IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Efficient Sim-to-Real Transfer in Reinforcement Learning through Domain Randomization and Domain Adaptation

AIDAR SHAKERIMOV, TOHID ALIZADEH, and HUSEYIN ATAKAN VAROL

Department of Robotics, School of Engineering and Digital Sciences, Nazarbayev University, Astana, 010000 Kazakhstan (e-mail: aidar.shakerimov@nu.edu.kz, tohid.alizadeh@nu.edu.kz)

Corresponding author: Huseyin Atakan Varol (e-mail: ahvarol@nu.edu.kz).

ABSTRACT Reinforcement learning has gained significant interest in modern industries for its advancements in tackling challenging control tasks compared to rule-based programs. However, the robustness aspect of this technique is still under development, limiting its widespread adoption. This problem has become more pronounced as users switch to training simulations to reduce costs, resulting in a reality gap that negatively affects real-world performance. One popular method employed to mitigate this problem is randomizing uncertain parameters of the environment during training. Nevertheless, this approach requires expert knowledge to determine the appropriate range of randomization. On the other hand, there is a technique that introduces fine-tuning of agents by adapting their policies to new environments. However, it is challenging to adapt policies when the new environment requires shifting them off their initial distribution. These obstacles limit the practical utilization and popularization of both techniques. Our study proposes a hybrid approach that handles these issues by fine-tuning agents trained with domain randomization through additional real-world training. To assess the efficacy of our approach, we conducted experiments involving a rotary inverted pendulum, augmented with an extra weight not represented in the simulation. Additionally, we employed simulated environments including a cart pole, a simple pendulum, a quadruped, and an ant robot scenario. These environments were given in two distinct versions with mismatching parameters to imitate a gap between training and testing conditions. The results demonstrate that adding as few as twenty to fifty additional real-world training episodes can significantly enhance the performance of agents trained with domain randomization. Moreover, including fifty to two hundred additional episodes can elevate it to a level comparable to those fully trained in the real world. Our study concludes that achieving efficient simulation-to-reality transfer is feasible with domain randomization and relatively small amounts of real-world training.

INDEX TERMS Distributional shift problem, domain adaptation, domain randomization, reality gap, reinforcement learning, robustness, sim-to-real

I. INTRODUCTION

REINFORCEMENT learning (RL) is a branch of machine learning inspired by behavioral psychology's principle of learning via rewards and punishments [1]. The field aims to develop algorithms that learn optimal control policies by using trial-and-error in environments with built-in reward functions that guide the learning process. The policy, which is a function mapping states to actions, is continuously updated based on the rewards observed from these interactions [2]. A policy is optimal if it is built from actions that provide the highest cumulative reward in accordance with Bellman's principle of optimality [3]. RL approximates the optimality of policies via sampled interactions rather than explicitly modeling the environment dynamics. The accuracy of the approximation can improve over time by observing more interactions [1].

RL is especially useful for robotics since it can be directly employed for system control. Indeed, it can be viewed as a variant of adaptive optimal control, with the key difference being that it does not require explicit knowledge of system dynamics [4]. In contrast, traditional optimal control theory requires explicit knowledge of a system's dynamics and con-

straints [5]. This makes RL beneficial when dealing with systems whose dynamics are hard to model, enabling it to discover a near-optimal policy through direct interaction with the environment.

The recent integration of RL with deep learning [10] and actor-critic approaches [11]–[13] have expanded RL's capacity, enabling it to solve problems in complex high-dimensional environments as well as continuous action spaces. This way, RL was used in real-world control applications, including soft-robotic manipulation [14], locomotion [15], and autonomous vehicles [16]. However, despite substantial progress, RL is still in its maturation phase, particularly in terms of robustness [17].

Since learning through trial-and-error in real-world setups can be time-consuming and even damage the setup, simulators for RL training have proven to be beneficial (see Fig. 1). However, RL training in simulators often lacks robustness due to the simulation-to-reality (sim-to-real) gap, i.e., the deviation between the simulation environment and the actual real-world one [18]. Therefore, it is important to enhance the robustness of RL agents against mismatches between training and testing environments.

