MARL Sim2real Transfer: Merging Physical Reality With Digital Virtuality in Metaverse

Haoran Shi, Guanjun Liu, *Senior Member, IEEE*, Kaiwen Zhang, Ziyuan Zhou, and Jiacun Wang, *Senior Member, IEEE*

***Abstract*—Metaverse is an artificial virtual world mapped from and interacting with the real world. In metaverse, digital entities coexist with their physical counterparts. Powered by deep learning, metaverse is inevitably becoming more intelligent in the interactions between reality and virtuality. However, it is confronted with a nontrivial problem known as *sim2real transfer* when deep learning techniques try to bridge the *reality gap* between the physical world and simulations. In this article, we use multiagent deep reinforcement learning (MARL) to implement collective intelligence for digital entities as well as their physical counterparts. To model the immersive environments in metaverse, we define a *nonstationary* variant of Markov games and propose a recurrent MARL solution to it. Based on the solution, MARL sim2real transfer that bridges real and virtual multiple unmanned aerial vehicle (multi-UAV) systems is successfully conducted by employing recurrent multiagent deep deterministic policy gradient (R-MADDPG) with the domain randomization technique. Additionally, we use perception-control modularization to improve the generalization performance of MARL policies and make training more efficient.**

***Index Terms*—Metaverse, multiagent deep reinforcement learning (MARL), multiple unmanned aerial vehicle (multi-UAV). nonstationary Markov game, sim2real transfer.**

I. Introduction

METAVERSE is an artificial virtual world mapped from and interacting with the real world. In metaverse, digital virtuality, physical reality, and people are seamlessly integrated. As a result, a concrete metaverse should be regarded as a specific realization of cyber–physical–social systems (CPSSs) [1] where digital entities and their physical counterparts are highly interconnected to provide immersive user experiences.

Manuscript received 11 November 2022; revised 6 December 2022; accepted 10 December 2022. Date of publication 20 December 2022; date of current version 17 March 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62172299 and Grant 62032019; in part by the Shanghai Science and Technology Committee under Grant 22511105500; and in part by the Lab of High Confidence Embedded Software Engineering Technology under Grant LHCE-SET202201. This article was recommended by Associate Editor F. Y. Wang. *(Corresponding author: Guanjun Liu.)*

Haoran Shi, Guanjun Liu, Kaiwen Zhang, and Ziyuan Zhou are with the Department of Computer Science, Tongji University, Shanghai 201804, China (e-mail: 2033006@tongji.edu.cn; liuguanjun@tongji.edu.cn; 2210133@tongji.edu.cn; ziyuanzhou@tongji.edu.cn).

Jiacun Wang is with the Computer Science and Software Engineering Department, Monmouth University, West Long Branch, NJ 07764 USA (e-mail: jwang@monmouth.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TSMC.2022.3229213.

Digital Object Identifier 10.1109/TSMC.2022.3229213

Driven by rapid developments of emerging technologies such as artificial intelligence, extended reality, and blockchain, metaverse is becoming an attainable reality. As a promising artificial intelligence technique, deep reinforcement learning (DRL) recently achieves remarkable success in both video games of virtuality [2], [3] and many real-world scenes, such as robotic manipulation [4], [5], mobile robot control [6], [7], [8], [9], [10], [11], [12], [13], and manufacturing process [14], [15], which makes it ideally suited for the realization of metaverse intelligence. Multiagent DRL (MARL) is a multiagent extension of DRL that concentrates on the relation and interaction of multiple agents in mixed cooperative-competitive environments [16]. As a result, MARL is more capable of implementing collective intelligence for complex CPSS.

Because a simulator is faster, more scalable, and lower cost than a physical platform for data collection, it is a popular choice to train DRL policies in simulations which can learn from millions of samples of data. Furthermore, a DRL training can adopt random exploration which is dangerous for physical hardware. Ideally, we should use DRL to learn intelligent policies encoding complicated decision-making behaviors completely from simulations in the virtual world, and then transfer the results to physical systems in the real world with minimal additional training. Unfortunately, the discrepancy between simulations and the real world, known as the *reality gap*, make it challenging to perform this kind of transfer. The reality gap in robot applications is mainly due to dynamic environments and uncertain physical hardware in the real world, such as object mass and friction that influence dynamics, GPS signal strength and communication conditions that cause positioning error, and lighting conditions and various environment backgrounds that confuse visual perception. Because the real world is dynamic and uncertain, the virtual world that is subject to current digital technologies faces difficulties in reproducing richness and randomness, which results in the reality gap.

Sim2real transfer is a class of methods to bridge the reality gap. The basic idea of sim2real transfer is to connect and integrate digital entities in simulations (e.g., DRL policies) with their physical counterparts in the real world, which is equivalent to how metaverse works. Fig. 1 shows the relation between sim2real transfer and metaverse, where sim2real transfer is a concrete form of the interactions between digital virtuality and physical reality in metaverse. Though sim2real transfer has been explored by many

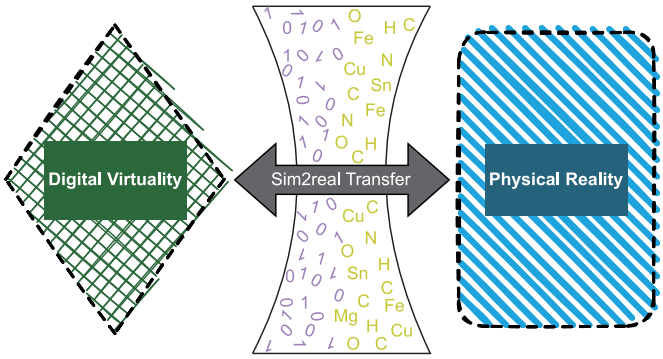


Fig. 1. Sim2real transfer in metaverse.

remarkable works [17], [18], [19], [20], [21], [22], [23], few of them focus on MARL sim2real transfer and its usage in metaverse. The work in [24] is the closest one to ours. Candela et al. [24] used a sim2real transfer technique called *domain randomization* to develop their multiple autonomous vehicles (multi-AVs). In our work, we propose a method to successfully transfer MARL-based multiple unmanned aerial vehicles (multi-UAVs) from simulations to the real world by utilizing domain randomization and perception-control modularization. Our multi-UAV system is to collaboratively deliver a good from a source place to a target place.

The majority of robotic systems are dynamic and uncertain. Hence, we present a *nonstationary* variant of Markov games [25] under the MARL framework to model such a nonstationary system in the real world and metaverse. This article also demonstrates theoretically and empirically that MARL algorithms with recurrent neural networks (recurrent MARL) can solve the *nonstationary* Markov games. From the perspective of MARL sim2real transfer, solving the nonstationary Markov games provides an approach to bridge the reality gap since the physical world can be reconstructed by utilizing domain randomization in simulations.

In order to evaluate the effectiveness of the aforementioned theory and method, we build a simulated scenario of multi-UAV collaboratively delivering goods on a simulator named Airsim [26]. Then, recurrent multiagent deep deterministic policy gradient (R-MADDPG) [27] and other MARL policies are trained with domain randomization in this simulation. Afterwards we conduct sim2real transfer onto our physical autonomous cooperative multi-UAV system to perform a real-world good delivery task. Additionally, a perception-control modularization is used to improve the generalization performance of MARL policies and make training more efficient.

