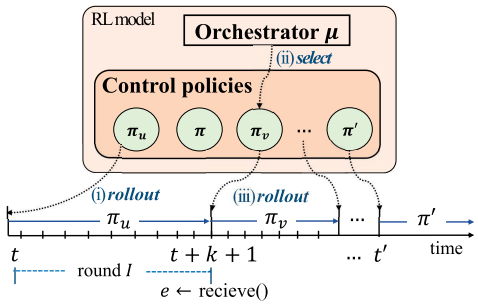


FIGURE 4. Hierarchical controls on rounds.

Consider a control procedure at a specific timestep t: (i) on the device-side, one of the control policies (say π_u) continuously rolls out with respect to its projection data from stale observations ω_t in Eq. (3), where near-device local data d_t are maintained update-to-date, while near-edge global data e_t often become stale. (ii) Suppose that at some timestep $(t+k+1)$ after a certain amount of time, an available connection is made between the device and the edge service, and up-to-date data can be shared through the connection. The orchestrator μ selects a policy (say π_v) at the same time. (iii) This enables the device to switch its running policy from π_u to π_v, if needed. Then, π_v keeps rolling out from the timestep $(t+k+1)$ to the next update. We refer to the time period between two successive updates with a variable time interval as a *round*. These steps of hierarchical controls over successive rounds are depicted in Figure 4. We presume that such update periods are neither configurable nor deterministic, considering the variability of underlying network conditions. That is, neither edge-side nor device-side modules know the next update time in advance.

As we will explain in Section III-A, an inter-round process can be abstracted as an MDP. We thus take a conventional RL method for training the orchestrator,

$$\mu \ : \ \mathcal{S}^{\mu} \times \Pi \rightarrow [0, 1] \tag{15}$$

where $\mathcal{S}^{\mu}$ denotes the state set for the orchestrator and Π denotes a set of control policies. On the other hand, we take several steps for systematically generating the set of control policies,

$$\Pi = \{\pi_1, \ldots, \pi_m\}, \quad \pi_j \in \Pi \ : \ \Omega^{(j)} \times \mathcal{A} \rightarrow [0, 1] \tag{16}$$

where $\Omega^{(j)}$ denotes the observation space for π_j.

To generate such a set of control policies that are capable of collectively dealing with variable staleness, we take a systematical training scheme that exploits the feature compression capability of autoencoder networks. The autoencoder-based scheme is capable of creating a set of control policies that can correspond to different spatio-temporal observation spaces, while aiming at reducing the difference of state distributions of different environments for each control policy. Thus, the scheme renders the control policies transferable between different environments with respect to limited data updates and data staleness.

Furthermore, we address the difficulty in training the orchestrator policy, which requires relatively long training times due to the temporal abstraction of a two-level hierarchy. To mitigate the difficulty, we adapt a model-based RL structure by employing the efficient policy parameter aggregation from both model-based and model-free learning. We will explain these policy training schemes in the next sections, the autoencoder-based observation transfer in Section III and the aggregated policy learning in Section IV.

III. HIERARCHICAL RL CONTROL

The **rocorl** model consists of an orchestrator μ and a set of control policies $\Pi = \{\pi_1, \ldots, \pi_m\}$, and it makes two-level inferences: (i) selecting a policy for each round by the orchestrator when a connection is made, and (ii) conducting low-level control actions by a selected control policy over timesteps within a round. Algorithm 1 represents the two-level inference, where the first inference function called, `inference`(μ, s), corresponds to the former in Eq (15) where the orchestrator μ selects a control policy based on its state s, and the second one corresponds to the latter in Eq (16) where a selected control policy π_v calculates an action a based on its observation transfer from observations ω. The `inference`$(\cdot)$ function yields an action, given a policy and an observation (or a state). The `transfer`$(\cdot)$ function conducts observation transfer for π_v, from ω to $\omega^{(v)}$, which will be explained in Section III-B.

To make the inference robust against stale observations, we leverage a transfer learning mechanism via hierarchical RL, aiming to adapt the knowledge of RL policies from a learnable environment (without stale observations) for a target, hard-to-learn environment (with stale observations).

Algorithm 1 Two-Level Inference

```
Control policies Π = {π_j}, orchestrator policy μ
// on the edge-side
loop
  s.e ← e ← GetData () // retrieve global data
  if Connected () then
    s.d ← UpdateData (e) // update between edge and device
    SetPolicy (Inference (μ, s)) // set policy for device
  end if
end loop

// on the device-side
loop
  ω.d ← GetData () // retrieve local data
  if Connected () then
    ω.e ← UpdateData (d) // update between edge and device
    π_v = GetPolicy () // update control policy by orchest. μ
  end if
  // projection map Φ_v in Eq. (25)
  a ← Inference (π_v, Transfer (Φ_v, ω)) // get control action
  Execute (a) // execute control action
end loop
```

In the following, we show how to train the policies in a hierarchy of **rocorl**. We first provide the proof that a decision process by the orchestrator μ conforms to an MDP between successive rounds. We then show how to train each control policy $\pi \in \Pi$ by employing the *autoencoder-based observation transfer* scheme that alters the observation formulation according to selective global data for each policy $\pi_j \in \Pi$.

A. ORCHESTRATION IN ROUNDS AS AN MDP

To show that the inter-round orchestration conforms to an MDP, we exploit the uniformization property that can turn a Semi-MDP (SMDP) into an equivalent MDP [11], [12]. An SMDP is used for representing a generalized decision process with variable time intervals (i.e., holding times) between successive actions. For an SMDP, the distribution of holding times $\tau(s, a, s')$ is specified for $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ in addition to the MDP representation in Eq. (5)

In the following, we first formulate the orchestrator μ as an SMDP, $(\mathcal{S}^{\mu}, \mathcal{A}^{\mu}, \mathcal{P}^{\mu}, R^{\mu}, \tau)$, and then finds its equivalent MDP. Given a set of states $\mathcal{S}$ and a set of control policies Π in our model, obviously we have $\mathcal{S}^{\mu} = \mathcal{S}$ and $\mathcal{A}^{\mu} = \Pi$ since the orchestrator μ performs macro-actions such as selecting a policy from Π for each round without any modification on underlying states. Given $\pi \in \Pi$ for a round I, further suppose that the holding time of I is given k (i.e., $k \sim \tau$). We then have the reward function $R^{\mu} \ : \ \mathcal{S} \times \Pi \rightarrow \mathbb{R}$ based on averaging over I,

$$R^{\mu}(s_t, \pi) = \frac{1}{k+1} \sum_{i=t}^{t+k} R(s_i, \pi(\omega_i)) \tag{17}$$

where t is the starting timestep in $I : [t, t+k]$. Furthermore, we represent the transition of the orchestrator μ according to its hierarchy by

$$\mathcal{P}^{\mu}(s, \pi, s', k) = \Pr[s' = s_{t+k+1} | s_t, \pi, k]. \tag{18}$$

In our model, the holding times τ correspond to the variable periods of rounds which are completely governed by

underlying network conditions. We thus establish the SMDP formulation for μ using Eq. (17)-(18) and network-dependent τ values.