One way to solve the sim-to-real gap is to utilize more precise simulators during training [18]. While obtaining an exact mathematical representation of the real world is challenging, system identification can be used to select the most suitable model from a set of available mathematical models of a system by observing and analyzing the relationships between inputs and outputs [19]. However, it is nontrivial to identify a nonlinear dynamic model and validate its accuracy [20]. In addition, gathering real-world data might be time-consuming, costly, and error-prone [21]. Furthermore, this may cause overfitting to the specific validation conditions captured through system identification, resulting in poor generalization when encountering varying real-world situations [20], [22].

Another strategy would be increasing the robustness of the RL algorithms to discrepancies between simulations and real-world environments [18]. One of the popular approaches is RL training with adversarial perturbances [23]–[26]. Another approach is domain randomization [21], [22], [27]–[31], which involves training in environments where specific parameters are altered randomly [18]. While training with adversarial perturbations can resolve the effects of specific actuation disturbances, e.g., motor noise or component wear, domain randomization is more related to robustness against parametric and non-parametric uncertainties [32].

Despite its benefits, domain randomization demands expert involvement and multiple iterations for optimal behavior [21]. Specifically, establishing a representative parameter distribution is vital to avoid real values falling outside that range. However, over-randomization may lead to suboptimal policies or nonconvergence [22], [29], [33].

Instead of repeated adjustments and tests of randomization ranges, we aim to fine-tune the policy obtained with domain randomization for real-world conditions by continued online real-world training. To achieve this, we propose using domain adaptation methods. These methods are generally employed in vision-based deep learning and natural language processing, where training data in an available training domain are mixed with a small number of samples from a target domain to generate a common feature space [34], [35]. However, a naive transition from training in a simulated environment to training in the real world has a high probability of a distributional shift problem [36]. This shift is caused by a mismatch between the training and testing data distributions and leads to an inadequate Q-value estimation.

We propose that merging domain adaptation with domain randomization into a hybrid methodology could offset the individual limitations of both strategies. It could equip RL algorithms with the capability to adapt to real-world specifics, thereby addressing the problem of underfitting, prevalent in domain randomization. Conversely, we hypothesize that using a simulation with a sufficient domain randomization range before domain adaptation to real-world conditions would create a greater training data distribution and decrease the effect of the distributional shift problem. Additionally, it will necessitate significantly fewer real-world episodes to shape a generalized policy, rather than overfitting to the discrepancies, thus enhancing overall robustness and adaptability [18], [32].

While efforts to combine domain randomization and domain adaptation have been made for addressing visual sim-to-real gap aspects [29], to the best of our knowledge, our research is the first to tackle such a combination for dynamics parameter mismatches. We specifically focus on cases involving physical parameter variations or initial uncertainties, such as when a system has imprecise component characteristics or must handle unspecified weights. Overall, the contributions of our paper can be listed as:

- A novel approach that combines domain randomization and domain adaptation to address the sim-to-real gap problem in RL.
- Demonstration of both the effectiveness and limitations of the proposed method in enabling a reinforcement learning (RL) agent, initially trained within a simulated environment, to proficiently perform tasks in the real world.
- Guidelines and insights for implementing the proposed approach in real-world applications based on observations from the experiments.

The organization of this paper is as follows: In Section II, we elaborate on the methodology, encompassing RL algorithms, as well as the implementation of an integrated approach to domain randomization and adaptation. Section III provides a detailed description of the experimental setup. In Section IV, we present the experimental results and a discussion of the main findings. Section V concludes the research by presenting our suggestions for future works and summarizing the study.