The main contributions of this article are summarized as follows.

1) We define nonstationary Markov games to model MARL problems with dynamic and uncertain systems.
2) We prove that recurrent MARL of centralized training and decentralized execution is an effective solution to the nonstationary Markov games.
3) We propose the methodology of sim2real transfer based on R-MADDPG and domain randomization, and illustrate that the perception-control modularization can accomplish the task of multi-UAV collaboratively delivering goods.

The remainder of this article is organized as follows. Section II introduces related works and compares some of them with ours. In Section III we give the framework of Markov games and introduce multiagent deep deterministic policy gradient (MADDPG). Section IV proposes the theory and methodology of MARL sim2real transfer. Section V presents experiments and results in both simulations and the real world to demonstrate the superiority of the proposed methods. Section VI concludes the work and gives future directions.

II. Related Works

A. Metaverse

The term metaverse was first introduced in a science fiction named *Snow Crash* in 1992 [28]. People can possess their own avatars in a virtual world called metaverse. Nowadays metaverse has been attracting wide attention from both industry and academia. Facebook CEO Mark Zuckerberg renames his company as *Meta* [29] aiming to build the social metaverse to connect each person. In addition, many technology enterprises, including Microsoft, NVIDIA, and Tencent, have participated in the concept of metaverse. As a key technique of metaverse, virtual reality has already been utilized in the serious game to improve educational effectiveness [30], [31], [32], [33], [34]. Furthermore, the work in [35] based its view on metaverse-powered online distance education. In [36], a survey was conducted to discuss how blockchain and artificial intelligence can influence metaverse. A DAO-based decentralized autonomous metaverse (*DeMetaverse*) was proposed in [37]. Wang [1] viewed CPSS as the abstract and scientific name for metaverse and introduced parallel intelligence in metaverse that overcomes Lighthill's gap, where the concept of Lighthill's gap resembles the reality gap between the physical world and the mental world. Consequently, MARL sim2real transfer can be regarded as a specific method to implement parallel intelligence for metaverse.

B. Multiagent Reinforcement Learning for Multientity Decision-Making Tasks

MARL is experiencing an explosive development in recent years. There is a vital paradigm of MARL referred to as centralized training with decentralized execution (CTDE) [38], [39]. Including general algorithms of this paradigm, such as MADDPG [40] and multiagent proximal policy optimization (MAPPO) [41], MARL has been proven to be powerful enough to tackle decision-making tasks for multiple entities (or agents) in various environments.

Vinyals et al. [42] proposed their MARL agent AlphaStar that achieved Grandmaster level in a video game named StarCraft II where every game entity learned to attack enemies and defend themselves. MARL was also employed in [43] to tackle sequential social dilemmas, a kind of multiagent stochastic game with partial information. You et al. [44] proposed their MARL-based packet routing algorithm which

significantly reduced the packet delivery time in general network topologies. In [45], an autonomous driving policy based on an MARL method was introduced for guaranteeing functional safety of the driving policy outcome.

Due to the fact that UAVs are usually deployed to carry out cooperative tasks in a decentralized fashion, multi-UAV systems are naturally suited to MARL of CTDE. As a consequence, many autonomous multi-UAV applications are studied based on MARL. Hu et al. [46] proposed a compound-action actor-critic (CA2C) algorithm to solve cooperative sensing and transmission tasks of UAVs. In [47], a distributed MARL algorithm was applied to task allocation of spectrum sharing among UAVs. The work reported in [48] used an MADDPG framework for the UAV-enabled secure communication with cooperative jamming. Cui et al. [49] developed an MARL framework to deal with dynamic resource allocation of multiple UAVs-enabled communication networks. In [50], a simultaneous target assignment and path planning (STAPP) algorithm was introduced on the basis of MADDPG.

It must be noted that none of the aforementioned works focused on sim2real transfer, and all of them conducted experiments only in simulations. Our work successfully applies MARL sim2real transfer to accomplish the task of multi-UAV collaboratively delivering goods. The MARL algorithm we use is R-MADDPG [27] which is a memory-based version of MADDPG [40].

C. Sim2real Transfer

Due to high sample complexity and safety issues, training MARL policies in the real world is challenging. Simulations overcome these challenges and serve as a testbed to perform experiments on algorithms. However, the use of simulations brings a new problem named *sim2real transfer* because of differences between simulations and the real world in dynamics, sensors, imagery, etc.

There are sim2real transfer techniques for robot applications that customize simulation engines [22], [23] or perform system identification [19], [21] to enhance the accuracy of simulators and to narrow the reality gap. However, these methods can work only when the real-world system can be modeled to be static and constant for simulations, which is hardly achievable for complicated systems due to the lack of observation samples, estimation errors, and so on. In contrast, the idea of domain randomization is to provide enough uncertainties for simulated systems, so that the policies trained to fit in the randomness can adapt to real-world systems. Through domain randomization, differences between the source domain and the target domain are modeled as randomness in the source domain. The randomness can cover all *domain parameters* that keep fixed during a certain period in physical systems, including vision parameters (e.g., lighting conditions, viewpoints, object appearance, and backgrounds) as well as robotic dynamics parameters (e.g., mass, friction, damping, and delay).

The work in [17] is the first one to apply domain randomization to visual data obtained from camera images to complete object localization tasks for the purpose of robotic control. Peng et al. [18] randomized the dynamics of the simulator to train and transfer DRL policies to perform an object-pushing task using a robotic arm. Similarly, randomly sampled parameters were used in [19] in order to apply the controllers learned with a DRL algorithm from simulations to real-legged robots. In the context of UAV applications, Loquercio et al. [20] also benefited from domain randomization to generate simulated visual data for their vision-based drone racing task. However, they only considered single-UAV situations and used domain randomization for robot perception, which makes their work quite different from ours.

As for MARL sim2real transfer, Zhang et al. [51] applied a cognitive consistency-based MARL method to the circle formation control problem for fish-like robots. Although they also performed sim2real transfer by using domain randomization for dynamics and observation, their work did not concentrate on how to improve the robustness of their MARL algorithm for sim2real transfer. In [52], an adversarial MARL algorithm was proposed to train robust policies for better sim2real transfer, but their approach only applies to a two-agent competitive configuration, which limits its usage. The study that most closely resembles ours is [24], where a multiple autonomous vehicles (multi-AVs) system was implemented. They trained multiagent policies using MAPPO [41] to control vehicles in a simulated environment of the Duckietown multirobot testbed with different levels of domain randomization, and then transferred the trained policies to the real environment. Their work used memoryless feedforward neural networks (FNNs) to model policies. Although their memoryless policies achieved better performance than a rule-based one, the lack of mechanisms to capture history information prevents their policies from inferring the domain parameters of environments. We think the ability to infer domain parameters is essential for bridging the reality gap when utilizing domain randomization to simulate the dynamic and uncertainty of the real world. By comparison, we show that our memory-based recurrent MARL method is far more efficient, robust, and general for sim2real transfer.

III. Background

In this section, we present the MARL framework and introduce notations used in the following sections.