Now, we can find such an MDP $M' = (\mathcal{S}', \mathcal{A}', \mathcal{P}', R')$ equivalent to the SMDP that we established above for a round $I : [t, t+k]$. It is obvious that we have

$$\mathcal{S}' = \mathcal{S}^{\mu},\ \mathcal{A}' = \mathcal{A}^{\mu} = \Pi \tag{19}$$

because they have nothing to do with the temporal abstraction by the holding times τ. Furthermore, we have $R' = R^{\mu}$ since the holding time effect can be removed when average rewards are taken. Regarding $\mathcal{P}'$, we have the conditional expectation over the holding time $k \sim \tau$,

$$\mathcal{P}' = \mathop{\mathbb{E}}_{k\sim\tau}\left[\mathcal{P}(s,\pi,s')|k\right] = \mathcal{P}^{\mu}(s,\pi,s',T(s,\pi)) \tag{20}$$

for π during I, where T denotes the expected holding time such as

$$T(s,\pi) = \int_{s'\in\mathcal{S}} \mathop{\mathbb{E}}_{\tau}\left[\tau(s,\pi,s')\right] d\Pr[s'|s,\pi]. \tag{21}$$

To maintain the Markov property on $\mathcal{P}'$, we design our hierarchical model to make the observation space of each control policy strictly confined within a single round. We then establish the equivalent MDP by Eq. (19) and (20).

The equivalent MDP enables the orchestrator to learn macro-action rules in a hierarchy, similarly to other hierarchical RL methods such as the option framework [13]. Given a set of policies $\Pi = \{\pi_1, \ldots, \pi_m\}$, it is possible to train the orchestrator μ using a conventional RL method with the objective function,

$$J(\mu) = \mathop{\mathbb{E}}_{\substack{\pi_n\sim\mu \\ k\sim\tau}}\left[\sum_{n=1}^{N} R^{\mu}(s_{t_n}, \pi_n)\right] \tag{22}$$

where $\{s_{t_n}\}_1^N$ is a finite subsequence of $\{s_t\}$ whose elements correspond to the starting time of rounds $I : [t, t+k]$.

B. TRAINING CONTROL POLICIES

As theoretically discussed in Section II-A and empirically shown in Figure 1, it is non-trivial to have a properly learned model in environments with stale observations, when using a conventional end-to-end RL method. Therefore, in **rocorl**, we train each control policy instead in a *normal* environment that is particularly set to have no stale observations (i.e., no restriction on data updates), and adapt it for a target environment having limited data updates.

Here, we describe how to use autoencoders for training a set of control policies. Later, we will explain how **rocorl** makes the control policies transferable from the normal to the target environment, by exploiting different observation spaces and combining a high-level orchestrator with the control policies in a hierarchy in Section III-C.

For training several control policies to be transferable, we employ the *autoencoder-based observation transfer* scheme. We first define an individual policy as a composite function $g \circ f$ such as

$$f : \Omega^{\#} \to \mathcal{Z},\ \ g : \mathcal{Z} \to \mathcal{A}. \tag{23}$$

The function f intends for exploiting a decomposable observation space and increasing the diversity in latent representation, while the function g conducts a common decision-making task. We refer to such a function f as a *observation transfer* and g as a common decider.

To generate a set of observation transfer in a systematic way, we adopt the autoencoder structure. (i) We first exploit the policy π well-learned for the normal environment M^{nm} to obtain samples S of full observation trajectory. (ii) Then, we train the autoencoder ($dec \circ f_{\text{full}} : s \mapsto z \mapsto s$) using $s \in S$, where the full observation space Ω is exploited at this time. Let the encoder part of the learned autoencoder be

$$f_{\text{full}} :\ \Omega \to \mathcal{Z} \quad \text{where}\ \ \omega \in \Omega,\ z \in \mathcal{Z}. \tag{24}$$

(iii) We then obtain restricted observations $\Omega^{(j)}$ from Ω with projection maps Φ_j for $j = 1, \ldots, m-1$.

$$\Phi_j : \Omega \to \Omega^{(j)} \quad \text{where}\ \ \Omega^{(j)} \subset \Omega,\ \omega^{(j)} \in \Omega^{(j)} \tag{25}$$

For instance, a projection can be made from Ω by excluding all (or some) features of global data. For each projection map, we then obtain another observation transfer f_j that is learnable by the pair samples from $\Omega^{(j)}$ and $\mathcal{Z}$,

$$f_j :\ \Omega^{(j)} \to \mathcal{Z}. \tag{26}$$

Since each $z \in \mathcal{Z}$ for $\Omega^{(j)}$ is equal to that in Eq. (24), each observation transfer f_j can be supervised-learned using relevant loss such as similarity distance from ground truth states [14],

$$L(f_j, \omega) = \|f_j(\Phi_j(s)) - z\| \tag{27}$$

for $j = 1, \ldots, m-1$, where $z = f_{\text{full}}(s)$. This learning process is iteratively conducted for different projection maps. To obtain the projection maps, we first group global data using their correlation into c sets. Then, for each set, including or not including it, we can create c^2 combinations for partial observation spaces, while several candidates turn out to be lower-performance policies than others and so they can be pruned.

(iv) To obtain the common decider g, we also train the policy $\pi = g \circ f_{\text{full}}$ for the fixed f_{full}. Finally, given a set of m observation transfers $\{f_1, \ldots, f_{m-1}, f_{\text{full}}\}$, we combine each with the common decider g, thereby achieving a set of m policies Π,

$$\{\pi_1 = g \circ f_1, \ldots, \pi_{m-1} = g \circ f_{m-1}, \pi_m = g \circ f_{\text{full}}\}. \tag{28}$$

Algorithm 2 shows the aforementioned steps (i)-(iv) for generating a set of control policies. The `SLTrain`$(\cdot)$ function is the implementation of conventional supervised learning algorithms, given a target model and labeled samples. Similarly, the `RLTrain`$(\cdot)$ function is the implementation of conventional RL algorithms for the normal environment, given a target policy. Note that we obtain the normal environment

