Closing the Dynamics Gap via Adversarial and Reinforcement Learning for High-Speed Racing

Jingyu Niu
Research Center for Intelligent Computing Systems
State Key Laboratory of Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences
Beijing, China
niujingyu17b@ict.ac.cn

Yu Hu*
Research Center for Intelligent Computing Systems
State Key Laboratory of Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences
Beijing, China
huyu@ict.ac.cn

Wei Li
Research Center for Intelligent Computing Systems
State Key Laboratory of Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences
Beijing, China
liwei2019@ict.ac.cn

Guangyan Huang
School of Information Technology
Deakin University
Melbourne, Australia
guangyan.huang@deakin.edu.au

Yinhe Han
Research Center for Intelligent Computing Systems
State Key Laboratory of Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences
Beijing, China
yinhes@ict.ac.cn

Xiaowei Li
Research Center for Intelligent Computing Systems
State Key Laboratory of Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences
Beijing, China
lxw@ict.ac.cn

2022 International Joint Conference on Neural Networks (IJCNN) | 978-1-7281-8671-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IJCNN55064.2022.9892424

***Abstract*—Autonomous racing has lately gained popularity because of its entertainment value and potential of advancing autonomous driving in high-speed situations. These high-speed racing efforts usually focus on a road domain with fixed dynamics. They cannot meet the challenge of policy adaptation between domains with large dynamics gaps. Meanwhile, existing policy adaptation methods either rely on experts to build new environments for policy training, or only handle a small dynamics gap for low-speed control tasks due to limited dynamics modeling and rigorous data collection assumptions. To overcome these drawbacks, we introduce DAARL, a novel policy adaptation algorithm that uses adversarial and reinforcement learning to bridge the large dynamics gap between different domains. It has two training stages. In the first training stage, a domain transfer function is learned by adversarial learning to better capture the dynamics gap. The single domain transfer function integrates with the source domain to implement the dynamics of different target domains virtually without the help of experts. We name these virtual domains the imaginary target domains. In the second training stage, the knowledge of the source-domain policy guides the reinforcement learning of a target-domain policy on an imaginary target domain. It improves the convergence of the target-domain policy. Five experiments have been conducted on a racing simulator with different road domains. All results show that DAARL outperforms baselines in terms of driving speed, stability, success rate, and domain scalability.**

***Index Terms*—policy adaptation, reinforcement learning, adversarial learning, autonomous racing**

This work was supported by National Natural Science Foundation of China under Grant No. 62176250 and No. 62003323, in part by the Innovation Project of State Key Laboratory of Computer Architecture, ICT, CAS under Grant No. CARCH5202 and No. CARCH5406.

*Corresponding author

I. Introduction

Autonomous racing has recently received increasing attention. The relevant international autonomous racing competitions include Roborace [1], Indy Autonomous Challenge [2], F1/10 autonomous racing [3], and Autonomous Formula SAE [4]. Autonomous racing not only keeps traditional racing entertaining, but also facilitates the development of autonomous driving in high-speed situations. These works primarily endeavor to find an optimal policy for a road domain with fixed dynamics parameters (including the friction coefficient, the rolling resistance coefficient, and roughness). However, always driving fast and stably on roads with changing dynamics is crucial for racing cars. For example, there are asphalt roads, dirt roads, and roads with gathered water in the rally race [5]. It is widely known that an optimal policy learned on the road domain whose dynamics are known (called the source domain) often fails to drive well on another road domain with a large dynamics gap (called the target domain). Our goal of this research is to adapt the racing policy between road domains with large dynamics gaps. Because the development of autonomous racing is complex and costly, it is necessary to set up a simulation environment for each unknown road domain to implement and test algorithms before they are deployed in the real world [6]. Setting up a simulation environment usually requires a lot of experts' effort and time cost. Therefore, we seek an efficient policy adaptation algorithm that can meet both racing requirements. One is to drive as fast and stably as possible on road domains with large dynamics gaps. The other is to implement the algorithm using only the existing source domain. There is no need to follow a serial development process from building environments for new domains to algorithm implementation and then to performance testing, because algorithm implementation can be done in parallel with building new environments. It accelerates the early algorithm development of autonomous racing.

Our topic is further refined as RL-based policy adaptation between domains with different dynamics, since deep reinforcement learning (DRL) [7] is a potential solution for racing policy learning [8], [9]. It belongs to a subset of policy transfer. There are two categories of previous works. One is to learn a robust policy for all domains [10]–[18]. The learned policy is sub-optimal due to the compromise between different domains. And it performs worse as the dynamics gap becomes larger. This category cannot meet the racing requirement of driving as fast and stably as possible through all road domains.

The other category can achieve the above requirement by learning an optimal policy for each domain. It contains fine-tuning methods and source augmentation methods. Fine-tuning methods [19]–[21] use the knowledge of the source-domain policy (the source policy for short) to guide the convergence of the target-domain policy (the target policy for short). They converge faster and better than learning from scratch. But these methods spend time building a training environment for each domain by experts. Our requirement of accelerating the development process is not met in such methods. Source augmentation methods [22]–[25] can speed up the development process. They learn a dynamics mapping function to bridge the dynamics gap between a source domain and a target domain. The dynamics of the target domain are implemented by the source domain and the mapping function. Thus, their algorithm implementation processes do not wait for building new environments for target domains. However, the optimization of their mapping functions and data collection assumptions are designed for specific low-speed robotic tasks. It does not provide sufficient modeling capability to bridge a larger dynamics gap in complex racing tasks. The domain scalability of these methods is poor because each new target domain adds a new mapping function. Besides, these methods do not exploit the potential of the source domain in improving the convergence efficiency of the target-domain policy.

We propose a novel policy adaptation algorithm that uses adversarial and reinforcement learning to satisfy all requirements of a racing challenge on road domains with different dynamics. We call it DAARL for short. Our method combines the best of the fine-tuning idea and the source augmentation idea. It consists of two training stages. In the first training stage, a domain transfer function is learned by adversarial learning [26]. It is used to create virtual environments for target domains based on an accessible source domain without experts' help. We call the virtual target environment the imaginary target domain since it does not exist in the simulation or the real world. Thanks to the combination of the adversarial loss, the domain classification loss, and the reconstruction loss, a single domain transfer function better models the dynamics of different domains using unpaired and randomly collected data. Its domain scalability is improved. And a more accurate imaginary target domain can be created to bridge a large dynamics gap. In the second training stage, an optimal target policy is trained on the imaginary target domain by DRL. We exploit the knowledge of the source policy to guide the target policy to converge faster and better than training from scratch.

The main contributions are threefold. First, we present a novel policy adaptation algorithm between domains with different dynamics by adversarial and reinforcement learning, namely DAARL. The introduction of adversarial learning allows us to better model the dynamics of different domains by a single domain transfer function. As a result, our algorithm bridges the large dynamics gap and has good domain scalability. There are few restrictions on the collection of training data, which is practical for high-speed applications. The target policy is trained on an imaginary target domain, which is the combination of the domain transfer function and the source domain. It avoids reliance on experts. In addition, we use the knowledge of source-domain policy to help the target policy converge faster and better. Second, we apply two representative reinforcement learning methods separately to optimize the target policy. We also implement two separate policy network structures that can exploit the source-policy knowledge. It shows the flexibility of our algorithm. Third, DAARL is applied to a challenging high-speed autonomous racing task on roads with varied dynamics. We evaluate it in the 3D racing simulator TORCS [27]. Experimental results show that DAARL is superior to baselines in the aspects of driving speed, stability, success rate, and domain scalability.