A. Markov Games

A multiagent extension of Markov decision processes (MDPs) called partially observable Markov games [25] can be defined by $\langle N, S, A_i, T, O_i, r, \gamma \rangle$. The possible configurations of N agents are described by a set of states S with a set of actions $A_1, \ldots, A_N$ and a set of observations $O_1, \ldots, O_N$ for each agent. At each discrete timestep t, each agent i ($i \in N$) obtains a private observation correlated with the current state

$$o_i^t : S \mapsto O_i$$

and uses a policy

$$\boldsymbol{\pi}_i : O_i \mapsto A_i$$

to take an action a_i^t, which leads to the next state according to the state transition function

$$T : S \times A_1 \times \ldots \times A_N \mapsto S.$$

Under cooperative settings, all agents then share the same scalar reward as a function of the last state and actions

$$r^t : S \times A_1 \times \ldots \times A_N \mapsto \mathbb{R}$$

and receive new observations. Each agent i aims to maximize its own total expected return

$$R_i = \mathbb{E}\left[\sum_t \gamma^t r^t\right]$$

where $\gamma \in [0, 1]$ is a discount factor.

B. Inherent Nonstationarity

Since we are discussing the nonstationary problem arising from dynamic and uncertain systems, it is worth pointing out that there is a classical problem of MARL known as the *inherent nonstationarity* caused by the influence of other behavior changing agents from the perspective of any individual agent at training time.

CTDE paradigm [38], [39] can be utilized to solve the inherent nonstationarity by making the most of the advantages of centralized training.

C. Multiagent Deep Deterministic Policy Gradient

MADDPG [40] is a powerful actor-critic policy gradient algorithm to solve MARL problems. Each agent i has its own policy $\boldsymbol{\pi}_i(o_i) = a_i$ which is referred to as the actor, and an action-value function for policy $\boldsymbol{\pi}_i$ which is the critic denoted by $Q_i^{\boldsymbol{\pi}}(x, c)$, where $x = (o_1, \ldots, o_N)$ and $c = (a_1, \ldots, a_N)$ when no additional information is provided. MADDPG uses these centralized critics that take into account extra information about all other agents to deal with the inherent nonstationarity, while making decentralized actors considered to remain in single-agent situations. Moreover, the framework of CTDE allows the critic to use sufficient additional information provided by the simulator at training time to reduce the variance of policy gradients and improve policy performance, while the actor (policy $\boldsymbol{\pi}$) remains restricted to its private observation since the additional information is not available for the policy at execution time. In this case, introducing additional information $z \in Z$ correlated with S, the critic is denoted by $Q_i^{\boldsymbol{\pi}}(x, c, z)$.

IV. Theory and Methodology

A. Nonstationary Markov Games

As a *nonstationary* variant of partially observable Markov games, *nonstationary* Markov games are defined by $\langle N, S, A_i, T, O_i, r, \gamma, \mu\rangle$, where μ is a set of domain parameters that parameterize the system with respect to agents' own properties and the environment. Here, the next state is achieved following the nonstationary state transition function

$$T : S \times A_1 \times \ldots \times A_N \times \mu \mapsto S.$$

General Markov games think the system with respect to agents' own properties and the environment is fixed. In contrast, considering that most real-world systems are dynamic and uncertain, we propose the nonstationary Markov games where the state transition function is affected by a set of variables of domain parameters μ and thus becomes nonstationary.

B. Recurrent MARL Solution to Nonstationary Markov Games

Assumption 1: Under the paradigm of CTDE, from the perspective of any decentralized agent i, we ignore the influence of other agents and consider other agents to be a part of the environment.

Assumption 2: With Assumption 1, we think that $s : O_i \mapsto S$ is true, which means $o_i : S \mapsto O_i$ is bijective.

Proposition 1: Given an MARL algorithm that satisfies the CTDE paradigm, if it is realized with recurrent neural networks (RNNs), i.e., the RNN model $\Pi_i(:, g_i^t)$ corresponds to the decentralized policy for agent i, where

$$g_i^t = g(\tau_i^t)$$

is the internal memory

$$\tau_i^t = \left[o_i^t, a_i^{t-1}, o_i^{t-1}, a_i^{t-2}, o_i^{t-2}, \ldots\right]$$

and : represents other inputs, then it can be a solution to nonstationary Markov games.

Proof: Based on Assumptions 1 and 2 and the nonstationary state transition function

$$T : S \times A_1 \times \ \ldots \times A_N \times \mu \mapsto S$$

we introduce the nonstationary observation transition function

$$P_{i,\mu}\left(o_i^{t+1} \middle| a_i^t, o_i^t; \mu\right) : O_i \times A_i \times \mu \times O_i \mapsto [0, 1]$$

(i is ignored in the following proof).

Let

$$H^t = \left[O^t, A^{t-1}, O^{t-1}, A^{t-2}, O^{t-2}, \ldots\right].$$

Then, we have

$$H^t \Leftrightarrow \left\{\left(O^{t-j} \middle| A^{t-j-1}, O^{t-j-1}\right) \middle| 0 \le j < t\right\}$$

and

$$\exists 0 \le j < t, \left(O^{t-j} \middle| A^{t-j-1}, O^{t-j-1}\right) \sim P_\mu.$$

According to the Glivenko–Cantelli theorem, with a large number of samples

$$\tau_i^t = \left[o_i^t, a_i^{t-1}, o_i^{t-1}, a_i^{t-2}, o_i^{t-2}, \ldots\right]$$

the true distribution function P_μ can be inferred, and so can μ which parameterizes the nonstationary problem.

Therefore, a recurrent MARL algorithm with decentralized policy $\Pi_i(:, g_i^t)$ that is able to infer P_μ and μ by learning from τ_i^t is a solution to nonstationary Markov games. ■

Assumption 1 is premised on the fact that the centralized part solves the *inherent nonstationarity* of MARL, and

therefore, the decentralized policy can be regarded to be reducible to single-agent situations. Assumption 2 is a common assumption under single agent situations where the environment is fully observed [53]. The partially observable problem is mainly considered by MARL and it is reasonable for the decentralized policy to have Assumption 2 under Assumption 1.

With the goal to maximize the total expected return R, the recurrent policy is sure to learn to infer P_μ and μ somehow so as to gain more information about the system and take better actions, which leads to more positive rewards. Eventually, it will converge to more effective solutions.

Under the setting of deterministic policies, inputs of the internal memory of the decentralized policy $\Pi_i(:, g_i^t)$ can be simplified as follows:

$$g_i^t = g\left(h_i^t\right)$$

where

$$h_i^t = \left[o_i^{t-1}, o_i^{t-2}, \ldots\right].$$

If the policy of each agent i is deterministic, denoted by $\boldsymbol{\pi}_i(o_i) = a_i$, the nonstationary observation transition function

$$P_{i,\mu}\left(o_i^{t+1} \middle| a_i^t, o_i^t; \mu\right)$$

can be simplified as follows:

$$P_{i,\mu}\left(o_i^{t+1} \middle| o_i^t; \mu\right)$$

since a_i^t is deterministic if o_i^t is deterministic.