Algorithm 2 Training a Set of Control Policies

set normal env., `RLTrain` (π) // RL training
// (i) collect samples using π in normal environment
samples $D \leftarrow \emptyset$
while $|D| < K$ **do**
  $done \leftarrow$ False, $t \leftarrow 0$
  while not $done$ **do**
    `Append` (D, s_t) // append states to samples D
    $a_t \leftarrow$ `Inference` (π, s_t)
    $s_{t+1}, done \leftarrow$ `EnvStep` (a_t)
    $t \leftarrow t + 1$
  end while
end while
// (ii) train autoencoder with samples D
autoencoder $dec \circ f_{\text{full}}$
`SLTrain` ($dec \circ f_{\text{full}}, D$) // supervised learning Eq. (24)
// (iii) train observation transfers f_j
for $j = 1, \ldots, m-1$ **do**
  samples $(\Omega^{(j)} \times \mathcal{Z}) \leftarrow \emptyset$ // samples of $\Omega^{(j)} \times \mathcal{Z}$
  for $s \in D$ **do**
    // extract samples for observation transfer Eq. (25)
    $\omega^{(j)} \leftarrow$ `Transfer`$(\Phi_j, s), z \leftarrow$ `Inference`(f_{full}, s)
    `Append` $((\Omega^{(j)} \times \mathcal{Z}), (\omega^{(j)}, z))$ // append pair $(\omega^{(j)}, z)$
  end for
  `SLTrain` $(f_j, (\Omega^{(j)} \times \mathcal{Z}))$ // for $f_j(\omega^{(j)}) \mapsto z$ Eq. (26)
end for
// (iv) train common decider g
`RLTrain` $(g \circ f_{\text{full}})$ // RL training
return $\Pi = \{\ \pi_1 = g \circ f_1, \ldots, \pi_m = g \circ f_{\text{full}}\ \}$ // Eq. (28)

from our target environment by statically configuring no limitation on global data updates. This normal environment setting is only for training each policy by Algorithm 2. All the tests are done under the target environment with various stale observations and non-deterministic round lengths.

C. POLICY TRANSFERABILITY

Given a target environment with stale obervations $M^{st} = (\mathcal{S}, \mathcal{A}, R, \mathcal{P}, \Omega, \mathcal{O})$ where we have $\mathcal{S} = \Omega$ as a set by Eq. (2) and (3), we can naturally induce such a normal MDP environment $M^{nm} = (\mathcal{S}, \mathcal{A}, R, \mathcal{P})$. Suppose we have optimal policies π^*_{st} for M^{st} and π^*_{nm} for M^{nm}. Then, it is possible to obtain identical state distributions which are induced by those policies [15].

$$\mathcal{P}_{\pi^*_{nm}} \approx \mathcal{P}_{\pi^*_{st}} \tag{29}$$

This leads to the same identical property for the reward distributions induced by the policies. That is, if there exist only morphological differences between the two environments, it is possible to transfer knowledge between the environments [15]. In our case, the normal M^{nm} and stale M^{st} environments have only morphological differences upon $\mathcal{O}$ and Ω, because they are the same except whether or not data updates are limited. In line with Eq. (29), then, our goal with respect to training an orchestrator can be formulated as the below, minimizing the statistical distance between M^{nm} and M^{st}.

$$\begin{aligned}\mu^* &= \underset{\mu}{\text{argmin}} \|\mathcal{P}_{\pi^*_{nm}} - \mathcal{P}_{\mu}\| \\ &= \underset{\mu}{\text{argmin}} \|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\mu}\| \\ &= \underset{\mu}{\text{argmax}}\, J(\mu)\end{aligned} \tag{30}$$

Here, $\mathcal{R}$ denotes a reward distribution, $\|\cdot\|$ is a norm between two probability measures (e.g., L_1 norm), and J is in Eq. (22).

In the following, we show that a set of control policies obtained by Algorithm 2 and a well-trained orchestrator together minimize the statistical distance of normal and target environments, as represented by the hierarchical learning objective in Eq (30).

Suppose that we have an orchestrator μ and a set of control policies Π in Eq. (28) trained according to the loss in Eq. (27). Further suppose that the target environment M^{st} is Lipschitz [16], [17] with respect to actions. That is, for actions $a, a' \in \mathcal{A}$, there exist $K_1, K_2 \geq 0$ such that

$$\begin{aligned}&\sup_{s \in \mathcal{S}} \|\mathcal{P}(s, a, \cdot) - \mathcal{P}(s, a', \cdot)\| \leq K_1 \|a - a'\| \\ &\sup_{s \in \mathcal{S}} |R(s, a) - R(s, a')| \leq K_2 \|a - a'\|.\end{aligned} \tag{31}$$

Then, we have M^{nm} is also Lipschitz, since M^{nm} is naturally induced by M^{st}; recall that M^{nm} shares $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, and R with M^{st}.

Given the hypothesis in Eq. (31), here, we show that the orchestrator μ with the policy set Π satisfies Eq. (30) with a bounded error. Specifically, we show that there is a fixed number $K > 0$ such that

$$\|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\mu}\|_1 \leq K \min_{j=1,\ldots,\text{full}} \mathbb{E}[L(f_j, \omega)|s] \tag{32}$$

where $\|\cdot\|_1$ is the L_1 norm.

First, we show that the error bound of each control policy $\pi_j \in \Pi$ is

$$\|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\pi_j}\|_1 \leq K \mathbb{E}\left[L(f_j, \omega))\right]. \tag{33}$$

For each observation transfer function f_j and $z = f_{\text{full}}(s) \in \mathcal{Z}$, consider the estimation error $\varepsilon_j > 0$ (i.e., loss in Eq. (27)) such that

$$\|z - \hat{z}^{(j)}\| < \varepsilon_j \tag{34}$$

holds where $\hat{z}^{(j)}$ is the estimated feature by f_j, i.e., $\hat{z}^{(j)} = f_j(\Phi_j(\omega)) \in \mathcal{Z}$ and z is the ground truth, $f_{\text{full}}(s)$. We then obtain a fixed number $K_3 > 0$ such that

$$\|z - \hat{z}^{(j)}\| < \varepsilon_j \Rightarrow \|g(z) - g(\hat{z}^{(j)})\| < K_3 \varepsilon_j \tag{35}$$

holds, since the common decider function g is also a Lipschitz function as a feed forward neural network [18]. Note that we implemented the g using a feed forward neural network. Using Eq. (31) and (35), we then obtain that the rewards achieved by $g \circ f_j$ are bounded as

$$\begin{aligned}&\|g(z) - g(\hat{z}^{(j)})\| < K_3 \varepsilon_j \\ &\qquad \Rightarrow |R(s_t, g(z)) - R(s_t, g(\hat{z}^{(j)}))| \leq K_2 K_3 \varepsilon_j.\end{aligned} \tag{36}$$

By integrating both sides of the second inequality in Eq. (36), we obtain the below,

$$\begin{aligned}&\int |R(s_t, g(z)) - R(s_t, g(\hat{z}^{(j)}))| d\lambda \leq \int K_2 K_3 \varepsilon_j d\lambda \\ &\qquad \Rightarrow \|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\pi_j}\|_1 \leq \mathbb{E}\left[K_2 K_3 \varepsilon_j\right]\end{aligned} \tag{37}$$

where $\lambda = |\mathcal{P}_{\pi^*_{nm}} - \mathcal{P}_{\pi_j}|$ is a measure defined on the set of events in $\mathcal{S}$. Let $K = K_2K_3$. Since K is a constant, we obtain

$$\|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\pi_j}\|_1 \leq K\mathbb{E}\left[\varepsilon_j\right] = K\mathbb{E}\left[L(f_j, \omega))\right] \quad (38)$$

that satisfies the error bound of Eq. (33).