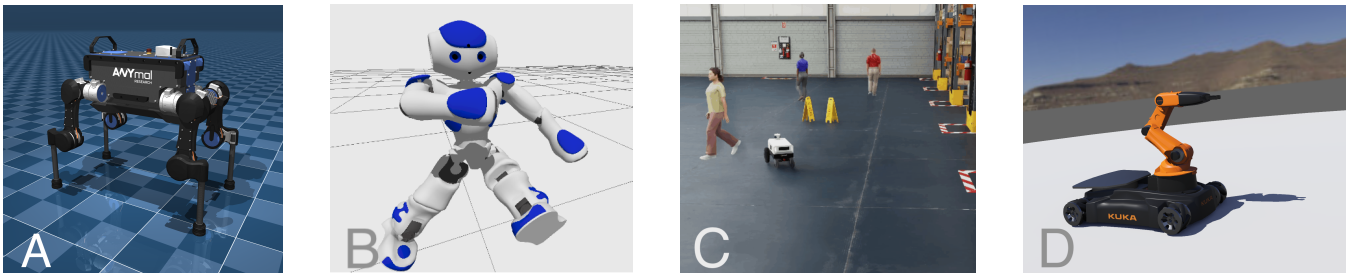


FIGURE 1: Examples of robot modes in different simulation environments: a) The ANYmal quadruped robot (MuJoCo) [6], b) The NAO humanoid robot (Gazebo) [7], c) Human activities within a warehouse (Isaac Sim) [8], d) The YouBot mobile robotic arm (Webots) [9].

II. METHODOLOGY

This section outlines our methodology for training an RL agent with domain randomization and domain adaptation techniques to minimize the performance drop during sim-to-real transfer. The first subsection discusses RL algorithms utilized in our study, starting with the fundamental concepts behind them. We then delve into the specifics of domain randomization and domain adaptation techniques.

A. RL AND DEEP RL

RL algorithms follow the trial-and-error approach to find the best policy of action in given states. The user provides the reward function to indicate whether the conditions obtained after performing a certain action a in a certain state s are good or bad [1]. Here, we describe a classical RL algorithm named Q-learning to provide an understanding of the basic principles behind three deep RL algorithms utilized in this work: Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO). Such a variety of algorithms was used in order to show the generalizability of our approach to the overall deep RL field.

1) Q-learning

In reinforcement learning (RL), the goal is to discover a policy that maximizes cumulative rewards [1]. In the Q-learning algorithm this is achieved through the use of Q-value function $Q(s,a)$ which maps states to actions, predicting potential rewards [37]. Q-values are stored and updated in a table during training based on experiences.

The update mechanism in Q-learning uses a temporal difference error (TD_{error}), calculated as the discrepancy between the observed ($Q_{observed}(s_t,a_t)$) and recorded ($Q_{old}(s_t,a_t)$) Q-values for a state-action pair (see (1)). This error guides the modification of Q-values according to the learning rate α. The updated Q-value combines the immediate reward r_t and future rewards discounted by factor γ, based on the Bellman optimality equation (see (2)).

$$Q_{new}(s_t,a_t) = Q_{old}(s_t,a_t) + \alpha TD_{error}. \tag{1}$$

$$Q_{observed}(s_t,a_t) = r_t + \gamma max(Q(s_{t+1},a)). \tag{2}$$

Q-learning ensures policy optimization through exploration and exploitation [38]. Strategies like epsilon-greedy and the Boltzmann distribution balance these two aspects by varying the action selection process, either randomly or based on the estimated values of actions [39]. These methods help in discovering efficient policies while avoiding suboptimal solutions.

2) Deep Q-Network

Deep Reinforcement Learning (Deep RL) combines the principles of traditional RL with deep learning to approximate the Q-value function through deep neural networks enabling agents to handle complex, high-dimensional input spaces. A pivotal advancement in Deep RL is the Deep Q-Network (DQN), developed by Mnih et al., which demonstrated remarkable success in various Atari video games.