C. R-MADDPG

R-MADDPG [27] is a memory-based extension of MADDPG. A model where both actors and critics are recurrent was verified to be more capable of learning under partially observable environments.

In this article, we propose the memory-based actor model

$$\boldsymbol{\pi}_i\left(o_i^t, y_i^t\right) = a_i^t$$

where

$$h_i^t = \left[o_i^{t-1}, o_i^{t-2}, \ldots\right]$$

and the internal memory

$$y_i^t = y\left(h_i^t\right)$$

acts as the history of past observations obtained by agent i, and the memory-based critic model

$$Q_i^{\boldsymbol{\pi}}\left(x^t, c^t, z^t, \mu, f^t\right)$$

where

$$f^t = f\left(h_1^t, \ldots, h_N^t\right)$$

is the internal memory of the action-value function and (z^t, μ) is the additional information only provided to the critic.

The internal memory enables the policy to infer the domain parameters of environments through the history information, which benefits the policy as the additional information does. Our work will show that R-MADDPG is remarkable when facing dynamic and uncertain systems under both simulations and real-world conditions.

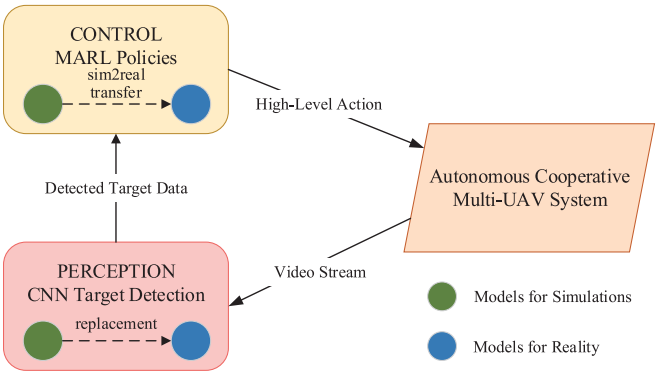


Fig. 2. Perception-control modularization on the autonomous cooperative multi-UAV system.

D. Perception-Control Modularization

Perception-control modularization divides the policies of robot applications into two modules: 1) perception and 2) control. The perception module is designed to detect target objects around the environment, and the control module functions to navigate robots to accomplish the task.

Many DRL-based robot applications are prone to learning end-to-end policies, e.g., from the input of pixel signals to the output of control signals. One advantage of this kind of end-to-end method is that visual data are easy to obtain and are very informative. In our work, we abandon the end-to-end method because it is often plagued by high sample complexity and low generalization. Specifically, the end-to-end method increases the consumption of time and manpower when training DRL policies with pixel data, and makes it difficult to deploy the trained policies to real-world environments due to the large gap between low-quality simulated images and real images.

Instead, we use modularization as an alternative. The idea of modularization is to decompose policies into multiple modules, each of which can be implemented independently and then combined into a complete system. The advantage of modularization is that it dissolves the tight coupling between input and output signals, so that more information can be extracted between different modules.

In this work, we conduct perception-control modularization on our autonomous cooperative multi-UAV system, as shown in Fig. 2. Specifically, the perception module employs a CNN-based target detection algorithm, which inputs pixel data from the video streams of cameras and outputs the result that locates the detected targets in the input streams. This step realizes feature extraction and dimension reduction from high-dimensional pixel data. Taking the detected target data obtained by perception (and other inputs), the control module uses MARL-based policies that output high-level actions to control the UAV. This perception-control modularization is distributively deployed on each UAV of our multi-UAV system.

It should be noted that our work of sim2real transfer is conducted on MARL policies of the control module. Since the perception module applies different but homogeneous CNN models trained by simulated and real-world images, respectively. For the simulated and real-world environments, we label

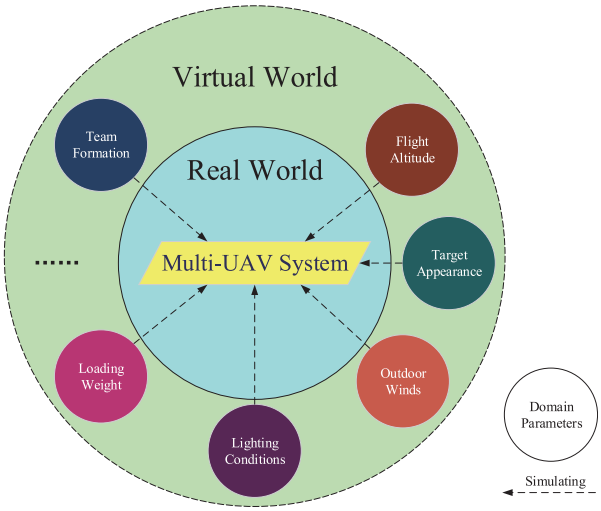


Fig. 3. Domain randomization for the simulation of multi-UAV systems in metaverse. By randomizing a variety of domain parameters corresponding to the factors of the multi-UAV system, domain randomization makes the real-world system a sample of the space of simulated systems in the virtual world of metaverse.

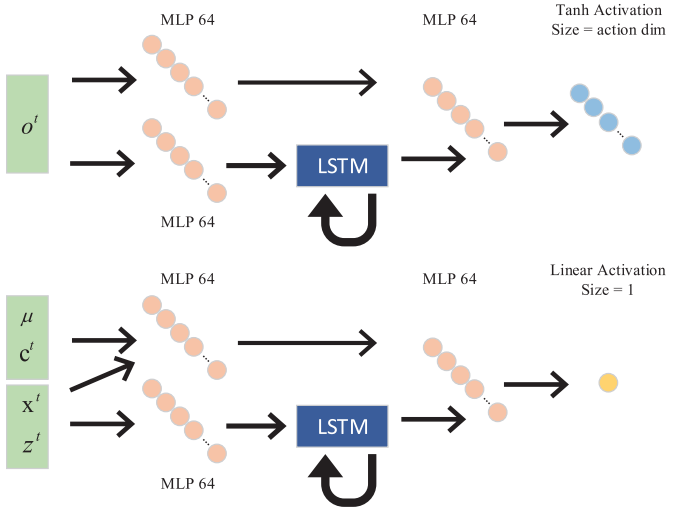


Fig. 4. RNNs of memory-based actor (top) and memory-based critic (bottom).

the transformation of the perception module as the "replacement" of CNN target detection models. Through assigning a part of subtasks to other algorithms that are more expert, perception-control modularization improves the generalization performance of perception and reduces sample complexity for the learning of control.

E. Domain Randomization

In multi-UAV systems, observation noise that arises from various sensors and communication conditions severely affects the robustness of systems. In addition, changing environments and various tasks result in differences in loading weight, team formation, flight altitude, outdoor winds, lighting conditions, target appearance, etc. These differences are abstracted as domain parameters in simulations to reproduce real-world multi-UAV systems.

Fig. 3 illustrates how domain randomization is used to simulate a multi-UAV system. By randomizing a variety of domain parameters, we aim to let real-world systems be samples of the space of simulated systems in every aspect. MARL policies trained in the randomized virtual world can, therefore, be generalized to complicated multi-UAV systems.