Second, we show that the bound of Eq. (38) (or Eq. (33)) obtained above drives Eq. (32), if combined with the well-trained orchestrator μ upon all $\pi_j \in \Pi$. For the well-trained orchestrator μ that maximizes the objective in Eq. (22), it is obvious that

$$J(\mu) \geq \max_{j=1,\ldots,m} J(\pi_j). \quad (39)$$

By exploiting the below relation based on Eq. (30),

$$\|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\pi_i}\| \leq \|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\pi_j}\| \Leftrightarrow J(\pi_i) \geq J(\pi_j), \quad (40)$$

we can obtain that

$$\begin{aligned} J(\mu) \geq \max_{j=1,\ldots,m} J(\pi_j) \\ \Rightarrow \|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\mu}\|_1 \leq \min_{j=1,\ldots,m} \|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\pi_j}\|_1 \end{aligned} \quad (41)$$

from Eq. (39). Using Eq. (41) and Eq. (38), we finally obtain the below same to Eq. (32).

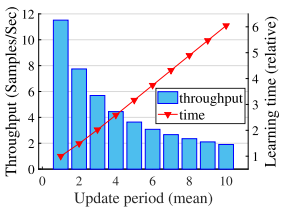
$$\begin{aligned} \|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\mu}\|_1 &\leq \min_{j=1,\ldots,m} \|\mathcal{R}_{\pi^*_{nm}} - \mathcal{R}_{\pi_j}\|_1 \\ &\leq K \min_{j=1,\ldots,\text{full}} \mathbb{E}[L(f_j, \omega)]. \end{aligned} \quad (42)$$

IV. MODEL-BASED ORCHESTRATOR

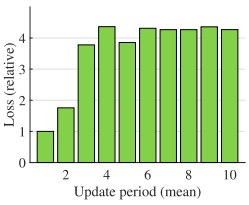
In this section, we describe our data-efficient training method for a high-level orchestrator.

A. DIFFICULTY IN TRAINING ORCHESTRATOR

In rocorl, an orchestrator makes decisions less frequently than control policies, which makes it hard to collect sufficient samples for training. Figure 5(a) depicts sample throughput (on the Y-axis) with respect to various time periods in timesteps for a single round (on the X-axis), when the orchestrator is trained with the Airsim simulator [8]. As the average time period increases, the sample throughput decreases; e.g., a 10 fold increase period results in 83.5% reduction in throughput, increasing the training hours 6 times. It is because the longer each round period, the fewer sampling points within a given training time.



(a) Sample throughput of model-free learning



(b) Loss of model-environment learning

FIGURE 5. Difficulty in training a high-level orchestrator.

Model-based RL methods are effective for solving real-world decision problems because of data-efficient learning capability [19]. However, in applying model-based RL methods to orchestrator training, we need to consider the uncertainty of a learned model-environment. In general, a learned model-environment often becomes inaccurate due to the high complexity of its target environment or overfitting, and this raises the problem of policies not being optimized by interaction with the model-environment [20], [21].

In our case, the orchestrator performs actions at variable time periods that are randomly determined based on network conditions. While each orchestrator action takes place, a control policy performs a relatively large number of actions, which can have a significant impact on environment dynamics. This hierarchical structure increases the uncertainty in predicting the next state for the orchestrator, hence reducing the accuracy of a learned model-environment. Figure 5(b) indicates such limitation where learning loss (on the Y-axis) increases with longer periods of each round (on the X-axis), where learning loss represents the model-environment accuracy measured by L_2 one-step predictions. Several methods are used to learn a model-environment, e.g., splitting a validation dataset for early stopping, input/output data normalization, batch normalization, L_2 regularizer [22], and model-ensemble techniques [21].

B. AGGREGATED POLICY LEARNING

To mitigate the issue of low accuracy of a learned model-environment, we propose the aggregate policy learning scheme by which the model parameters of the orchestrator are updated through interacting with not only a model-environment but also a target environment in parallel.

In general, a model-based learning process consists of transition sampling, model-environment learning, and policy improvement. First, a fixed policy is exploited to collect experienced transitions $D = \{(s_t, \pi_t, R^{\mu}(s_t, \pi_t), s_{t+k+1}) \,|I : [t, t+k]\}$ through interacting with its target environments for update periods I. Then, a model-environment is learned to approximate a transition probability $\mathcal{P}^{\mu}(s, \pi, s')$ and reward function $R^{\mu}(s, \pi)$ via a dynamics function $\mathcal{P}_{\psi}$ and a reward function R_{ϕ}. We see the transition probability as a distribution of random variable s', given s and π. Notice that learning $\mathcal{P}_{\psi}$ and R_{ϕ} is conducted in a supervised manner with the experienced transitions D. The objective functions are defined to minimize the L_2 one-step prediction losses: $\min_{\psi} \frac{1}{|D|} \sum_D \|s_{t+k+1} - \mathcal{P}_{\psi}(s_t, \pi_t)\|_2$ and $\min_{\phi} \frac{1}{|D|} \sum_D \|R^{\mu}(s_t, \pi_t) - R_{\phi}(s_t, \pi_t)\|_2$ respectively. Then, a learned model-environment with $\mathcal{P}_{\psi}$ and R_{ϕ} can be used for policy improvement.

Figure 6 depicts the aggregated policy learning scheme in which model-free learning with a target environment is performed in parallel with model-based learning. In model-free learning, accurate trajectories from target environments are used for policy improvement, thus the sample throughput is low. On the other hand, in model-based

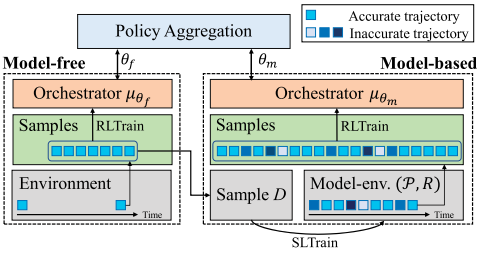


FIGURE 6. Training orchestrator by aggregated policy learning.

learning, more trajectories including inaccurate ones from model-environments are used, thus the sample throughput is high.