II. Related Work

RL-based policy adaptation between different domains mainly bridges two different domain gaps, namely the visual gap and the dynamics gap. The visual gap occurs when the policy input is the image, such as light and background [28], [29]. It can be avoided when the non-visual multi-sensor information is used as the input for control tasks. The dynamics gap is induced by different physical parameters of environments, such as the friction coefficient and the agent mass. Our work focuses on bridging the dynamics gap. Related methods can be broadly divided into two categories.

The main objective of the first category is to learn a robust policy that fits all domains. It is subdivided into domain randomization (DR) methods and domain-invariant feature methods. DR methods build several source domains by randomizing physical parameters for policy training [10]–[14]. However, DR methods rely on experts to set these parameter ranges and engineer source domains. Domain-invariant feature methods assume that high-level motion features of the source policy are also feasible for the target domain [15]–[18]. A common feature is the trajectory. The first category uses the idea of averaging domains [30] to achieve policy robustness. The learned policy is sub-optimal. It cannot meet the racing requirement of driving at its best on all domains. Moreover, the policy is hard to converge on road domains with a large dynamics gap.

The main objective of the second category is to learn their optimal policies for different dynamics domains with the help of the source domain. It satisfies the racing requirement above. There are two subcategories, fine-tuning methods and source augmentation methods. Fine-tuning methods [19]–[21] use the knowledge of the source policy to guide the convergence of the target policy efficiently. Nageshrao et al. [20] take the parameters of a source policy as the initialization parameters of a target policy at the beginning of training. We call this intuitive approach directly fine-tuning (D-FT). Rusu et al. [21], [31] propose the progressive neural network (PNN) to fully reuse the hierarchical features of the source policy throughout the whole training of the target policy. But, their training processes need experts to build a new environment for matching the dynamics of each target domain. It is time-consuming for the algorithm development of a racing task.

Source augmentation methods [22]–[25] use the existing source domain to train the target policy without building an additional environment. First, they learn a mapping function to compensate for the dynamics gap between the source domain and the target domain. Then, the dynamics of a target domain are equal to the dynamics of the source domain coupled with the mapping function. The idea of augmenting a source domain as an imaginary target domain for policy training saves the time of developing a complex racing algorithm.

However, these methods are strict with data collection during the learning of the mapping function. Each pair of motion data collected from two domains in Neural-Augmented Simulation (NAS) [22] has the same start state and action. It is unfeasible to align data, such as speed, all the time for high-speed racing tasks. A series of grounding methods [23]–[25] collect data from specific policies. It restricts the generalization ability of the mapping function and the diversity of data. The mapping function in [23] uses a regression loss function to optimize. [24] and [25] replace the regression loss with the RL loss to improve the accuracy of modeling dynamics. However, when both the mapping function and the target policy are trained by DRL, their convergences are time-consuming and difficult to reach. These disadvantages make them only suitable for low-speed robotic tasks. Unlike the above methods, Jiang et al. [32] assume the source domain is a hybrid simulator. It combines an analytical dynamics model and partial learnable parameters. The mapping function learns these parameters to change the source domain to the target domain by adversarial learning. Different domains collect training data from the same policy. It is a strict assumption that prevents it from bridging a large dynamics gap. Thus, [32] is difficult for complex racing tasks. Because it still relies on experts to design a powerful simulator and choose appropriate parameters. Moreover, as the number of new target domains increases, so does the number of the mapping functions in the above methods. Since this subcategory always trains the target policy from scratch, it makes no progress in accelerating policy training.

Our method is most relevant to source augmentation methods. The advantages of our method are as follows. (a) The combination of the adversarial loss, the domain classification loss, and the reconstruction loss is used in learning a dynamics mapping function for domain transfer. We call this function the domain transfer function. Note that some policy adaptation methods [28], [29] have adopted adversarial learning for image translation [33], [34]. They can only bridge the visual gap. Unlike them, we use adversarial learning for dynamics translation. The optimization of our domain transfer function models the dynamics of different domains more accurately by sampling random trajectories from all domains. It contributes to bridging the large dynamics gap. This easier way of collecting data is practical for high-speed racing tasks. (b) We use a single domain transfer function to create different imaginary target domains. It is scalable for increasing target domains. (c) We incorporate the idea of fine-tuning methods into the training of target policy. In this way, our method not only uses the source policy to help the target policy converge effectively, but also avoids the building cost of fine-tuning methods because of the proposed imaginary target domain.

III. APPROACH

A. Problem Formulation

In this study, we consider a policy adaptation problem to bridge the dynamics gap between different road domains for a high-speed racing task. Each source domain $\mathcal{S}$ and each target domain $\mathcal{T}i$, $i = 1, 2, \ldots$ is a Markov Decision Process. This work assumes that different domains have the same state space S, action space A, and reward function r_t. The difference between domains is the transition function of dynamics, $P^{\mathcal{S}}(s_{t+1}|s_t, a_t)$ and $P^{\mathcal{T}i}(s_{t+1}|s_t, a_t)$. t is the time step. We want to exploit the source domain to learn an optimal target policy $\pi^{\mathcal{T}i}(s_t)$. The objective is to maximize the expected discounted sum of future rewards for the target domain, $E_{P^{\mathcal{T}i}}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where the discount factor $\gamma \in [0, 1]$ controls the influence of future rewards.

B. DAARL Overview

In order to solve the formulated problem, we introduce an novel algorithm to adapt the policy between domains with different dynamics via adversarial and reinforcement learning. We call it DAARL for short. Fig. 1(a) shows it is a two-stage algorithm. The goal of the first training stage is to learn a scalable domain transfer function (the blue block in Fig. 1(a)) by adversarial learning. Its scalability is reflected in that one function can cover the transfer between multiple domains. The training data is the trajectories collected from the source domain and different target domains by acting randomly and setting random starting points. It is an easier way than other source augmentation methods, because any driver passing through one domain can provide valid data