F. Network Architecture

The RNNs of actor and critic are shown in Fig. 4. Both networks are composed of a feedforward branch and a recurrent branch. The feedforward branch consists of a fully connected layer, and the recurrent branch contains an embedding fully connected layer followed by an LSTM layer where the internal memory is updated at every timestep during an episode. The outputs of both branches are then concatenated and processed by two fully connected layers. Each fully connected layer has an ReLU activation except the output layer. All hidden layers including LSTM consist of 64 units.

The actor network takes in observation o^t as input and sends its copies to both branches. While the recurrent branch is aimed at extracting features of the environment through the history of observations, the feedforward branch helps to gain direct information of observations which is essential for determining the output action a^t. The critic network is of a similar architecture with the actor network, but their difference lies in their inputs. Copies of (x^t, z^t) are the input of both branches and the feedforward branch additionally takes in (μ, c^t). While the critic network has a linear output unit corresponding to the Q-value, the output of the actor network is activated with tanh and scaled to fit each action variable for different scenarios.

G. MARL Sim2real Transfer

The workflow of MARL sim2real transfer is shown in Fig. 5. Domain parameters need to be determined according to the demands of the real-world task and then the simulated system can be implemented with domain randomization. After training and testing the MARL policy in the simulation, sim2real transfer can be directly performed onto the adapted real system. If the policy does not work as expected, the domain parameters or the MARL policy should be reconsidered. We think this process is faster and lower cost than that of a conventional rule-based approach.

V. Experiments

The UVAs of our physical autonomous cooperative multi-UAV system are composed of PX4 as flight controllers and quad-rotor F450 as body frames. Additionally, differential GPS systems, front and down monocular cameras, and Jetson Nano with quad-core ARM Cortex-A57 MPCore CPU and 128-Core NVIDIA Maxwell architecture GPU as the onboard computer are equipped.

The simulator Airsim [26] is an open-source platform providing researchers with lifelike simulations in both physical and visual aspects for the development of autonomous

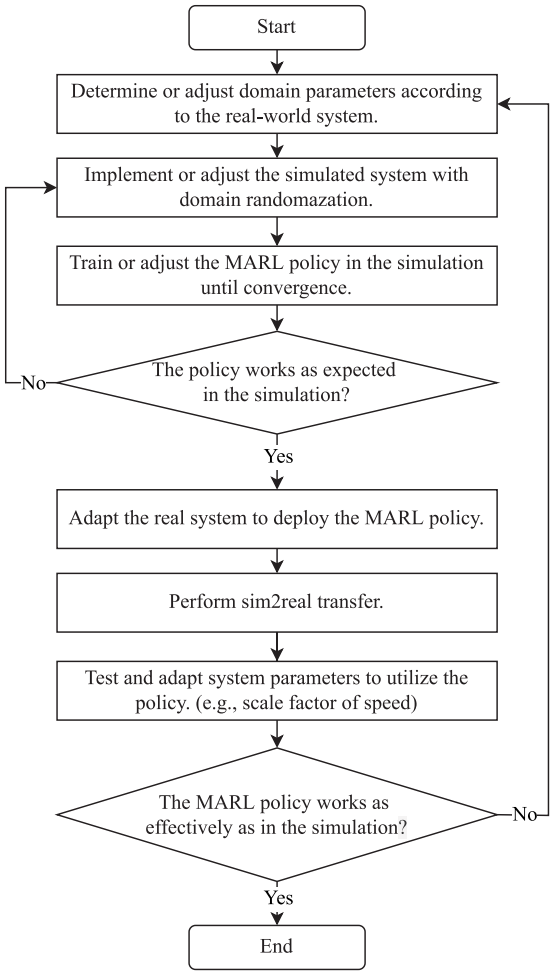


Fig. 5. Workflow of MARL sim2real transfer.

drones and vehicles. The simulated UAVs are also quad-rotor ones and equipped with front and down monocular cameras.

The target detection algorithm applied to the perception module is from our previous works [54], [55].

Five MARL policies of our cooperative multi-UAV system are trained by using five different methods in the simulated environment implemented on Airsim and then transferred to our physical multi-UAV system. The training is performed by episodes of finite length composed of discrete timesteps.

A. Multi-UAV Cooperative Goods Delivery Task

We introduce a cooperative goods delivery task under a simple setting with two UAVs in the system. An illustration of the task and an image of real-world experiments are shown in Fig. 6. Specifically, goods are tied under two UAVs with, respectively, two ropes of finite length. Two UAVs work as a team to transport the goods to the unloading point pointed out by two stationary ground markers. UAVs are initially deployed at the starting point where ground markers can be perceived and detected. The environment is free of obstacles. The task is completed when each UAV reaches and hovers directly above one arbitrary ground marker. The task is terminated if the two UAVs are either too close or too far away from each other before the task is completed.

The basic components of aforementioned Markov games for the task are defined as follows.

1) *Agent:* Each UAV is considered homogeneously as an agent that can obtain its private observation and choose to take an action according to its policy. Therefore, the task is mutiagent and of decentralized execution.
2) *Observation:* The private observation of each agent O_i consists of two parts: a) the two-dimensional (2-D) location in Cartesian coordinate (height is fixed and therefore, neglected, here, and below) of the other agent relative to itself and b) the target detection result from the perception module. The former is denoted by $o\_\text{loc}(lx, ly)$, and the latter $o\_\text{pept}(px_1, py_1, px_2, py_2)$ indicates the upper-left and lower-right pixel points for each ground marker in each video stream.
3) *Action:* The space of action A_i corresponding to high-level control interfaces is continuous, consisting of the 2-D linear velocity and the yaw angular rate, i.e., (vx, vy, yaw).
4) *Reward:* This task is cooperative and, therefore, a shared team reward that considers all agents' states is introduced. The reward function is defined as follows:

$$r = \alpha r_{d_1} + \beta r_{d_2} + \omega r_{\text{com}}$$

where α, β, and ω are scale factors. $r^t_{d_1}$ is a function of ${d_1}^t_i$, that is, the distance from UAV i to the closest ground marker at timestep t, i.e.,

$$r^t_{d_1} = \sum_i {d_1}^{t-1}_i - {d_1}^t_i.$$

$r^t_{d_2}$ is a function of d^t_2, that is, the distance between the two UAVs if the task is not terminated at timestep t otherwise it becomes a punishment function with respect to d^t_2, i.e.,

$$r^t_{d2} = \begin{cases} -\left|d^{t-1}_2 - d^t_2\right|, & \text{if not terminated} \\ \text{pun}\left(d^t_2, \text{speed}\right), & \text{otherwise} \end{cases}$$

where UAVs' speed is defined as follows:

$$\text{speed} = \sum_i \sqrt{{vx_i}^2 + {vy_i}^2}.$$

r_{com} is the completion reward, i.e.,

$$r^t_{\text{com}} = \begin{cases} \Gamma - t, & \text{if completed} \\ 0, & \text{otherwise} \end{cases}$$

where Γ is the time horizon (maximum episode length).

Additional information Z is defined as relative 2-D locations of all ground markers from the perspective of all agents.

B. Domain Parameters

Each episode of training holds a different set of domain parameters μ that are sampled following their distributions. The randomized domain parameters of μ in our work are as follows.

1) Standard deviation of observation noise for $o\_\text{loc}(lx, ly)$, denoted by σ_l.