As such, to alleviate the low accuracy issue of policy improvement, the two policies updated by model-free and model-based learning are aggregated. Specifically, for policy aggregation, we use the weighted model averaging algorithm in federated learning techniques [23],

$$\theta_f, \theta_m \leftarrow (1-\alpha)\theta_f + \alpha\theta_m \tag{43}$$

where θ_f and θ_m represent the respective model parameters of learned policies by model-free and model-based learning, and $\alpha \in (0, 1)$ denotes an aggregation weight.

V. EVALUATION

In this section, we describe the implementation of **rocorl**, and evaluate it with various simulation conditions. We also demonstrate our case study, automated drone navigation in the Airsim simulator, where a flying RL agent is presumed to run on a drone and to rely on observations with near-edge global and near-device local sensor data.

Our model implementation is based on Python v3.7 and Tensorflow v1.14, and neural networks for policies are trained on a system of an Intel(R) Core(TM) i7-7700 processor and an NVIDIA RTX 2080 GPU. The orchestrator and control policies share common hyperparameter settings for their neural network structure, except that each control policy has its own observation space, as explained in Section III-B. For training policies, we use the PPO and SAC algorithms [24], [25], and the Adam optimizer [26]. The hyperparameter settings are summarized in Table 1. For comparison purposes, we also implement and test several algorithms in addition to the **rocorl** model.

- `REC_FU` and `REC_PA` are recurrent models with different observation spaces, where the former manages a full observation space with stale data, and the latter manages a partial observation space without stale data.
- `DVRL` is a state-of-the-art POMDP method [27]. Different from recurrent models that has no direct influence on belief states, it exploits the evidence lower bound (ELBO) loss to directly affect the belief state inference during learning.

In evaluating models upon stale observations, we concentrate on model robustness, and accordingly, we measure the ratio of the model performance to an ideal reference. That is, for a model F, the performance degradation ratio (Perf. ratio) is calculated as

$$\mathop{\mathbb{E}}_{M^{st}}[J(F)] \;/\; \mathop{\mathbb{E}}_{M^{nm}}\left[J(\pi^*_{nm})\right] \tag{44}$$

where M^{st} and M^{nm} are the stale and normal environments defined in Section III-C and J is defined in Eq. (6). In our tests, the reference performance is empirically calculated based on the accumulated reward obtained in normal environments through the properly learned policy π^*_{nm}. We conduct $10K$ trials for each test case.

TABLE 1. Hyperparameter settings for policy networks.

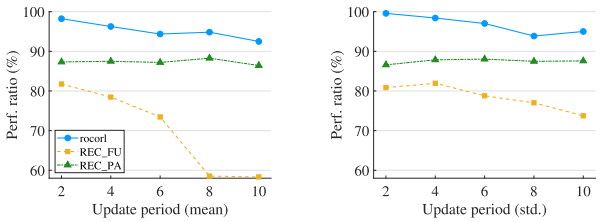
Hyper Parameter	Value
Learning rate	1×10^{-5}
Rollout steps	256
Entropy coefficient	1×10^{-3}
Value coefficient	0.5
Mini-batch size	4
Surrogate optimization interval	4
Clip range	0.5
Max gradient norm	0.5
Generalized advantage estimator	0.95
Batch size	256×32
Layer size (Common)	64×64
Layer size (Value)	64
Layer size (Policy)	64
Activation	x^+(relu)
Num. of samples	1.5×10^8
Aggregation weight (α)	0.05

A. SIMULATION TESTS

For various simulation tests, we implement a moving object environment using pyBox2D [28]. Similar to self-driving scenarios, an agent is set to avoid moving objects and go to the goal location. The agent receives the lidar-like yet simple sensor data as local data, while it receives dynamic map information including the position and speed of long-range obstacles as global data. Regarding actions and rewards, the agent manipulates its steering and velocity, acquiring a binary reward; 1 if the agent achieves its goal, 0 otherwise. The simulation environment settings are summarized in Table 2.

TABLE 2. Simulation environment settings: in the *type* column, *e* and *d* denote the global and local data type respectively, while *c* denotes non-temporal data that can be commonly accessed from both device node and edge service with no stale observation.

Env. setting	Value	Description	Type
Position	$[0, 1]^2$,	device position	d
Sensor	$[0, 1]^{16}$,	lidar like info.	d
Obstacle pos.	$[0, 1]^{2\times 9}$	position of obs. (x,y)	e
Obstacle spd.	$U[0.5, 1 \sim 9]$	speed of obstacles	e
Goal	$[0, 1]^2$	goal location	c
Action	$[-1, 1]^2$	next velocity (v_x,v_y)	-
Reward	$\{0, 1\}$	1 for goal achieve.	-
Period distr.	lognormal	update period distr.	-



(a) Mean (b) Deviation

FIGURE 7. Various settings on data update periods.

Figure 7 shows that **rocorl** outperforms the other models, where we measure model performance in Eq. (44) with respect to various settings on data update periods. **rocorl** shows that it maintains stable performance regardless of (a) mean and (b) deviation values on data update periods in timesteps, in contrast to other models. The larger mean or deviation, the more degraded the performance of `REC_FU`. On the other hand, `REC_PA` is not significantly affected by the update periods since stale data are all removed from its observations, but it has a limitation of relatively low performance.

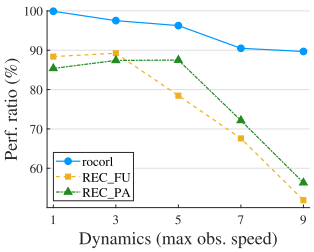


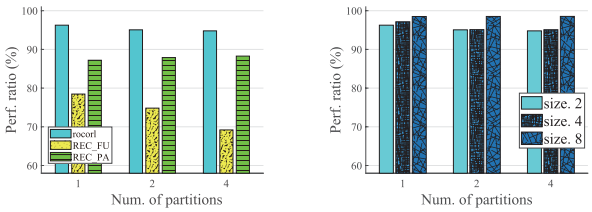
FIGURE 8. Environment dynamics: the environment dynamics is set by obstacle speed ranges.

In Figure 8, we also evaluate the models with respect to various dynamics levels of target environments. We observe that high dynamics settings with rapid moving obstacles increase the difficulty of goal tasks and thus affect the performance of all the models in comparison. Yet, the performance degradation of **rocorl** is not relatively significant, indicating its robustness. In a highly dynamic environment, it is important to make good use of available observations including stale global data, while doing so might have an adverse effect of increasing the likelihood of misusing them. **rocorl** selectively uses proper control policies over time to effectively reduce the likelihood of such misuse through the orchestrator.