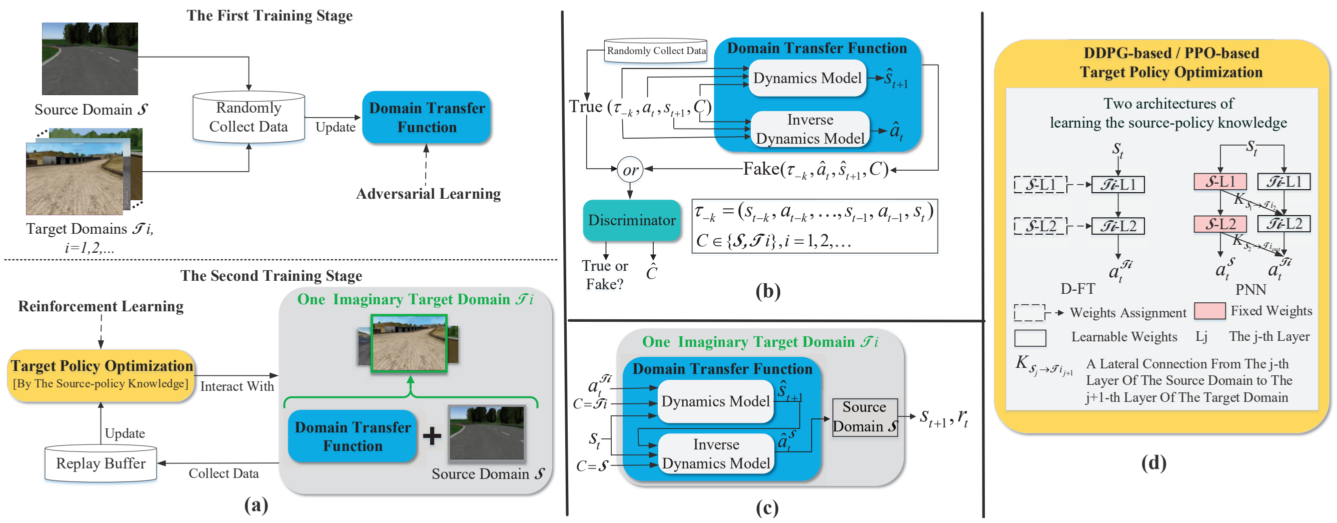
The First Training Stage
Source Domain $\mathcal{S}$
Target Domains $\mathcal{T}i$, $i=1,2,\ldots$
Randomly Collect Data
Update
Domain Transfer Function
Adversarial Learning
The Second Training Stage
Reinforcement Learning
Target Policy Optimization [By The Source-policy Knowledge]
Interact With
One Imaginary Target Domain $\mathcal{T}i$
Update
Replay Buffer
Collect Data
Source Domain $\mathcal{S}$
(a)
Randomly Collect Data
Domain Transfer Function
Dynamics Model
Inverse Dynamics Model
True $(\tau_{-k}, a_t, s_{t+1}, C)$
Fake$(\tau_{-k}, \hat{a}_t, \hat{s}_{t+1}, C)$
or
Discriminator
$\tau_{-k} = (s_{t-k}, a_{t-k}, \ldots, s_{t-1}, a_{t-1}, s_t)$
$C \in \{\mathcal{S}, \mathcal{T}i\}, i = 1,2,\ldots$
True or Fake?
$\hat{C}$
(b)
One Imaginary Target Domain $\mathcal{T}i$
Domain Transfer Function
Dynamics Model
Inverse Dynamics Model
Source Domain $\mathcal{S}$
s_{t+1}, r_t
(c)
DDPG-based / PPO-based Target Policy Optimization
Two architectures of learning the source-policy knowledge
D-FT
PNN
Weights Assignment
Fixed Weights
Learnable Weights
Lj The j-th Layer
A Lateral Connection From The j-th Layer Of The Source Domain to The j+1-th Layer Of The Target Domain
(d)

Fig. 1. Overview of the proposed DAARL algorithm. (a) DAARL has two training stages. The first training stage (top) is to learn the domain transfer function by adversarial learning. The training data is collected from the source and the target domain by randomly selecting starting positions and actions. The second training stage (bottom) is to learn the target policy by reinforcement learning with the help of the source-policy knowledge. (b) The domain transfer function consists of a dynamics model and an inverse dynamics model. The number of the domain transfer function does not increase as the number of the target domains grows. (c) The imaginary target domain is implemented by the transfer function and the source domain. (d) The target policy adopts two network structures to accelerate the convergence. D-FT (left) is a directly fine-tuning way [10]. PNN (right) is a progressive neural network way [21], [31].

for DAARL regardless of driving skill. The learned domain transfer function bridges the dynamics gap between the target domain and the source domain. Consequently, an imaginary target domain (gray shaded block in Fig. 1(a)) is implemented with the combination of the existing source domain and the domain transfer function.

The goal of the second training stage is to optimize a target policy (the yellow block in Fig. 1(a)) efficiently on the imaginary target domain by reinforcement learning. The efficiency of policy convergence is reflected in using the knowledge of the source policy as the guidance of the agent's movement. It leverages the idea of fine-tuning methods, but avoids building a new training environment by experts. The detailed design of DAARL is described below.

C. Learning The Domain Transfer Function

Inspired by the scalability of the StarGAN method [35], which uses only a single model for image translation between multiple domains, we borrow this idea to learn a single domain transfer function for bridging the dynamics gap between any two domains rather than the visual gap. The details of the first training stage of DAARL are depicted in Fig. 1(b). The domain transfer function acts as the generator G of the adversarial learning framework. It consists of a dynamics model and an inverse dynamics model. The dynamics model $f_{dyn}(\tau_{-k}, a_t, C) = \hat{s}_{t+1}$ is trained to predict the next state $\hat{s}_{t+1}$ based on the recent history of the agent state $\tau_{-k} = (s_{t-k},\ a_{t-k}, \ldots, s_{t-1}, a_{t-1}, s_t)$, the current action a_t, and the domain label $C \in \{\mathcal{S}, \mathcal{T}i\}, i = 1, 2, \ldots$. Note that the domain label does not contain concrete road information. It is just a unique indicator to classify different domains. The inverse dynamics model $f_{inv}(\tau_{-k}, s_{t+1}, C) = \hat{a}_t$ is trained to compute the current action $\hat{a}_t$ based on τ_{-k}, the next state s_{t+1}, and the domain label C.

The collected trajectories $(\tau_{-k},\ a_t,\ s_{t+1},\ C) = (\chi,\ C)$ are treated as true data. They are fed into the domain transfer function G and output the predicted trajectories with the same domain label, $(\tau_{-k},\ \hat{a}_t,\ \hat{s}_{t+1},\ C) = (\hat{\chi},\ C) = (G(\chi, C),\ C) = (\tau_{-k},\ f_{inv}(\tau_{-k},\ s_{t+1},\ C),\ f_{dyn}(\tau_{-k},\ a_t,\ C),\ C)$. The predicted trajectories are treated as fake data. The domain transfer function G aims at making the fake data true enough to confuse the discriminator D (the green block in Fig. 1(b)) of the adversarial learning framework. The discriminator D in our method has two functions: D_{tf} is to distinguish between true trajectories χ and fake trajectories $\hat{\chi}$, and D_{cls} is to classify each trajectory into its domain, i.e., $D : \chi \rightarrow \{D_{tf}(\chi), D_{cls}(\chi)\}$. The optimization objectives of the first training stage are described as follows.

1) Adversarial Loss: The objective of a GAN [26] can be expressed as

$$\begin{aligned} L_{adv} =& \mathrm{E}_{\chi}[\log D_{tf}(\chi)] + \\ & \mathrm{E}_{\chi, C}[\log(1 - D_{tf}(G(\chi,\ C)))], \end{aligned} \tag{1}$$

where the domain transfer function G tries to minimize this objective while the discriminator D acts against it to maximize this objective. To stabilize and improve the training process, the Wasserstein GAN objective with gradient penalty [36], [37] is used instead of (1), which can be expressed as

$$\begin{aligned} L_{adv} =& \mathrm{E}_{\chi}[D_{tf}(\chi)] - \mathrm{E}_{\chi,\ C}[D_{tf}(G(\chi,\ C))] - \\ & \lambda_{gp}\mathrm{E}_{\chi_h}[(||\nabla_{\chi_h} D_{tf}(\chi_h)||_2 - 1)^2], \end{aligned} \tag{2}$$

where $\chi_h = \varepsilon\chi + (1-\varepsilon)\hat{\chi}$, $\varepsilon \sim U[0,\ 1]$, λ_{gp} is the gradient penalty coefficient.

2) Domain Classification Loss: An auxiliary classifier D_{cls} is added to our discriminator. It works as a domain constraint to help the convergence of D and G. The loss term for training D is

$$L_{dcls} = \mathrm{E}_{\chi,\, C}[-\log D_{cls}(C|\chi)], \quad (3)$$

where $D_{cls}(C|\chi)$ denotes a probability distribution over domain labels when a true trajectory is entered into D. Minimizing L_{dcls} guides D to classify the true trajectory to its own domain C. Meanwhile, the loss term for training G is

$$L_{gcls} = \mathrm{E}_{\chi,\, C}[-\log D_{cls}(C|G(\chi,\, C))]. \quad (4)$$

By minimizing L_{gcls}, G is expected to generate a more accurate trajectory, which can confuse D and be correctly identified with its corresponding domain C.