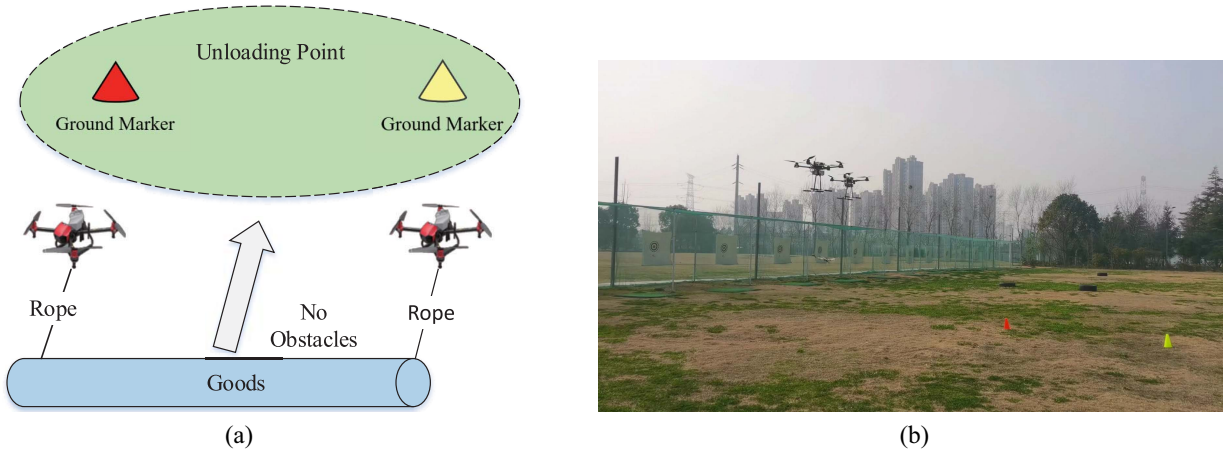


Fig. 6. (a) Illustration of the muti-UAV cooperative goods delivery task. (b) Image of real-world experiments of the task.

2) Standard deviation of observation noise for o_pept(px_1, py_1, px_2, py_2), denoted by σ_p.
3) False positive (FP) probability of target detection.
4) Flight height of each UAV.
5) Initial distance between UAVs (same for ground markers), denoted by D_1.
6) Initial distance between the UAV group and the ground marker group, denoted by D_2.

The observation noise follows the independent Gaussian distribution with zero mean and random standard deviation for each episode. FP is a target detection term, also known as false detection. Because few other objects interfere with the detection of ground markers in simulations in contrast to real-world situations, FP occurring with a certain probability is manually implemented in simulations. All parameters are sampled randomly at the beginning of each episode and held fixed during the episode. Flight height is sampled independently for each UAV, considering that the length of ropes tied under different UAVs may differ in reality. Due to the reliable high-level control interfaces provided by PX4, we do not take mass of UAVs and goods, gains for controllers, outdoor winds, etc., into consideration in this work.

C. Experiments of MARL Sim2real Transfer

First, trained with domain randomization, an R-MADDPG policy (RMRand) and an MADDPG policy (MRand) are compared to demonstrate the advantages of the memory-based method. Next in order to verify the recurrent MARL solution to nonstationary Markov games, we consider two MADDPG policies, one trained with a memory-based RNN actor and a memoryless FNN critic (RARand), and the other trained with a memoryless FNN actor and a memory-based RNN critic (RCRand). Finally, an MADDPG policy trained without domain randomization (MADDPG) is also included as a baseline.

Some MARL training settings and hyperparameters are as follows.

1) Number of episodes for training: 20 000.
2) Maximum episode length (number of timesteps): 30.
3) Learning rate for both actor and critic: 5e-3.

TABLE I
DOMAIN PARAMETERS AND THEIR RANGES

Parameter	Range
Standard Deviation σ_l (m)	[1,3]
Standard Deviation σ_p (pixel)	[5,25]
FP Probability (%)	[0.25,1]
Flight Height (m)	[1,3]
Initial Distance D_1 (m)	[2,5]
Initial Distance D_2 (m)	[5,15]

4) Discount factor γ: 0.99.
5) Update mini-batch size (number of episodes): 128.
6) Update interval (number of episodes): 16.
7) Optimizer: ADAM [56].

Every timestep takes a fixed duration of 0.8-s during which each agent performs an action (vx, vy, yaw) determined by its actor. The simulation on Airsim is accelerated, eventually leading to a training cost of about 40 h per policy.

Parameters of domain randomization are sampled uniformly with certain ranges that are shown in Table I. All ranges of domain parameters are set according to conditions of real-world experiments, e.g., the observation o_loc in reality is obtained by transforming data from onboard GPS, and the error arising from GPS noise and communication conditions is approximately between 1 and 3 m.

To compare the performance of the five policies, we evaluate them by the mean episode reward over 40 evaluation episodes every 270 training episodes. All evaluation episodes are performed with domain randomization. The evaluation curves during training are shown in Fig. 7. RMRand gets a higher convergence rate and better performance than MRand, indicating that the memory-based one is more powerful when facing domain randomization. The performance of RARand which is much better than RCRand and MRand verifies the correctness of our proposition that a CTDE MARL algorithm with memory-based decentralized policies is a solution to nonstationary Markov games. Moreover, the low-performance gap between RARand and RMRand shows that a memory-based

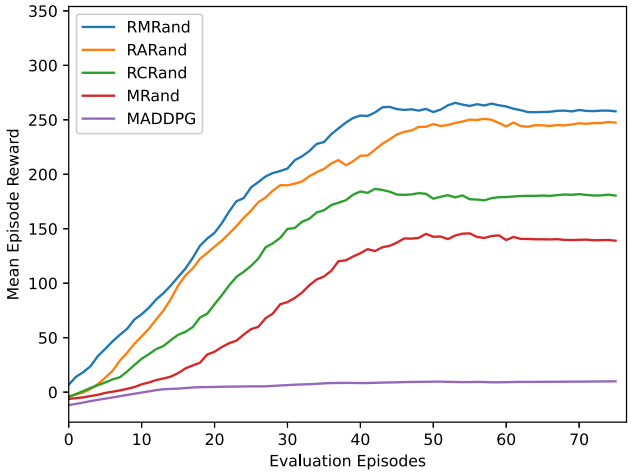


Fig. 7. Evaluation curves of five MARL policies.

TABLE II
POLICY PERFORMANCE OF SIMULATED EXPERIMENTS

Policy	Completion Rate (Sim)	Completion Timesteps (Sim)
RMRand	76.0%	12.2
MRand	71.0%	16.0
RARand	73.2%	12.2
RCRand	75.5%	15.6
MADDPG	37.2%	18.9

TABLE III
POLICY PERFORMANCE OF REAL-WORLD EXPERIMENTS

Policy	Completion Rate (Real)	Completion Time (Real)
RMRand	8/10	51.1s
MRand	6/10	60.2s
RARand	7/10	52.8s
RCRand	6/10	57.0s
MADDPG	2/10	54.2s

actor is more important than a memory-based critic for nonstationary Markov games since the improvement of the actor benefits the policy more directly. Comparing MRand with MADDPG, it is shown that a memoryless policy can learn to adapt to domain randomization to a certain extent.