In Figure 9(a), we evaluate the models with respect to various staleness. Here, the number of partitions (on the X-axis) represents the number of sensor data sets with different update times; e.g., two individual global data sets differently updated make three partitions including one for device data. Accordingly, more partitions normally increase the variety of stale observations in most cases, tending to make hard to learn stable rules upon stale observations. While the default setting of our target environments sets to have two partitions, one from the device and the other from the edge service, we also consider a situation in which global data from multiple edge services (or nodes) are asynchronously updated to the device. **rocorl** shows more robust performance than the others against various staleness, e.g., 17∼25% higher than `REC_FU`.

While the performance of `REC_FU` degrades with more partitions, that of `REC_PA` and **rocorl** does not. Moreover, **rocorl** outperforms `REC_PA` by 6∼9%. As the number of partitions increases, the variety of stale observations increases, which in turn renders a more difficult environment and more unstable observation probability for `REC_FU` to learn the optimal policy.

In Figure 9(b), we evaluate the relationship of the number of policies with different observation spaces and the number of partitions in **rocorl**. The large policy set (i.e., the size = 8) shows slightly more robust performance than the other (the size = 2). Note that the large set is configured to contain all the policies of the small set in this test for direct comparison. The result clearly shows the benefit of **rocorl** that makes use of the diverse capability of transferable control policies upon variable staleness. It is consistent with Eq. (32).



(a) Various staleness (b) Size of policy set Π

FIGURE 9. Policy sets upon various staleness.

B. CASE STUDY

In the following, we describe our case study for autonomous quad-copter controls with the Airsim simulator [8]. Figure 10 illustrates the system implementation where **rocorl** works as a flying agent for a self-operation drone. We set device-attached sensor measurements such as device position, velocity, acceleration, orientation as well as lidar measurements as near-device local data. We set long-range dynamic map information such as the trajectories of moving

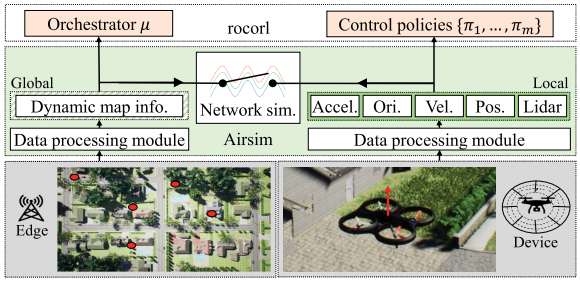


FIGURE 10. Case study system architecture with Airsim.

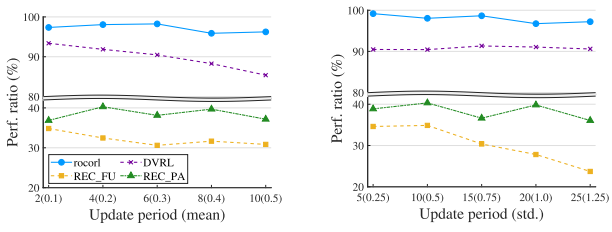
TABLE 3. Case study environment settings: the *type* specifies different data types in the same as in Table 2.

Env. setting	Value	Description	Type
Position	$\mathbb{R}^2 \times [0, 25]$	drone position	d
Velocity	$\mathbb{R}^3$	drone velocity	d
Acceleration	$\mathbb{R}^3$	drone acceleration	d
Orientation	$[-180°, 180°]^3$	drone orientation	d
Lidar	$[0, 200]^{20}$,	lidar info.	d
Obstacle pos.	$\mathbb{R}^2 \times [0, 25]$	pos. of obs. (x,y,z)	e
Obstacle spd.	$U[1, 2 \sim 32]$	speed of obstacles	e
Goal	$\mathbb{R}^2$	goal location	c
Time/step	0.05sec/step	timestep period	-
Action	$[-10, 10]^3$	accel. (a_x,a_y,a_z)	-
Reward	$\mathbb{R}$	based on goal distance	-

objects as near-edge global data. This configuration is based on one of the edge computing scenarios [29], [30]. Furthermore, the dynamic map information is updated to the device, similarly to the learning map scenario [31], where update periods are randomly given.

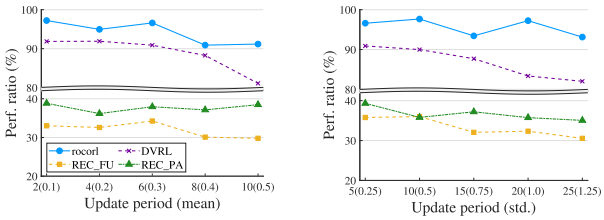
Regarding actions and rewards, we implement an agent that continuously manipulates the 3-D acceleration of a drone, and acquires rewards according to the distance from the goal, i.e., $R(s_t, a_t) = \|go - po_t\|_2 - \|go - po_{t+1}\|_2$ where po_t is the drone position at time t and go is the goal location. The case study environment settings are summarized in Table 3, while other unspecified settings are the same as those for the previous simulation experiments.

Here we evaluate model performance with respect to various settings on update periods. We set random periods with various mean and deviation values (on the X-axis) for successive near-edge global data updates from the edge service to the drone. In Figure 11, we assume that those values are independent from other environment conditions, while in Figure 12, we configure a practical correlation pattern between update periods and environment density. Timely updates to the drone can be more restricted due to possible interference in a harsh area where many obstacles exist. As shown, **rocorl** outperforms the other models, e.g., 60∼71% enhancement over the recurrent `REC_FU` and `REC_PA` models as well as 2∼11% enhancement over the state-of-the-art `DVRL` for all test cases.



(a) Mean (b) Deviation

FIGURE 11. Performance w.r.t. uncorrelated update periods: the X-axis denotes update periods in timesteps (and in seconds).



(a) Mean (b) Deviation

FIGURE 12. Performance w.r.t. correlated update periods: the X-axis denotes update periods in timesteps (and in seconds).

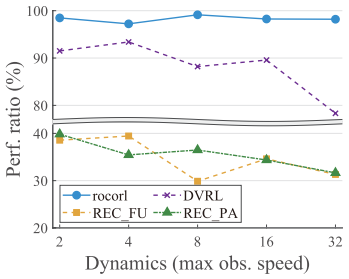


FIGURE 13. Performance with respect to environment dynamics.

In Figure 13, we evaluate model performance with respect to dynamic conditions of a flying environment where the dynamics level is configured by speed distributions of moving obstacles. **rocorl** shows highly stable performance for all cases, while the others show degraded performance when they are affected by severe changes; e.g., `DVRL` is degraded by 15% with an 8 times increase of maximum obstacle speed.