In DQN using deep neural networks allows for efficient handling of environments with high-dimensional state spaces. A network predicts Q-values for state-action pairs ($Q_{predicted}(s_t,a_t)$), with its parameters refined through the back-propagation of a loss function *Loss* based on the discrepancy between predicted and observed Q-values (see (3)).

$$Loss = |Q_{observed}(s_t,a_t) - Q_{predicted}(s_t,a_t)|. \tag{3}$$

Two other key modifications in DQN contribute to its enhanced performance and stability compared to traditional Q-learning. First, to avoid the suboptimal convergence caused by correlations in consecutive transition experiences, DQN employs experience replay. Transition experiences are stored in a buffer and sampled randomly for Q-value updates, allowing for a more uniform experience estimation. Second, stability is further improved by using a separate target network for policy estimation. This target network is a slower-updated duplicate of the main prediction network, ensuring stable learning in dynamic environments.

3) Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a deep RL algorithm presented by Lillicrap et al. [11], which addresses a key limitation of DQN: the inability to handle continuous action spaces. DQN excels in environments with high-dimensional or continuous inputs but fails when the action

space itself is continuous. DDPG overcomes this by utilizing a deterministic policy gradient approach suitable for such environments.

Unlike DQN, where actions are selected from a set of discrete choices, DDPG employs an actor network to continuously derive action values. The updates to this actor network are guided by Q-values from a critic network. The critic network, in turn, evaluates the Q-value of the state-action pairs, incorporating feedback from both the state observed and the action proposed by the actor network.

DDPG enhances stability and performance by employing target networks, similar to DQN. These target networks are slower-updating versions of the actor and critic networks, used for consistent policy evaluation.

Both DQN and DDPG are deterministic, meaning they consistently select what they deem the best action for a given state. This characteristic is crucial for precision and accuracy in RL applications. To prevent early convergence to suboptimal policies, DDPG introduces exploration by adding noise to the actor network's output, effectively exploring the continuous action space.

Lastly, DDPG, like DQN, is an off-policy algorithm. It stores experiences in a replay buffer during environment interaction and uses these experiences for policy updates post-episode, leading to predictable and consistent improvements in policy.

4) Proximal Policy Optimization

Proximal Policy Optimization (PPO) [40] is another RL algorithm based on an actor-critic technique to solve continuous action spaces. However, distinct from DDPG, it adopts a stochastic, on-policy approach, allowing for the consideration of multiple actions with high reward probabilities, rather than selecting a single deterministic action. This stochastic nature helps PPO avoid getting trapped in local reward maxima.

Unlike DQN and DDPG, which use offline policy updates, PPO updates its policy online, immediately following each interaction with the environment. This enables PPO to adapt rapidly to changing conditions and makes it well-suited for environments where optimal policies are uncertain or evolving.

PPO also utilizes advantage values to assess actions, focusing on how much better the actual outcome of an action is compared to the expected outcome. This approach helps in learning strategies that might yield short-term negative rewards but are beneficial in the long run.

A defining feature of PPO is its use of a proximal policy with a 'clipping' mechanism to constrain updates to the policy. This clipping limits the extent of change between the old and new policy, ensuring gradual and stable behavioral changes during the training process. Such constraints are crucial for developing robust and reliable policies in dynamic learning scenarios.

B. DOMAIN RANDOMIZATION

Domain randomization is a technique for sim-to-real transfer drawing from the theory of robust control under uncertainty which introduces randomized disturbances to certain dynamics or measurement parameters [41]. Examples of dynamics disturbances might be mismatches of dynamics parameters, actuation, or dynamics modeling errors. Measurement disturbances involve sensor noises and variations of lighting, textures, colors, or other visual properties of objects and scenes. Dynamic disturbances directly affect state transitions, whereas measurement disturbances have an effect on the observation of states. During training, a learning algorithm updates the control policy towards obtaining maximal cumulative reward under every encountered disturbance scenario [32].

In our study, we focus on dynamics disturbances, specifically the cases with mismatches of dynamics parameters such as mass or lengths of system components. We perform training with domain randomization such that in every new episode of training a parameter will have a new random value inside a specified range. Although our assumption that the target configuration of the parameters lies in a range of +/- 20 percent might be too loose, we utilize further domain adaptation to avoid underfitting.