In addition, Table II compares the five policies after training by task completion rates and mean values of completion timesteps over 1000 testing episodes for each policy. Completion rates of the other four policies except MADDPG are relatively close, so comparing them by completion timesteps, the same result can be concluded that RMRand and RARand are of similar performance and they are better than MRand and RCRand.

Directly transferring the policies from simulations to the real world, we perform real-world experiments on our autonomous cooperative multi-UAV system to complete the cooperative goods delivery task. Table III compares the real-world performance of all five directly transferred policies. RMRand has the highest completion rate and time efficiency, showing that it is most capable of bridging the reality gap in nonstationary systems. Comparing with RCRand, RARand's performance is closer to RMRand, indicating that memory-based actors play a more vital role in MARL sim2real transfer. The fact that MADDPG, the policy trained without domain randomization, cannot adapt to the real-world environment shows the significance of training MARL policies with domain randomization in improving their generalization performance.

VI. CONCLUSION

To empower digital entities with metaverse intelligence and bridge the reality gap in dynamic and uncertain metaverse systems, this article defines a nonstationary variant of Markov games and proposes a recurrent MARL solution to it. Specifically, we prove theoretically recurrent MARL is capable of solving the nonstationary Markov games so that it can be leveraged with domain randomization and perception-control modularization to perform MARL sim2real transfer. Unlike most MARL studies that are confined to simulations, we successfully transfer MARL policies onto our physical autonomous cooperative multi-UAV system to accomplish multi-UAV cooperative goods delivery, empirically demonstrating the high generalization performance of recurrent MARL.

As an emerging and promising research field, MARL sim2real transfer needs more investigation. We will further verify different recurrent MARL algorithms with more varied domain parameters from a broader scope, and explore more efficient sim2real transfer techniques in theory and practice. We hope this work inspires more studies on MARL sim2real transfer for robot applications, and arouses more attention to metaverse intelligence.

REFERENCES

[1] F.-Y. Wang, "Parallel intelligence in metaverses: Welcome to HANOI!" *IEEE Intell. Syst.*, vol. 37, no. 1, pp. 16–20, Jan./Feb. 2022.

[2] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," in *Proc. AAAI Conf. Artif. Intell.*, vol. 31, 2017, pp. 1553–1561.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 3389–3396.

[5] W. He, H. Gao, C. Zhou, C. Yang, and Z. Li, "Reinforcement learning control of a flexible two-link manipulator: An experimental investigation," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 12, pp. 7326–7336, Dec. 2021.

[6] M. Pei, H. An, B. Liu, and C. Wang, "An improved dyna-Q algorithm for mobile robot path planning in unknown dynamic environment," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 7, pp. 4415–4425, Jul. 2022.

[7] J. Yu, Z. Wu, X. Yang, Y. Yang, and P. Zhang, "Underwater target tracking control of an untethered robotic fish with a camera stabilizer," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 10, pp. 6523–6534, Oct. 2021.

[8] J. Yan, X. Li, X. Yang, X. Luo, C. Hua, and X. Guan, "Integrated localization and tracking for AUV with model uncertainties via scalable sampling-based reinforcement learning approach," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 11, pp. 6952–6967, Nov. 2022.

[9] J. Liu, Z. Huang, X. Xu, X. Zhang, S. Sun, and D. Li, "Multi-kernel online reinforcement learning for path tracking control of intelligent vehicles," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 11, pp. 6962–6975, Nov. 2021.

[10] J. Chen, T. Shu, T. Li, and C. W. de Silva, "Deep reinforced learning tree for spatiotemporal monitoring with mobile robotic wireless sensor networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 11, pp. 4197–4211, Nov. 2020.

[11] G. Wen, W. Hao, W. Feng, and K. Gao, "Optimized backstepping tracking control using reinforcement learning for quadrotor unmanned aerial vehicle system," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 8, pp. 5004–5015, Aug. 2022.

[12] Y. Wang, J. Sun, H. He, and C. Sun, "Deterministic policy gradient with integral compensator for robust quadrotor control," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 10, pp. 3713–3725, Oct. 2020.

[13] Z. He, L. Dong, C. Sun, and J. Wang, "Asynchronous multithreading reinforcement-learning-based path planning and tracking for unmanned underwater vehicle," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 5, pp. 2757–2769, May 2022.

[14] Q. Xiao, C. Li, Y. Tang, and L. Li, "Meta-reinforcement learning of machining parameters for energy-efficient process control of flexible turning operations," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 1, pp. 5–18, Jan. 2021.

[15] X. Zhao, C. Li, Y. Tang, and J. Cui, "Reinforcement learning-based selective disassembly sequence planning for the end-of-life products with structure uncertainty," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7807–7814, Oct. 2021.

[16] Z. Zhou and G. Liu, "RoMFAC: A robust mean-field actor-critic reinforcement learning against adversarial perturbations on states," 2022, *arXiv:2205.07229*.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2017, pp. 23–30.

[18] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018, pp. 3803–3810.

[19] J. Hwangbo et al., "Learning agile and dynamic motor skills for legged robots," *Sci. Robot.*, vol. 4, no. 26, 2019, Art. no. eaau5872.

[20] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Trans. Robot.*, vol. 36, no. 1, pp. 1–14, Feb. 2020.

[21] J. Tan et al., "Sim-to-real: Learning agile locomotion for quadruped robots," 2018, *arXiv:1804.10332*.

[22] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 895–902, Apr. 2018.

[23] W. C. Martin, A. Wu, and H. Geyer, "Robust spring mass model running for a physical bipedal robot," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2015, pp. 6307–6312.

[24] E. Candela, L. Parada, L. Marques, T.-A. Georgescu, Y. Demiris, and P. Angeloudis, "Transferring multi-agent reinforcement learning policies for autonomous driving using sim-to-real," 2022, *arXiv:2203.11653*.

[25] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proc. ICML*, 1994, pp. 157–163.

[26] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*. Cham, Switzerland: Springer, 2018, pp. 621–635.

[27] R. E. Wang, M. Everett, and J. P. How, "R-MADDPG for partially observable environments and limited communication," 2020, *arXiv:2002.06684*.

[28] J. Joshua, "Information bodies: Computational anxiety in Neal Stephenson's snow crash," *Interdiscipl. Literary Stud.*, vol. 19, no. 1, pp. 17–47, 2017.

[29] "Introducing Meta: A social technology company." Meta. 2021. [Online]. Available: https://about.fb.com/news/2021/10/facebook-company-is-now-meta/

[30] J. Liang, Y. Tang, R. Hare, B. Wu, and F.-Y. Wang, "A learning-embedded attributed Petri net to optimize student learning in a serious game," *IEEE Trans. Comput. Social Syst.*, early access, Dec. 23, 2021, doi: 10.1109/TCSS.2021.3132355.

[31] Y. Tang, J. Liang, R. Hare, and F.-Y. Wang, "A personalized learning system for parallel intelligent education," *IEEE Trans. Comput. Social Syst.*, vol. 7, no. 2, pp. 352–361, Apr. 2020.