3) Reconstruction Loss: This work focuses on a high-speed racing task to handle the large dynamics gap between different road domains. The trajectory input χ contains non-visual multi-sensor states and two-dimensional actions to avoid the visual gap. Details of states and actions are shown in the last paragraph of section IV-A. The reconstruction loss is designed to ensure that G learns a detailed description of the dynamics changes. For the dynamics model in G, the reconstruction loss term L_{dyn} is a weighted mean squared error between the true and the predicted next state, as shown in (5).

$$L_{dyn} = \frac{1}{M}\sum_{j=1}^{M}\sum_{n=1}^{N} b_n \big(s_{t+1}^{jn} - f_{dyn}^{jn}({\tau_{-k}}^{j},\, a_t^j,\, C^j)\big)^2, \quad (5)$$

where M is the batch size, N is the number of sensors, j is the jth data in each sampled batch, s_{t+1}^{jn} and f_{dyn}^{jn} represent the nth sensor from the true next state vector and the predicted next state vector, respectively, on the same domain C^j, b_n is the weighted coefficient.

For the inverse dynamics model in G, the loss term L_{inv} is the L1 norm between the true and the predicted current action vector, as shown in (6),

$$L_{inv} = \mathrm{E}_{\tau_{-k},\, s_{t+1},\, C}[||a_t - f_{inv}(\tau_{-k},\, s_{t+1},\, C)||_1]. \quad (6)$$

4) Full Objective: After combining the above mentioned objectives, the full G loss function is

$$L_G = w_1 L_{adv} + w_2 L_{gcls} + w_3 L_{dyn} + w_4 L_{inv}, \quad (7)$$

where w_1, w_2, w_3, and w_4 are the weighted coefficients that balance the influence of different loss functions. The full D loss function is

$$L_D = -w_1 L_{adv} + w_2 L_{dcls}. \quad (8)$$

The first training stage helps DAARL outperform previous works in terms of prediction accuracy and domain scalability. In addition, our discriminator is not only useful during the training process. Its classification ability also plays a role during the execution process to determine whether a road domain has been learned. If it is a familiar domain, the agent automatically switches to the corresponding policy trained by our method. If not, we collect data from the new target domain to update the domain transfer function and the discriminator.

D. Forming The Imaginary Target Domain

After finishing the first training stage, the source domain and the domain transfer function form a virtual replacement of the target domain, i.e., the imaginary target domain. This imaginary domain implements the dynamics of the target domain without building an actual new environment.

A detailed forming process of the imaginary target domain is shown in Fig. 1(c). The current state s_t, the action intended to use on the target domain $a_t^{\mathcal{T}i}$, and the domain label $C = \mathcal{T}i$ are entered into the dynamics model of the domain transfer function. It predicts the next state $\hat{s}_{t+1}$ on the target domain $\mathcal{T}i$. Then, we input the predicted next state $\hat{s}_{t+1}$, the current state s_t, and the domain label $C = \mathcal{S}$ into the inverse dynamics model of the domain transfer function. Its role is to compute the appropriate current action $\hat{a}_t^{\mathcal{S}}$ applied to the source domain. $\hat{a}_t^{\mathcal{S}}$ can transit the current state s_t to the next state s_{t+1} on the source domain, just like $a_t^{\mathcal{T}i}$ does on the target domain. Afterward, $\hat{a}_t^{\mathcal{S}}$ interacts with the source domain to obtain the next state s_{t+1} and the reward r_t. In this way, a tuple $(s_t, a_t^{\mathcal{T}i}, s_{t+1}, r_t)$ is generated on the imaginary target domain. It is useful for learning the target policy in the second training stage. Note that although the dynamics model and the inverse dynamics model are trained together in the first training stage, they can be used independently.

E. Learning The Target Policy

After getting the imaginary target domain, the second training stage uses it as a training environment to optimize the target policy by reinforcement learning. In order to obtain higher efficiency of policy convergence than previous source augmentation methods [22]–[25], we integrate the advantage of policy guidance in the fine-tuning methods [19]–[21] into our method to accelerate the convergence of the target policy.

The bottom part of Fig. 1(a) shows a complete RL process. A target policy to be optimized (the yellow block) decides a current action according to the current state, $a_t^{\mathcal{T}i} = \pi^{\mathcal{T}i}(s_t)$. $a_t^{\mathcal{T}i}$ is used to interact with the imaginary target domain. From the above explanation of Fig. 1(c), we receive a tuple $(s_t, a_t^{\mathcal{T}i}, s_{t+1}, r_t)$ from each interaction. The tuple is accumulated into the replay buffer. The target policy samples tuples from the replay buffer and is updated by maximizing $E_{P_{\mathcal{T}i}}[\sum_{t=0}^{\infty}\gamma^t r_t]$. The reward function r_t adopts the racing reward proposed in our previous work [38], as shown in (9).

$$r_t = \Delta l_t(\cos\psi_{t+1} - |\sin\psi_{t+1}| - |\Delta dis_{t+1}|), \quad (9)$$

where Δl_t is the distance traveled by the agent car between the time step t and $t+1$, ψ_{t+1} is the angle between the vehicle heading and the track axis at time step $t+1$, Δdis_{t+1} is the distance between the car location and the track axis at time step $t+1$.

Our effort on accelerating the policy convergence is shown in Fig. 1(d). There are two policy architectures to exploit the source-policy knowledge, so that the training process gets help from some useful experiences. The D-FT solution on the left of Fig. 1(d) is inspired by the directly fine-tuning method [20]. The policies of different domains have the same network structure. At the beginning of optimizing a target policy by RL, the network parameters of the source policy are assigned to the target policy. The PNN solution on the right of Fig. 1(d) adopts the progressive net methods [21]. The target policy network has two columns. The first column (the red layers) from the left of PNN represents the learned source policy. When training the target policy, the parameters of the source-policy network in the first column are fixed. Only the second column network and the lateral connections between two columns are learnable. As Fig. 1(d) is a two-layer network example, the second layer of the target policy is shown as

$$h_2^{\mathcal{T}i} = f(W_2^{\mathcal{T}i} h_1^{\mathcal{T}i} + K_{\mathcal{S}_1 \to \mathcal{T}i_2} h_1^{\mathcal{S}}), \tag{10}$$

where $h_1^{\mathcal{T}i}$ is the output of the first layer of the target policy, $h_1^{\mathcal{S}}$ is the output of the first layer of the source policy, $W_2^{\mathcal{T}i}$ is the weight matrix of the second layer of the target policy, $K_{\mathcal{S}_1 \to \mathcal{T}i_2}$ is the lateral connection weight matrix from the first layer of the source policy to the second layer of the target policy, f is a non-linear function.

We choose two state-of-the-art DRL algorithms to optimize the above policy architectures for a high-speed racing task, i.e., Deep Deterministic Policy Gradient (DDPG) [39] and Proximal Policy Optimization (PPO) [40]. DDPG represents the off-line RL. It contains an actor network and a critic network. The actor network $\pi^{\mathcal{T}i}(s_t|\theta^{\pi^{\mathcal{T}i}})$ generates an action $a_t^{\mathcal{T}i}$ for the target policy. The critic network $Q^{\mathcal{T}i}(s_t, a_t^{\mathcal{T}i}|\theta^{Q^{\mathcal{T}i}})$ is the state-action value function to measure the quality of a state-action pair. We apply the above architectures to the actor and critic network, respectively. For the D-FT solution, all network parameters are learnable. For the PNN solution, the weight matrix of the network column of the target policy and the lateral connections are learnable. The weight matrix of the network column of the source policy is fixed. To simplify the description, we use $\theta^{Q^{\mathcal{T}i}}$ and $\theta^{\pi^{\mathcal{T}i}}$ directly in the optimization equations below. They mean optimizing the learnable parts of their network parameters.