C. DOMAIN ADAPTATION

The domain adaptation is a transfer learning technique in machine learning that aims to direct trained agents to create a single feature space for both the training environment and the target environment by adding some amount of target data into the training dataset [34]. There are several approaches to domain adaptation. In our study, we focus on fine-tuning pre-trained models through continual training on the target domain. It requires minimal adjustments of model architecture, yet studies report the efficiency of the approach [45], [46]. However, the direct transition from training in simulation to training in the real world can cause a distributional shift problem, i.e. a mismatch of simulated and real data distribution leading to a wrong estimation of the Q-values due to the strong influence of the simulated experience.

D. HYBRID APPROACH

In our hybrid approach, we incorporate domain adaptation by a small number of additional episodes in the real world after an agent was initially trained in the randomized environment (see Algorithm 1). In this context, M_{sim} and M_{real} represent the number of training episodes in the simulation and the real world, respectively. We set $M_{sim} \gg M_{real}$ to ensure the safety of the real system. Here, T denotes the number of steps in an episode, while n indicates the number of steps required for a delayed update of the target networks. The simplified process diagram of our combined approach can be seen in Fig. 3.

The distributional shift problem is solved by increasing the training data distribution via randomization. At the same time, the consecutive fine-tuning allows the sharpening of the generalized policy towards a specific environment.

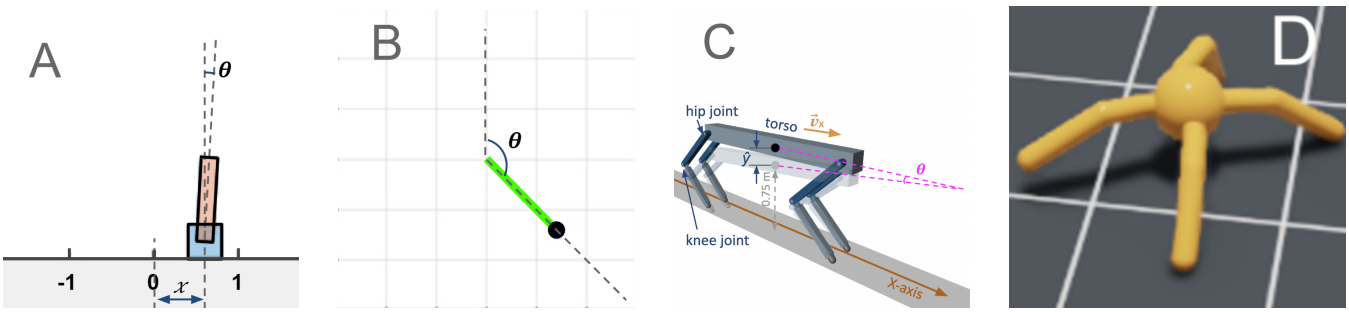


FIGURE 2: Visualisation of Cart Pole [42] (a), Simple Pendulum [43] (b), and Quadruped [44] (c) systems in Matlab, and (d) Ant robot system in Isaac Sim used for the experiments described in the subsection III-A.

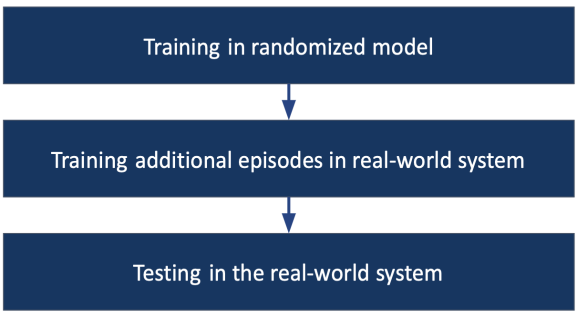


FIGURE 3: The pipeline of our training and testing processes.