[32] F.-Y. Wang, Y. Tang, X. Liu, and Y. Yuan, "Social education: Opportunities and challenges in cyber-physical-social space," *IEEE Trans. Comput. Social Syst.*, vol. 6, no. 2, pp. 191–196, Apr. 2019.

[33] R. Mourning and Y. Tang, "Virtual reality social training for adolescents with high-functioning autism," in *Proc. IEEE Int. Conf. Syst. Man Cybern. (SMC)*, 2016, pp. 004848–004853.

[34] C. Franzwa, Y. Tang, A. Johnson, and T. Bielefeldt, "Balancing fun and learning in a serious game design," *Int. J. Game-Based Learn.*, vol. 4, no. 4, pp. 37–57, 2014.

[35] S. Mystakidis, "Metaverse," *Encyclopedia*, vol. 2, no. 1, pp. 486–497, 2022.

[36] Q. Yang, Y. Zhao, H. Huang, Z. Xiong, J. Kang, and Z. Zheng, "Fusing blockchain and AI with metaverse: A survey," *IEEE Open J. Comput. Soc.*, vol. 3, pp. 122–136, 2022.

[37] X. Wang, J. Yang, J. Han, W. Wang, and F.-Y. Wang, "Metaverses and deMetaverses: From digital twins in CPS to parallel intelligence in CPSS," *IEEE Intell. Syst.*, vol. 37, no. 4, pp. 97–102, Jul./Aug. 2022.

[38] F. A. Oliehoek, M. T. Spaan, and N. Vlassis, "Optimal and approximate Q-value functions for decentralized POMDPs," *J. Artif. Intell. Res.*, vol. 32, pp. 289–353, May 2008.

[39] L. Kraemer and B. Banerjee, "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, vol. 190, pp. 82–94, May 2016.

[40] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6379–6390.

[41] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," 2021, *arXiv:2103.01955*.

[42] O. Vinyals et al., "Grandmaster level in starCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[43] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," 2017, *arXiv:1702.03037*.

[44] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, and H. Yan, "Toward packet routing with fully distributed multiagent deep reinforcement learning," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 2, pp. 855–868, Feb. 2022.

[45] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," 2016, *arXiv:1610.03295*.

[46] J. Hu, H. Zhang, L. Song, R. Schober, and H. V. Poor, "Cooperative Internet of UAVs: Distributed trajectory design by multi-agent deep reinforcement learning," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 6807–6821, Nov. 2020.

[47] A. Shamsoshoara, M. Khaledi, F. Afghah, A. Razi, and J. Ashdown, "Distributed cooperative spectrum sharing in UAV networks using multi-agent reinforcement learning," in *Proc. 16th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2019, pp. 1–6.

[48] Y. Zhang, Z. Zhuang, F. Gao, J. Wang, and Z. Han, "Multi-agent deep reinforcement learning for secure UAV communications," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2020, pp. 1–5.

[49] J. Cui, Y. Liu, and A. Nallanathan, "Multi-agent reinforcement learning-based resource allocation for UAV networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 2, pp. 729–743, Feb. 2020.

[50] H. Qie, D. Shi, T. Shen, X. Xu, Y. Li, and L. Wang, "Joint optimization of multi-UAV target assignment and path planning based on multi-agent reinforcement learning," *IEEE access*, vol. 7, pp. 146264–146272, 2019.

[51] T. Zhang, Y. Li, S. Li, Q. Ye, C. Wang, and G. Xie, "Decentralized circle formation control for fish-like robots in the real-world via reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021, pp. 8814–8820.

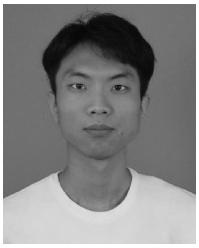
[52] V. R. Nema and B. Ravindran, "Interactive robust policy optimization for multi-agent reinforcement learning," in *Proc. Deep RL Workshop NeurIPS*, 2021, pp. 1–10.

[53] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[54] P. Zhou, G. Liu, J. Wang, Q. Weng, K. Zhang, and Z. Zhou, "Lightweight unmanned aerial vehicle video object detection based on spatial-temporal correlation," *Int. J. Commun. Syst.*, vol. 35, no. 17, 2022, Art. no. e5334.

[55] Z. Zhang, J. Liu, G. Liu, J. Wang, and J. Zhang, "Robustness verification of swish neural networks embedded in autonomous driving systems," *IEEE Trans. Comput. Social Syst.*, early access, Jun. 9, 2022, doi: 10.1109/TCSS.2022.3179659.

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.



Haoran Shi received the B.S. degree in software engineering from Donghua University, Shanghai, China, in 2020. He is currently pursuing the M.S. degree in computer software and theory with Tongji University, Shanghai.

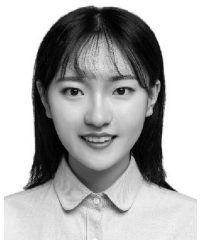
His current research interests include reinforcement learning and learning control systems.

Guanjun Liu (Senior Member, IEEE) received the Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2011.

He was a Postdoctoral Research Fellow with the Singapore University of Technology and Design, Singapore, from 2011 to 2013, and a Postdoctoral Research Fellow with the Humboldt University of Berlin, Berlin, Germany, from 2013 to 2014, supported by the Alexander von Humboldt Foundation. He is currently a Professor with the Department of Computer Science, Tongji University. He has authored over 130 papers and three books. His research interests include Petri net theory, model checking, machine learning, cyber–physical systems, workflow, and credit card fraud detection.

Kaiwen Zhang received the M.S. degree in automation engineering from the Shanghai University of Electric Power, Shanghai, China, in 2021. He is currently pursuing the Ph.D. degree in computer software and theory with Tongji University, Shanghai, China.

His current research interests include software analysis, Petri net, and formal verification.



Ziyuan Zhou received the B.S. degree in computer science and technology from the China University of Mining and Technology, Xuzhou, China, in 2020. She is currently pursuing the Ph.D. degree in computer software and theory with Tongji University, Shanghai, China.

Her research interests include reinforcement learning and multiagent system.

Jiacun Wang (Senior Member, IEEE) received the Ph.D. degree in computer engineering from the Nanjing University of Science and Technology, Nanjing, China, in 1991.

He was a Research Associate with the School of Computer Science, Florida International University, Miami, FL, USA. From 2001 to 2004, he was a member of Scientific Staff with Nortel Networks, Richardson, TX, USA. He is currently a Professor of Software Engineering with Monmouth University, West Long Branch, NJ, USA. He has authored *Timed Petri Nets: Theory and Application Kluwer* (1998), *Real-Time Embedded Systems* (Wiley, 2018), and *Formal Methods in Computer Science* (CRC, 2019), edited *Handbook of Finite State Based Models and Applications* (CRC, 2012), and published over 130 research papers in journals and conferences. His research interests include software engineering, discrete event systems, formal methods, machine learning, and real-time distributed systems.

Dr. Wang served as the general chair, the program chair, the special session chair, or the steering committee chair for many international conferences. He was an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS PUBLICATION INFORMATION. He is currently an Associate Editor of IEEE/CAA JOURNAL OF AUTOMATICA SINICA.