The critic network is trained by minimizing TD error,

$$\begin{cases} L(\theta^{Q^{\mathcal{T}i}}) = \frac{1}{\mathrm{N}} \sum\limits_{t=1}^{N} \left(y_t - Q^{\mathcal{T}i}(s_t, a_t^{\mathcal{T}i}|\theta^{Q^{\mathcal{T}i}})\right)^2 \\ y_t = r_t + \gamma \hat{Q}^{\mathcal{T}i}(s_{t+1}, \hat{\pi}^{\mathcal{T}i}(s_{t+1}|\, \theta^{\hat{\pi}^{\mathcal{T}i}})|\theta^{\hat{Q}^{\mathcal{T}i}}) \end{cases}, \tag{11}$$

where N is the batch size of sampling, γ is the discount factor, r_t is the reward function. $\hat{\pi}^{\mathcal{T}i}(s_t|\theta^{\hat{\pi}^{\mathcal{T}i}})$ and $\hat{Q}^{\mathcal{T}i}(s_t, a_t^{\mathcal{T}i}|\theta^{\hat{Q}^{\mathcal{T}i}})$ are two other networks in DDPG. They correspond to the actor network and the critic network. Their role is to copy the network parameters with a delay factor τ for stabilizing learning,

$$\begin{cases} \theta^{\hat{Q}^{\mathcal{T}i}} = \tau\theta^{Q^{\mathcal{T}i}} + (1-\tau)\theta^{\hat{Q}^{\mathcal{T}i}} \\ \theta^{\hat{\pi}^{\mathcal{T}i}} = \tau\theta^{\pi^{\mathcal{T}i}} + (1-\tau)\theta^{\hat{\pi}^{\mathcal{T}i}} \end{cases}. \tag{12}$$

The actor network is updated by

$$\nabla_{\theta^{\pi^{\mathcal{T}i}}} \mathrm{J} \approx \frac{1}{\mathrm{N}} \sum_{t=1}^{N} \nabla_a Q^{\mathcal{T}i}(s, a|\theta^{Q^{\mathcal{T}i}})|_{s=s_t, a=\pi^{\mathcal{T}i}(s_t)} \nabla_{\theta^{\pi^{\mathcal{T}i}}} \pi^{\mathcal{T}i}(s|\theta^{\pi^{\mathcal{T}i}})|_{s=s_t}, \tag{13}$$

where N is the batch size of sampling.

PPO represents the on-line RL. It is also an actor-critic algorithm. Different from DDPG, PPO's critic network is the state-value function, $V^{\mathcal{T}i}(s_t|\theta^{V^{\mathcal{T}i}})$. Its loss function is

$$L(\theta^{V^{\mathcal{T}i}}) = \frac{1}{\mathrm{N}} \sum_{t=1}^{N} (\hat{A}_t)^2, \tag{14}$$

where $\hat{A}_t$ is the estimated advantage function:

$$\hat{A}_t = \sum_{j=0}^{e-1} \gamma^j r_{t+j} + \gamma V^{\mathcal{T}i}(s_{t+e}|\theta^{V^{\mathcal{T}i}}) - V^{\mathcal{T}i}(s_t|\theta^{V^{\mathcal{T}i}}). \tag{15}$$

$t+e$ in (15) is the time step when a terminal state of one trajectory is reached. It varies from state to state.

The actor aims at maximizing (16) below,

$$L(\theta^{\pi^{\mathcal{T}i}}) = \hat{\mathrm{E}}_{\mathrm{t}}[L_t^{CLIP}(\theta^{\pi^{\mathcal{T}i}}) + \xi L_t^{ent}(\theta^{\pi^{\mathcal{T}i}})], \tag{16}$$

where ξ is a weighted coefficient, $L_t^{CLIP}(\theta^{\pi^{\mathcal{T}i}})$ is a clipped surrogate objective,

$$L_t^{CLIP}(\theta^{\pi^{\mathcal{T}i}}) = \min(\frac{\pi_\theta^{\mathcal{T}i}(a_t|s_t)}{\pi_{\theta old}^{\mathcal{T}i}(a_t|s_t)}\hat{A}_t, clip(\frac{\pi_\theta^{\mathcal{T}i}(a_t|s_t)}{\pi_{\theta old}^{\mathcal{T}i}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon)\hat{A}_t). \tag{17}$$

It limits the disparity between the new target policy $\pi_\theta^{\mathcal{T}i}(a_t|s_t)$ and the old target policy $\pi_{\theta old}^{\mathcal{T}i}(a_t|s_t)$ for better convergence. ε is usually 0.1 or 0.2. $L_t^{ent}(\theta^{\pi^{\mathcal{T}i}})$ is an entropy bonus to ensure sufficient exploration.

These two fine-tuning solutions and two DRL algorithms illustrate our method is common rather than the previous works only implemented on a certain RL.

IV. EXPERIMENTS

A. Experimental Setup

Five experiments are designed to evaluate if our method can better bridge the large dynamics gap between different road domains for high-speed racing than prior studies. Since our work aims to be beneficial for the early algorithm development of an autonomous racing task with changeable road domains, we use a 3D realistic racing simulator (TORCS) with different road scenes to conduct experiments. The dynamics parameters describing these road scenes include the friction coefficient, the rolling resistance coefficient, and roughness. These experiments use six TORCS roads, as indicated in Fig. 2. The roads contain three groups of road dynamics (named *asphalt*, *dirt*, *sand*) shown in Table I and four shapes displayed in Fig. 2. Each experiment is represented by a blue arrow and a number. We call these experiments DA①, DA②, DA③, DA④, and DA⑤ for short. The domains of DA① and DA② are two roads that have the same shape but different dynamics. The purpose is to eliminate the influence of other factors on the evaluation of policy adaptation. The domains of DA③, DA④, and DA⑤ have different shapes and different dynamics. They are used to further prove the practicability of our method. The shapes of Road 1 to Road 4 include varied curves. Besides curves, Road 5 and Road 6 contain uphill and downhill sections.

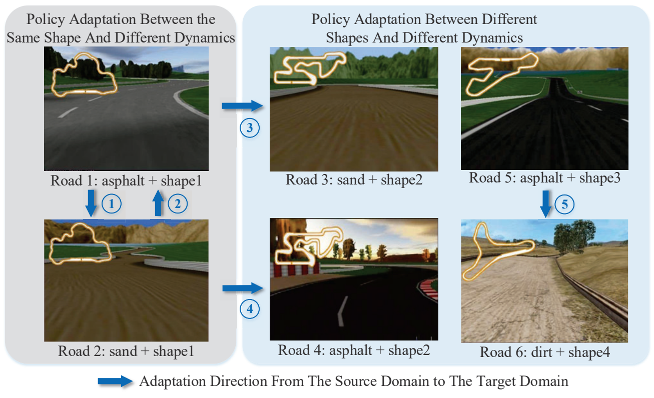
Policy Adaptation Between the Same Shape And Different Dynamics
Policy Adaptation Between Different Shapes And Different Dynamics
Road 1: asphalt + shape1
Road 2: sand + shape1
Road 3: sand + shape2
Road 4: asphalt + shape2
Road 5: asphalt + shape3
Road 6: dirt + shape4
Adaptation Direction From The Source Domain to The Target Domain

Fig. 2. The shapes and dynamics of 6 roads for performance testing.

TABLE I
DYNAMICS PARAMETERS OF ROAD DOMAINS

Road Domains	Friction Coefficient	Rolling Resistance Coefficient	Roughness	Roughness Wavelength
asphalt	1.2	0.001	0	1.0
dirt	0.85	0.005	0.02	30.0
sand	0.9	0.006	0.04	8.0

*1) **Baselines**:* We choose the following baselines based on the condition that they try to get a target policy with limited source domain resources.