III. ROBUST RL EXPERIMENTS

The section details the procedures of our experiments. The first subsection describes our approach to bridging the simulation-to-simulation (sim-to-sim) gap, where RL is tasked to perform well in a simulation that imitates a real system while being trained on another simulated version with mismatching parameters. Throughout the first subsection, the version of the environment used for training is called the nominal model, whereas the term "real plant" is used to refer to the testing version imitating reality. The latter subsection reports our real-world experiments that verify our approach to address a true simulation-to-reality gap. There, the term "real plant" implies the real-world rotary inverted pendulum having mismatching parameters with its simulated nominal model used for training.

To assess the generalizability of our approach, we employ a diverse range of environments and leverage three distinct deep reinforcement learning algorithms. These environments have varying levels of complexity, encompassing both commonly utilized benchmark tasks like the cart pole and rotary inverted pendulum, as well as more intricate scenarios, such as a quadruped robot and an ant robot. Additionally, our investigation covers the spectrum of action space types, dealing with discrete (as exemplified by the cart pole) and continuous (as demonstrated by the simple pendulum, rotary inverted pendulum, quadruped, and ant robot) action space settings. To address these distinct action space requirements, we utilize the DQN algorithm for discrete action spaces and the DDPG and PPO algorithms for continuous action spaces. Finally, we utilize Matlab with Simulink and Isaac Sim to display the independence of our results from the utilized simulator. This comprehensive approach enables us to thoroughly evaluate the adaptability and robustness of our proposed method across a wide spectrum of environments and control paradigms.

A. SIM-TO-SIM TRANSFER

Deep RL agents were trained in simulations representing training models (nominal models and randomized models) and tested in simulations with another configuration representing a "real plant". We used three systems simulated in the Matlab/Simulink environment: a cart pole with uncertain cart mass, a simple pendulum with uncertain pendulum mass, and a quadruped with uncertain leg lengths (see Fig 2-a, 2-b, 2-c). Additionally, we simulated a system in the Isaac Sim environment, specifically an ant robot with uncertain feet masses (see Fig. 2).

Certain agents underwent training using our approach, employing domain randomization within a simulation, followed by consecutive domain adaptation using a simulation with real-world configuration. The number of additional "real-world" episodes varied in the range of 20 to 500, aiming to identify an optimal value that balances efficiency and affordability.

At the same time, other agents were trained on static nominal models with a mismatch of parameters with the real plant. These agents were expected to fail on "real plants".

1) Cart Pole Experiments

The cart pole environment requires an agent to balance the pole on a cart in the upright position. The nominal model system has a mass of 4.0 kg, while the "real plant" has a mass equal to 4.5 kg. To bridge the sim-to-sim gap, an agent was first trained in a randomized model with mass varying within a range of 3.2 kg to 4.8 kg and then was further trained for 20, 100, 200, and 500 episodes on the "real plant" simulation with mass equal to 4.5 kg.

An agent obtained +1 reward per step for keeping the pole angle θ between -12 and 12 degrees, and -10 for falling beyond this range. The optimal score for an episode with 500 steps was 500 implying that the pendulum was upright during the whole episode. The RL model took the position of the