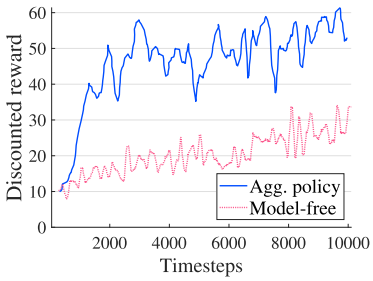
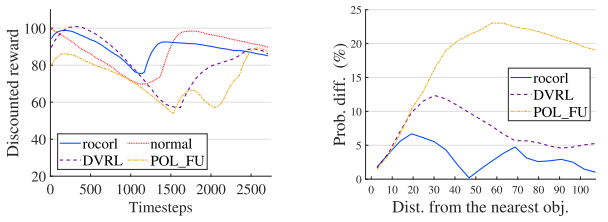


FIGURE 14. Data efficiency of rocorl.

In Figure 14, we evaluate the data-efficiency of **rocorl**. The aggregated policy learning of **rocorl** shows its learning curve with rapid increases of acquired discounted rewards over timesteps, in comparison with the other case where only model-free learning is used, indicating 2∼3 times improved training capability than the other case in terms of data-efficiency.

In Figure 15(a), we compare the discounted reward patterns achieved over time by different models. Here, in addition to `DVRL` and **rocorl**, we include the well-learned reference model running in the normal environment (denoted as "normal" in the figure) and one of the control policies in **rocorl** (denoted as "`POL_FU`") in this comparison. The



(a) Achieved reward patterns (b) Obstacle distance patterns

FIGURE 15. Transferable policies of rocorl.

POL_FU policy takes full observation data as input. Except for the normal model, these models run in the target environment. As noticed, the normal model and **rocorl** show more similar patterns than others. This implies that we have transferable control polices in **rocorl**, which can minimize the state and reward distribution differences between the normal and target environments, thus allowing Eq. (30) to be established.

Furthermore, in Figure 15(b), we compare the obstacle distance patterns obtained by different models. Given the probability distribution of moving obstacle distance by the normal model, we represent its difference from another model (i.e., $\|\mathcal{P}_{\pi^*_{nm}} - \mathcal{P}_{\mu}\|$ in Eq. (30)). Similar to the reward patterns, we notice that the difference between the normal model and **rocorl** is much smaller than that between the normal model and DVRL or POL_FU. It indicates that **rocorl** running in the target environment learns how to control upon stale observations, as if it were running in the normal environment. This policy transferability is enabled by hierarchical learning with the autoencoder-based observation transfer scheme in **rocorl**.

VI. RELATED WORKS

The recent advance of reinforcement learning (RL) has yielded its broad adoption in the area of autonomous cyber-physical systems that operate through interacting with surrounding environments, e.g., autonomous driving [1], [2], robots [3], [32], [33], drones [4], [34], mobile edge computing [30], and others. Such an RL-based system senses environment states, aiming to make timely observations and conduct proper reactions. Thus, the system performance normally becomes dependent on the quality of observations to some extent.

For a linear dynamic system with Gaussian noise observations, the Kalman filter has been adopted to estimate the ground truth of underlying system states [35], [36]. While Gaussian noise observations normally form a POMDP (partially observable MDP) problem, stale observation problems cannot be necessarily formulated with Gaussian noise observations. In the RL research community, many works were investigated to address the issues of partial or noisy observations which are formulated as a POMDP. In [10], Wierestra *et al.* demonstrated that recurrent networks with policy gradient algorithms can be effective for POMDPs. Similarly, in [9], Hausknecht *et al.* adopted the recurrent neural network with the Q-learning algorithm (DQN). Recently, the multi-agent RL system with noisy private observations and selective communication between agents was investigated [37]. Moreover, DVRL [27] and PILCO [20], [38] were proposed. DVRL exploits the evidence lower bound (ELBO) loss, leading to direct estimation on belief states for recurrent RL policies, while PILCO employs dynamic models learned upon belief states directly than observations. These previous works commonly concentrated on partial observations, yet none of them addressed the stale observation problem.

Since POMDP methods were generally designed under the assumption of a fixed correlation between ground truth states and observations (i.e., $\mathcal{O}(s_t) = \Pr(\omega_t|s_t)$), it is difficult to apply them for an environment with stale observations in which gradients of neural networks cannot be given accurately. To the best of our knowledge, our work is the first to formulate the problem of stale observations in the RL context and to propose the solution employing the transfer learning via a hierarchical model.

The option framework was first proposed in [13], and was much studied recently, e.g., the end to end option learning with deep RL [39], transfer learning with options [40]. Our work is in the same vein of these option-based works for handling different levels of temporal abstraction and solving a complex decision making problem. However, none of the previous works considered limited data updates or stale observations.

Meanwhile, Gupta *et al.* [15] presented a knowledge transfer theory for RL agents in the form of statistical distance. They focused on finding out good feature extraction functions that can reduce the morphological difference between two different environments. In our problem, it is difficult to directly extract features from stale environments. Thus, we exploit observation transfer functions in an option-like hierarchical learning structure, rather than extracting features directly from stale environments.

There has been a body of research on model-environment representation in the field of robotics [32], [33]. Recently, several studies have been introduced for adopting deep neural networks in model-based RL and handling complex environments [21], [41], [42]. In [21], the model ensemble technique based on the trust-region policy optimization was introduced to tackle the shortcomings of backpropagation through time. In [41], asynchronous process structures were investigated to alleviate the model bias problem. Using the techniques of [21] and [41], we employ an integrated learning approach that combines both model-free and model-based learning to solve the model-bias issue caused by hierarchical RL structures.

Our system design is based on edge-device data synchronization and it shares a similar structure with decentralized POMDPs [37], [43], [44]. In [37], MADDPG-M handles a specific constrained case where agents operate under partial observations that are weakly correlated to true states. While MADDPG-M focuses on agents' decisions about what to share with others, our work seeks to intelligently make use of imperfect observations, under a simple but nonrestrictive assumption that data are shared randomly in time.

VII. CONCLUSION

In this paper, we presented **rocorl**, the hierarchical RL-based control model that can effectively deal with temporally outdated observations incurred by intermittent sensor data updates in a cyber-physical environment. For training the model, we employ the autoencoder-based observation transfer and aggregated policy learning schemes. Our approach is based on a set of policy variants with different observation transfers by which the learned knowledge is transferable for environments with stale observations. Through experiments with the Airsim simulator, we show that **rocorl** is robust against various restrictive conditions of sensor data updates, compared with several other models including a state-of-the-art POMDP method.

Our future work is to adapt the **rocorl** model for a large scale edge computing environment where many edge computing nodes communicate and interact.