- **S-policy**: A policy is trained to handle the dynamics of a source domain, i.e., a source-domain policy.
- **(Inv)Dyn** [15]: It is a domain-invariant feature method. First, a dynamics model of a source domain and an inverse dynamics model of a target domain are learned by regression. They are used as an action adaptation function. Then, it assumes the state sequence of the source policy is valid for the target domain. There is no extra policy learning process in this method. The target policy is obtained using the source policy and the action adaptation function. Specifically, the dynamics model predicts the state sequence based on the source policy. The inverse dynamics model outputs the target action to reach the same state on the target domain.
- **GAT** [23]: It is a representative source augmentation method. First, it learns a mapping function by regression to bridge the dynamics gap between two domains. The mapping function consists of a dynamics model of a target domain and an inverse dynamics model of a source domain. Then, an action generated from the target policy can be optimized from scratch by DRL on the source domain with the help of the mapping function.

*2) **Two Types Of Our Method**:* Because we use two solutions (i.e., D-FT and PNN) to accelerate the convergence of the target policy in the second training stage of DAARL, we named it **DAARL-D** and **DAARL-P**.

For the sake of fair comparison, different algorithms and training techniques have the same network architecture for dynamics, inverse dynamics, and policy networks, respectively.

*3) **Ablation Settings**:* We set two configurations for the ablation study to check how much each part of our method contributes.

- **Stage1**: DAARL without the help of fine-tuning in the second training stage, that is, policy learning from scratch. Its goal is to test the effect of adversarial learning on capturing dynamics.
- **Stage2**: DAARL without the help of adversarial learning in the first training stage. Its goal is to test the effect of merging the fine-tuning technique into policy training. It can be subdivided into **Stage2-D** and **Stage2-P**.

*4) **Evaluation Metrics**:* The following metrics are adopted to evaluate these methods.

- **Avg. Step-Reward**: the average reward per decision during policy training, the higher the better.
- **Success rate**: the percentage of the number of completing the task in 100 testing episodes, the more the better.
- **Avg. vel**: the average speed of success testing episodes, the faster the better.
- **Avg. ang**: the average angle of success testing episodes. The smaller angle, the better stability the policy has.
- **No. of INV-DYN**: the number of inverse dynamics models and dynamics models learned in all experiments. The fewer number of models, the better scalability the method has.
- **Precision & Recall**: two classical classification evaluation indexes [41] used to test our discriminator, the higher the better. Precision measures the probability of trajectories classified as positive that are truly positive. Recall measures the probability of positive trajectories that are correctly labeled.

*5) **Implementation Details**:* For all experiments, the state s_t at time step t is a 29-dimension vector from multiple sensors consisting of an angle between the agent direction and the track axis, three speeds along the longitudinal, transverse, and Z axis of the car, respectively, nineteen distances between the agent and the track edge, one distance between the agent and the track axis, four wheel rotation speeds, and one engine rotation speed. The action is a two-element vector [steering; acceleration] $\in [-1,\ 1]^2$ where steering and acceleration have continuous values. The dynamics model is a two-layer LSTM with 600 units. The inverse dynamics model has two fully connected (FC) hidden layers with 256 units. The discriminator has the same number of hidden layers as the inverse dynamics model. The k of τ_{-k} is 4. The actor and critic network of DDPG have two FC hidden layers of (100, 100) and (500, 500) units, respectively. The actor and critic network of PPO both have three FC hidden layers of (400, 300, 300) units. All FC hidden layers use ReLU activations. All optimizers are Adam. We set both the max number of episodes and the maximum number of time steps per episode to 2,000. All experiments are trained on an NVIDIA GTX 1080Ti GPU.

B. Comparison with baselines

Table II shows the results of two solutions of our method (i.e., DAARL-D, DAARL-P) and baselines on five experiments

TABLE II
PERFORMANCE COMPARISON BETWEEN DAARL AND BASELINES BASED ON DDPG [32] AND PPO [35] ON ALL EXPERIMENTS

Methods	**DA①**						**DA②**						**DA③**						**DA④**						**DA⑤**					
	Avg.vel (km/h)		**Avg.ang (degree)**		**Success rate(%)**		**Avg.vel (km/h)**		**Avg.ang (degree)**		**Success rate(%)**		**Avg.vel (km/h)**		**Avg.ang (degree)**		**Success rate(%)**		**Avg.vel (km/h)**		**Avg.ang (degree)**		**Success rate(%)**		**Avg.vel (km/h)**		**Avg.ang (degree)**		**Success rate(%)**	
	DDPG	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***
S-policy	N/A	N/A	N/A	N/A	0	0	85.83	87.00	4.66	4.63	60	61	N/A	N/A	N/A	N/A	0	0	86.00	88.68	4.90	4.26	50	42	N/A	N/A	N/A	N/A	0	0
(Inv)Dyn	83.13	82.51	6.01	6.12	62	59	86.13	87.91	4.60	4.41	70	77	83.45	80.79	5.71	4.34	50	48	86.29	84.24	4.64	4.02	68	56	49.70	43.71	4.15	4.21	38	34
GAT	85.24	84.04	4.29	5.31	70	76	93.73	92.51	4.61	3.92	80	82	87.05	85.39	5.44	4.97	64	64	93.85	92.66	4.55	5.44	80	76	63.92	58.12	3.95	3.82	50	58
DAARL-D	**97.54**	**95.08**	**3.91**	**4.44**	**87**	**84**	104.75	104.07	4.38	3.70	86	**88**	**96.84**	**94.80**	**4.89**	**4.07**	**77**	**74**	100.52	99.59	4.41	4.03	**88**	**90**	**72.16**	**70.86**	**3.74**	**3.21**	70	74
DAARL-P	93.23	91.05	3.95	4.71	84	82	**106.76**	**107.18**	**4.17**	**3.63**	**90**	**88**	92.11	93.70	4.95	4.52	72	70	**103.58**	**101.21**	**4.33**	**3.96**	85	**90**	66.76	64.50	4.15	3.45	**76**	**78**

(DA① to DA⑤) from different aspects. First, our method outperforms baselines whether these algorithms are implemented based on an off-line RL representative (DDPG) or an on-line RL representative (PPO). Our method can be applied flexibly with different RL algorithms. Second, we observe that our method works best in all experiments. Road domains of DA① and DA② have the same shape but different dynamics. They demonstrate the policy adaptability of different approaches without other road influencing factors. Road domains of DA③, DA④, and DA⑤ have different shapes and different dynamics. They illustrate that DAARL is indeed a practical policy adaptation method rather than being limited to domains with the same shape. It should be mentioned that this research focuses on adapting a driving policy to road domains with the dynamics gap. The choice of the source domain needs to cover the shape features of the target domain. For example, in the DA③ experiment, the source domain is an asphalt road that includes some bend parts. It can create an imaginary road with the shape of the source road and sand dynamics of the target domain. So the policy trained on this imaginary road has the ability to drive through different bends. Similarly, the source domain of DA⑤ includes uphill and downhill parts to learn a useful policy for the target domain with slopes.