Algorithm 1 RL training with domain randomization and real-world fine tuning

Initialize randomization range R
Initialize replay buffer D
Initialize networks:
  Initialize Q-value network Q with random weights θ_Q
  and/or (based on the choice of RL algorithm)
  Initialize policy network μ with random weights θ_μ
Initialize target networks:
  Initialize Q' with weights $\theta'_Q = \theta_Q$
  and/or (based on the choice of RL algorithm)
  Initialize μ' with weights $\theta'_\mu = \theta_\mu$
{Training in a simulation with domain randomization}
for episode = 1, M_{sim} **do**
  Set parameter p of environment ε to a random value in R
  Reset the environment ε
  Receive initial observation s_1
  for t = 1, T/n **do**
    for t = 1, n **do**
      Select action a_t
      (based on RL algorithm and exploration strategy)
      $r_t, s_{t+1} \leftarrow \varepsilon(s_t, a_t, p)$
      Store transition $(s_{t+1}, a_t, r_t, s_{t+1})$ in D
      $s_{t+1} \leftarrow s_{t+1}$
      Update parameters θ_Q, and/or θ_μ
    end for
    Update target parameters θ'_Q and/or θ'_μ
  end for
end for
{Fine-tuning on real system}
$p \leftarrow p_{real}$
for episode = 1, M_{real} **do**
  Reset the environment ε
  Receive initial observation s_1
  for t = 1, T/n **do**
    for t = 1, n **do**
      Select action a_t
      (based on RL algorithm and exploration strategy)
      $r_t, s_{t+1} \leftarrow \varepsilon(s_t, a_t, p)$
      Store transition $(s_{t+1}, a_t, r_t, s_{t+1})$ in D
      $s_{t+1} \leftarrow s_{t+1}$
      Update parameters θ_Q, and/or θ_μ
    end for
    Update target parameters θ'_Q and/or θ'_μ
  end for

cart x, the velocity of the cart $\dot{x}$, the angle of the pole θ, and the angular velocity of the pole $\dot{\theta}$ as an input state, and performed either a discrete move to the right or a discrete move to the left [42]. This environment is visualized in Fig. 2a.

We applied the DQN algorithm for controlling the cart pole because of the discrete action space. RL agent observed the system state and took certain control actions. The actions were selected based on the epsilon-greedy strategy. After performing an action, the agent observed a new state, reward, and termination status. The transition was then stored in an experience replay buffer, from which random samples were taken to update the prediction network. Periodically, the target network parameters were copied from the prediction network.

2) Simple Pendulum Experiments

In the simple pendulum environment, an agent is required to swing up and balance the pendulum in an upward position by directly applying a torque between -2 and 2 Nm. The nominal model has a 0.875 kg pendulum, whereas the pendulum mass of the "real plant" is 1 kg. In our experiments, the randomized model varied the mass between 0.8 and 1.2 kg.

We utilized the default reward function for a simple pendulum suggested by MathWorks. The agent was penalized for keeping the pendulum angle θ far from an upright position and the angular velocity $\dot{\theta}$ high (see Fig. 2b). It also penalized the agent for excessive effort. The reward function is then formulated as:

$$r = -(\theta_t^2 + 0.1\dot{\theta_t}^2 + 0.001u_{t-1}^2), \tag{4}$$

where θ_t is the displacement angle in relation to the upright position and u_{t-1} is a torque applied in the previous simulation step [43].

Since the action space was continuous, we used DDPG for controlling the pendulum. During a training step, the agent observed the pendulum cosine and sine values with angular velocity as a state and performed an action based on a prediction from the actor network. A small noise was added to the output in order to promote the exploration of new transitions during the training. The agent then recorded the transitions and rewards into the experience replay buffer. Later, random samples of transitions were taken from the replay buffer in order to perform policy updates. The parameters of the actor network were updated in accordance with the Q-value estimated by the critic network, whereas the critic network was updated based on the loss between the estimated Q-value and the target Q-value taken from the target network.

3) Quadruped Robot Experiments

The next experiment addressed the sim-to-sim gap in quadruped locomotion. In the nominal model of the quadruped, all leg lengths were 0.5517 m. In the randomized model, the leg lengths varied between 0.5 and 0.6 m, and in the "real plant" simulation, the leg lengths for each leg were assigned randomly (0.53 m, 0.51 m, 0.52 m, and 0.59 m). The observation space included the vertical and lateral positions of the quadruped robot torso's center of mass, its orientation as a

quaternion, linear velocities in three directions, angular rates for roll, pitch, and yaw, hip and knee joint positions and velocities for each leg, ground contact forces, and joint torques of the previous time step. The agent was provided with eight simultaneous actions for the joints of the quadruped legs. The actions were continuous torques in the range ± 10 Nm [44]. The simulation model of the quadruped robot is visualized in Fig. 2c.