REFERENCES

[1] A. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electron. Imag.*, vol. 2017, no. 19, pp. 70–76, Jan. 2017.
[2] B. Osinski, A. Jakubowski, P. Miłos, P. Ziecina, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," 2019, *arXiv:1911.12905*. [Online]. Available: http://arxiv.org/abs/1911.12905
[3] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine, "Collective robot reinforcement learning with distributed asynchronous guided policy search," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 79–86.
[4] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 6008–6014.
[5] K.-D. Kang, S. H. Son, J. A. Stankovic, and T. F. Abdelzaher, "A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases," in *Proc. 14th Euromicro Conf. Real-Time Systems. Euromicro RTS*, Jun. 2002, pp. 203–212.
[6] C. Deng, G. Li, Q. Zhou, and J. Li, "Guarantee the quality-of-service of control transactions in real-time database systems," *IEEE Access*, vol. 8, pp. 110511–110522, 2020.
[7] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2731–2735.
[8] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," 2017, *arXiv:1705.05065*. [Online]. Available: http://arxiv.org/abs/1705.05065
[9] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Ser.*, 2015, p. 32.
[10] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, "Solving deep memory POMDPs with recurrent policy gradients," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*. Berlin, Germany: Springer, 2007, pp. 697–706.
[11] R. Fruit and A. Lazaric, "Exploration-exploitation in MDPs with options," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, vol. 54, 2017, pp. 576–584.
[12] B. Ravindran and A. G. Barto, "SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes," in *Proc. 18th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2003, pp. 1011–1016.
[13] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, nos. 1–2, pp. 181–211, Aug. 1999.
[14] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2005, pp. 539–546.
[15] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," 2017, *arXiv:1703.02949*. [Online]. Available: http://arxiv.org/abs/1703.02949
[16] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare, "DeepMDP: Learning continuous latent space models for representation learning," 2019, *arXiv:1906.02736*. [Online]. Available: http://arxiv.org/abs/1906.02736
[17] M. Pirotta, M. Restelli, and L. Bascetta, "Policy gradient in Lipschitz Markov decision processes," *Mach. Learn.*, vol. 100, nos. 2–3, pp. 255–283, Sep. 2015.
[18] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 214–223.
[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.
[20] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 465–472.
[21] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018.
[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 448–456.
[23] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*. [Online]. Available: http://arxiv.org/abs/1610.05492
[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347
[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 1861–1870.
[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.
[27] M. Igl, L. Zintgraf, T. Anh Le, F. Wood, and S. Whiteson, "Deep variational reinforcement learning for POMDPs," 2018, *arXiv:1806.02426*. [Online]. Available: http://arxiv.org/abs/1806.02426
[28] Pybox2d. (2020). *Pybox2d*. [Online]. Available: https://github.com/pybox2d/pybox2d
[29] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2018, pp. 159–173.
[30] L. Xiao, Y. Ding, D. Jiang, J. Huang, D. Wang, J. Li, and H. V. Poor, "A reinforcement learning and blockchain-based trust mechanism for edge networks," *IEEE Trans. Commun.*, vol. 68, no. 9, pp. 5460–5470, Sep. 2020.
[31] P. Mirowski, M. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, A. Zisserman, and R. Hadsell, "Learning to navigate in cities without a map," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 2419–2430.
[32] J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 2, May 2001, pp. 1615–1620.
[33] V. Kumar, E. Todorov, and S. Levine, "Optimal control with learned local models: Application to dexterous manipulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 378–383.
[34] H. Lu, Y. Li, S. Mu, D. Wang, H. Kim, and S. Serikawa, "Motor anomaly detection for unmanned aerial vehicles using reinforcement learning," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2315–2322, Aug. 2018.
[35] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.
[36] G. Welch and G. Bishop, "An introduction to the Kalman filter," Dept. Comput. Sci., Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, Tech. Rep. TR 95-041, 1995.
[37] O. Kilinc and G. Montana, "Multi-agent deep reinforcement learning with extremely noisy observations," 2018, *arXiv:1812.00922*. [Online]. Available: http://arxiv.org/abs/1812.00922
[38] R. McAllister and C. E. Rasmussen, "Data-efficient reinforcement learning in continuous state-action Gaussian-POMDPs," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 2040–2049.
[39] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–9.

[40] B. Fernandez-Gauna, J. M. Lopez-Guede, and M. Graña, "Transfer learning with partially constrained models: Application to reinforcement learning of linked multicomponent robot system control," *Robot. Auto. Syst.*, vol. 61, no. 7, pp. 694–703, Jul. 2013.
[41] Y. Zhang, I. Clavera, B. Tsai, and P. Abbeel, "Asynchronous methods for model-based reinforcement learning," in *Proc. Conf. Robot Learn. (CORL)*, 2020, pp. 1338–1347.
[42] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, "Model based reinforcement learning for Atari," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, 2020.
[43] F. A. Oliehoek, "Decentralized POMDPs," in *Reinforcement Learning*. Berlin, Germany: Springer, 2012, pp. 471–503.
[44] B. Eker and H. L. Akın, "Using evolution strategies to solve DEC-POMDP problems," *Soft Comput.*, vol. 14, no. 1, pp. 35–47, Jan. 2010.

GWANGPYO YOO received the B.S. degree in mathematics and computer engineering from Sungkyunkwan University, Suwon, in February 2020, where he is currently pursuing the M.S. degree in computer science and engineering. His research interests include real time data orchestration and reinforcement learning.

MINJONG YOO received the B.S. degree in mathematics and computer science and engineering from Sungkyunkwan University, Suwon, in 2020, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research interests include reinforcement learning, deep learning, and networked cyber-physical systems.

IKJUN YEOM received the B.S. degree in electronic engineering from Yonsei University, Seoul, South Korea, in February 1995, and the M.S. and Ph.D. degrees in computer engineering from Texas A&M University, in August 1998 and May 2001, respectively. He worked with Dacom, Inc., from 1995 to 1996, and Nortel Networks Corporation, in 2000. He had been an Associate Professor with the Department of Computer Science, KAIST, from 2002 to 2008. He is currently a Full Professor with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea. His research interests include AQM, congestion control, TCP, wireless networks, and future Internet architecture.

HONGUK WOO (Member, IEEE) received the B.S. degree in computer science from Korea University, Seoul, in 1995, and the M.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin TX, USA, in 2002 and 2008, respectively. From 2008 to 2018, he worked for Samsung Research of Samsung Electronics as a Principal Engineer and the Vice President. Since 2018, he has been an Assistant Professor with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea. His research interests include intelligent application, analytic monitoring, cloud computing, and networked cyberphysical systems.