Third, we analyze performance comparisons of methods in Table II. S-policy gets the lowest success rate in all experiments. It demonstrates the necessity of policy adaptation between road domains with different dynamics. Specifically, S-policy's success rates in DA② and DA④ perform better than S-policy's success rates in DA①, DA③, and DA⑤. It is because the policy trained on an asphalt road converges to a faster and more aggressive point than the policy trained on a sand road or a dirt road, where an asphalt road has a larger friction coefficient and lower roughness than the other two domains. The aggressive policy always makes the vehicle lose control on a rougher road. (Inv)Dyn achieves a higher success rate than S-policy in all experiments. It is because (Inv)Dyn makes the source policy apply to the target domain with the help of the action adaptation function. But the driving performance is conservative due to the idea of trading off between different domains. Different from the above methods, GAT and our method pursue an optimal target policy. Their results in Table II are faster and more stable than (Inv)Dyn. What's more, DAARL-D and DAARL-P, as two implementations of our method, achieve the highest average velocity, the best stability, and the highest success rate. There are two reasons for these improvements. One is that our method combines the adversarial loss, the domain classification loss, and the reconstruction loss to train the domain transfer function. It enhances the accuracy of building an imaginary target domain for later policy training. The other is that our method leverages the knowledge of the source-domain policy to increase the efficiency of policy convergence.

TABLE III
DOMAIN SCALABILITY COMPARISON BETWEEN DAARL AND BASELINES FROM DA① TO DA⑤

Methods	**S-policy**	**(Inv)Dyn**	**GAT**	**DAARL**
No. of INV-DYN	N/A	8	8	**2**

Fourth, compared the results of DAARL-P with DAARL-D, we observe that DAARL-D outperforms DAARL-P in DA①, DA③, and DA⑤, whereas DAARL-P performs better than DAARL-D in DA② and DA④. It illustrates an optimum solution is different in different directions of policy adaptation. An aggressive source policy plays a positive guiding role in the early part of the convergence of the target policy, but a negative role in the late part of convergence. Since the guiding role of the source policy in DAARL-P is played all the time, DAARL-P is suitable for transferring from a source domain where we can learn a conservative policy. However, the driving guidance of DAARL-D primarily affects the beginning of policy training, so it performs better on policy adaptation from an aggressive source policy than DAARL-P.

Table III shows a comparison of domain scalability between our method and baselines by the No. of INV-DYN metric. Baselines have poorer scalability than our method, except that S-policy does not have a processing step for bridging the dynamics gap. Specifically, (Inv)Dyn needs to learn three dynamics models (for Road 1, Road 2, Road 5) and five inverse dynamics models (for Road 1, Road 2, Road 3, Road 4, Road 6) from DA① to DA⑤. GAT needs to learn five dynamics models (for Road 1, Road 2, Road 3, Road 4, Road 6) and three inverse dynamics models (for Road 1, Road 2, Road 5). Unlike them, DAARL learns only one dynamics model and one inverse dynamics model to cover all road domains.

Additionally, as we described in the last paragraph of section III-C, the classification ability of the discriminator in our method is also useful during the execution process. It is used

TABLE IV
DOMAIN CLASSIFICATION CAPABILITY OF OUR DISCRIMINATOR

Road Domains	**asphalt**	**sand**	**dirt**
precision(%)	96.92	95.69	98.41
recall(%)	98.43	95.01	96.78

TABLE V
ABLATION STUDY BASED ON DDPG AND PPO ON ROADS WITH THE SAME SHAPE AND DIFFERENT DYNAMICS

Methods	**DA①**						**DA②**					
	Avg.vel (km/h)		**Avg.ang (degree)**		**Success rate(%)**		**Avg.vel (km/h)**		**Avg.ang (degree)**		**Success rate(%)**	
	DDPG	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***	***DDPG***	***PPO***
Stage1	90.28	89.21	4.29	5.01	81	80	95.19	93.49	4.55	3.71	85	84
Stage2-D	88.41	88.44	4.25	5.16	78	78	97.04	95.11	4.60	3.90	80	81
Stage2-P	86.74	87.05	4.21	5.48	74	76	99.65	97.95	4.73	3.85	81	82

to determine whether the dynamics of the road ahead can be matched with a policy we have learned. We check the classification accuracy in Table IV by the classical metrics (i.e., precision and recall). The result shows our discriminator is reliable as an indicator of domain switching.

C. Ablation studies

Table V reports the contribution of each training stage of our method in two experiments with the same road shape and different dynamics (i.e., DA① and DA②). Compared with GAT in Table II, Stage1 in Table V achieves higher success rates and better driving performance in both experiments. It indicates that the domain transfer function of our method can model the change of dynamics better by introducing adversarial learning. This improvement helps us create a more reliable imaginary target domain for the second training stage. As a result, our method can bridge a large dynamics gap. Stage2-D and Stage2-P take advantage of two fine-tuning methods to help the target policy converge better in the second training stage of our method. They obtain faster and more stable target policies in both experiments. In particular, Stage2-D in DA① and Stage2-P in DA② perform best in the comparison of Stage2-D, Stage2-P, and GAT in Table II, respectively.

Besides, Fig. 3 demonstrates the average rewards per decision of Stage1, DAARL-D, and DAARL-P during the target policy learning in DA① and DA②. The shapes of the reward curves of Stage1 converge slower and lower than DAARL-D and DAARL-P in all experiments. The confidence intervals of Stage1 are wider than DAARL-D and DAARL-P. These facts indicate that incorporating the fine-tuning technique in the second training stage of our method not only makes policy converge better, but also accelerates the convergence. In other words, it improves the convergence efficiency. Meanwhile, whether based on DDPG or PPO, the reward curve of DAARL-D is higher than DAARL-P in DA① and lower than DAARL-P in DA②. It corresponds to the fourth analysis in section IV-B.

V. CONCLUSION

In this work, we have presented DAARL, a two-stage policy adaptation algorithm based on adversarial learning and reinforcement learning, to bridge the large dynamics gap between different domains for a high-speed racing challenge. First, we learn a domain transfer function by adversarial learning. One domain transfer function and one source domain can create different imaginary target domains with large dynamics gaps. Then, a target policy is trained on the corresponding imaginary domain. During policy training, we use the guidance of the source-domain policy to assist the policy to converge faster and better. This solution is useful to accelerate the early algorithm development of complex racing tasks, because the algorithm implementation does not rely on experts to build an actual new training environment. The algorithm implementation and the build of an actual environment for later performance testing can run in parallel. We have evaluated it on the racing simulator TORCS. The results demonstrated that DAARL converged effectively to a faster and more stable policy than baselines.

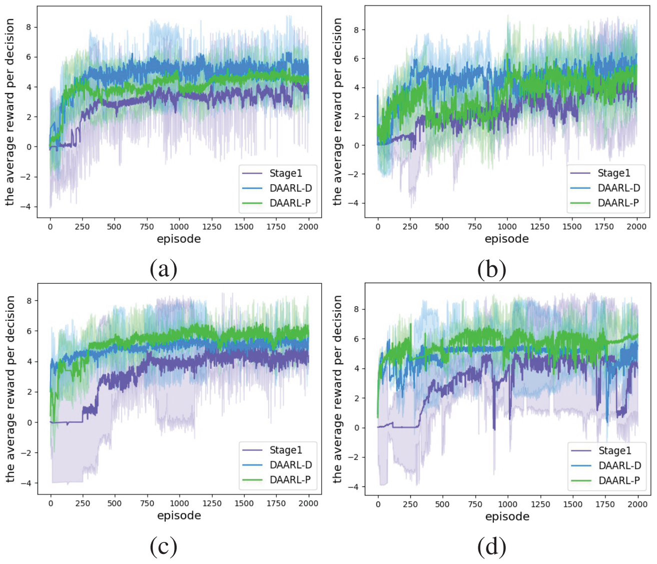

Fig. 3. The average reward per decision of Stage1, DAARL-D and DAARL-P (a): DDPG-based policy training in DA①, (b): PPO-based policy training in DA①, (c): DDPG-based policy training in DA②, (d): PPO-based policy training in DA②

REFERENCES

[1] L. Hermansdorfer, J. Betz, and M. Lienkamp, "Benchmarking of a software stack for autonomous racing against a professional human race driver," in *Proc. International Conference on Ecological Vehicles and Renewable Energies (EVER)*, Monte-Carlo, Monaco, 2020, pp. 1-8.
[2] G. Hartmann, Z. Shiller, and A. Azaria, "Autonomous Head-to-Head Racing in the Indy Autonomous Challenge Simulation Race," *arXiv preprint arXiv:2109.05455*, 2021.
[3] M. O'Kelly, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, and R. Mangharam, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567*, 2019.
[4] S. Koppula, "Learning a cnn-based end-to-end controller for a formula sae racecar," *arXiv preprint arXiv:1708.02215*, 2017.
[5] Fédération Internationale de l'Automobile, "History of FIA World Rally Championship," WRC Promoter GMbH. Paris, France. [Online]. Available: https://www.wrc.com/en/more/wrc-history/wrc-present/
[6] J. Culley, S. Garlick, E. G. Esteller, P. Georgiev, I. Fursa, I. V. Sluis et al., "System design for a driverless autonomous racing vehicle," in *Proc. International Symposium on Communication Systems, Networks*

and Digital Signal Processing (CSNDSP), Porto, Portugal, 2020, pp. 1-6.

[7] Richard S. Sutton and Andrew G. Barto, *Reinforcement learning: An Introduction*, 2nd ed., Cambridge, MA, USA: MIT Press, 2018, pp. 1-2.

[8] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot and F. Nashashibi, “End-to-end race driving with deep reinforcement learning,” in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Australia, 2018, pp. 2070-2075.

[9] Y. Zhu and D. Zhao, “Driving Control with Deep and Reinforcement Learning in The Open Racing car Simulator,” in *Proc. International Conference on Neural Information Processing (ICONIP)*, Cambodia, 2018, pp. 326-334.

[10] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, “sim-to-real transfer of robotic control with dynamics randomization,” in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Australia, 2018, pp. 3803-3810.

[11] P. Cai, X. Mei, L. Tai, Y. Sun, and M. Liu, “High-speed autonomous drifting with deep reinforcement learning,” *IEEE Robotics and Automation Letters (RA-L)*, vol.5, no.2, pp. 1247-1254, 2020.

[12] W. Yu, J. Tan, C. K. Liu, and G. Turk, “Preparing for the Unknown: Learning a Universal Policy with Online System Identification,” presented at *Robotics: Science and Systems (RSS)*, Massachusetts, USA, 2017.

[13] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac et al., “Closing the sim-to-real loop: Adapting simulation randomization with real world experience,” in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Canada, 2019, pp. 8973-8979.

[14] F. Sadeghi and S. Levine, “CAD2RL: Real single-image flight without a single real image,” presented at *Robotics: Science and Systems (RSS)*, Massachusetts, USA, 2017.

[15] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, “Transfer from simulation to real world through learning deep inverse dynamics model,” *arXiv preprint arXiv:1610.03518*, 2016.

[16] A. Gupta, C. Devin, Y. X. Liu, P. Abbeel, and S. Levine, “Learning invariant feature spaces to transfer skills with reinforcement learning,” presented at *International Conference on Learning Representations (ICLR)*, Toulon, France, 2017.

[17] Z. Xu, C. Tang and M. Tomizuka, “Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control,” in *Proc. International Conference on Intelligent Transportation Systems (ITSC)*, Maui, Hawaii, USA, 2018, pp. 2865-2871.

[18] Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek, “Adapting learned robotics behaviours through policy adjustment,” in *Proc. International Conference on Robotics and Automation (ICRA)*, Singapore, 2017, pp. 5837-5843.

[19] D. Isele and A. Cosgun, “Transferring autonomous driving knowledge on simulated and real intersections,” presented at *the Lifelong Learning Workshop of International Conference on Machine Learning (ICML Workshop)*, Sydney, Australia, 2017.

[20] S. Nageshrao, H. E. Tseng, and D. Filev, “Autonomous Highway Driving using Deep Reinforcement Learning,” in *Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC)*, Bari, Italy, 2019, pp. 2326-2331.

[21] A. A. Rusu, M. Večerłk, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, “Sim-to-real robot learning from pixels with progressive nets,” in *Proc. Conference on Robot Learning (CoRL)*, Mountian View, USA, 2017, pp. 262-270.

[22] F. Golemo, A. A. Taiga, P. Y. Oudeyer, and A. Courville, “Sim-to-real transfer with neural-augmented robot simulation,” in *Proc. Conference on Robot Learning (CoRL)*, Zurich, Switzerland, 2018, pp. 817-828.

[23] J. P. Hanna and P. Stone, “Grounded Action Transformation for Robot Learning in Simulation,” in *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, San Francisco, California, USA, 2017, pp. 3834-3840.

[24] H. Karnan, S. Desai, J. P. Hanna, G. Warnell, and P. Stone, “Reinforced grounded action transformation for sim-to-real transfer,” presented at *IEEE/RSJ International Conference on Interlligent Robots and Systems (IROS)*, Las Vegas, NV, USA, 2020.

[25] J. P. Hanna, S. Desai, H. Karnan, G. Warnell, and P. Stone, “Grounded action transformation for sim-to-real reinforcement learning,” *Machine Learning*, vol.110, pp. 2469-2499, 2021.

[26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair et al., “Generative adversarial nets,” in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Montreal, Canada, 2014, pp. 2672-2680.

[27] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Simulated car racing championship: Competition software manual,” *arXiv preprint arXiv:1304.1672*, 2013.

[28] X. Pan, Y. You, Z. Wang, and C. Lu, “Virtual to Real Reinforcement Learning for Autonomous Driving,” presented at *British Machine Vision Conference (BMVC)*, London, UK, 2017.

[29] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, “RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, virtual, 2020, pp. 11157-11166.

[30] J. Liang, S. Saxena, and O. Kroemer, “Learning Active Task-Oriented Exploration Policies for Bridging the Sim-to-Real Gap,” presented at *Robotics: Science and Systems (RSS)*, 2020.

[31] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu et al., “Progressive neural networks,” *arXiv preprint arXiv:1606.04671*, 2016.

[32] Y. Jiang, T. Zhang, D. Ho, Y. Bai, C. K. Liu, S. Levine et al., “SimGAN: Hybrid Simulator Identification for Domain Adaptation via Adversarial Reinforcement learning,” in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Xi’an, China, 2021, pp. 2884-2890.

[33] J. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks,” in *Porc. IEEE International Conference on Computer Vision (ICCV)*, Venice, 2017, pp. 2242-2251.

[34] X. Huang, M. Liu, S. Belongie, and J. Kautz, “Multimodal Unsupervised Image-to-Image Translation,” in *Proc. European Conference on Computer Vision (ECCV)*, Munich, Germany, 2018, pp. 179-196.

[35] Y. Choi, M. Choi, M. Kim, J. Ha, S. Kim, and J. Choo, “StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, USA, 2018, pp. 8789-8797.

[36] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proc. International Conference on Machine Learning (ICML)*, Sydney, Australia, 2017, pp. 214-223.

[37] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved Training of Wasserstein GANs,” in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, USA, 2017, pp. 5767-5777.

[38] J. Niu, Y. Hu, B. Jin, Y. Han, and X. Li “Two-stage Safe Reinforcement Learning for High-Speed Autonomous Racing,” in *Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC)*, Toronto, Canada, 2020, pp. 3934-3941.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa et al., “Continuous control with deep reinforcement learning,” presented at *International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2016.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[41] J. Davis and M. Goadrich, “The relationship between Precision-Recall and ROC curves,” in *Proc. International Conference on Machine Learning (ICML)*, Pittsburgh, Pennsylvania, USA, 2006, pp. 233-240.