To deal with the continuous action space, we used DDPG once again. However, unlike the simple pendulum, the quadruped should get eight actions to control all actuators on each joint. A comprehensive reward function suggested by MathWorks was employed during training, which encouraged forward movement, avoided early termination, and penalized unwanted states such as deviations from the desired height and orientation or excessive joint torques. This reward function is written as:

$$r_t = v_x + 25\frac{T_s}{T_f} - 50\hat{y}^2 - 20\theta^2 - 0.02\sum_i u_{t-1}^i, \quad (5)$$

where the x-direction velocity of the torso's center of mass is represented by v_x. T_s and T_f correspond to the sample time and the final simulation time of the environment, respectively. The scaled height error, $\hat{y}$, is the torso's center of mass height deviation from the desired height of 0.75 m. The pitch angle of the torso is denoted by θ. The previous time step's action value for joint i is represented by u_{t-1}^i [44].

4) Ant Robot Experiments

Exploring various simulators can enhance the generalizability of our findings and provide a broader perspective on the applicability of our approach. In our study, we opted to leverage Isaac Sim from NVIDIA, which serves as a versatile robotics simulation application and synthetic data-generation tool. Isaac Sim empowers us with the capability to create photorealistic and physically accurate virtual environments, thereby offering a more efficient and effective means to develop, test, and manage AI-based robotic systems.

Within Isaac Sim, there's a library known as "RL games" that offers a diverse range of robot models and tasks for experimentation. For our research, we chose the "Ant" environment (see Fig 1-d, representing a complex 3D robot imitating an insect with four legs. This choice allowed us to apply the same strategy employed in our previous experiments, ensuring consistency and comparability across our findings.

The task of an agent controlling the ant robot is to move the robot forward as far as possible by applying continuous torques in motors located at each joint of the legs. The reality gap in the experiment is implemented by setting the feet masses of the test model imitating a "real plant" to 0.7, 0.6, 0.8, and 0.5 kg, whereas specifying all feet masses in the nominal training model to 0.67 kg. The randomization of the environment is done by varying each foot independently between 0.4 and 1.0 kg.

5) Cart pole trajectories capture

These experiments aim to analyze the learned trajectories of cart pole agents. We focus on capturing the dynamics of pole angle versus cart position. A strict policy will converge to a single trajectory, while a loose policy distribution will result in a wider spread of trajectories. The success criteria include the convergence of trajectories at zero pole angle and staying within the ranges of -0.26 to 0.26 radians for pole angle and -2.5 to 2.5 meters for cart distance. We explore six scenarios: (a) Distribution of optimal behavior for the "real plant," (b) Distribution of optimal behavior for the nominal model, (c) Distribution of optimal behavior for the randomized model, (d) Trajectories of the agent trained on the nominal model on the "real plant," (e) Trajectories of the randomized agent after 200 training episodes on the "real plant," and (f) Trajectories the agent trained with the nominal model after 200 training episodes on the "real plant." Our hypothesis posits that agents trained with the nominal model will exhibit a narrow policy distribution and struggle to adapt to the "real plant" due to distributional shifts. Conversely, agents trained in the randomized environment are expected to have a wider policy distribution and better adaptation to the "real plant."

The parameters of the models are equivalent to those described in the subsection III-A1 "Cart Pole Experiments."

B. SIM-TO-REAL TRANSFER

In order to verify the efficacy of our RL training methodology in the real world, we conducted experiments with a rotary inverted pendulum setup controlled by the DDPG algorithm. There we train an agent on a Simulink model and then test on the real plant (i.e., real-world pendulum system). The following subsections describe our experimental setup, including hardware and software aspects, RL agent training procedures, and details of the experiments.

1) Rotary Inverted Pendulum Experimental Setup

Our setup consisted of a workstation (AMD Ryzen Threadripper 3970x CPU and 128 GB memory) with Ubuntu 20.04.6 LTS operating system, a real-time controller (National Instruments, NI-cRIO), and a voltage amplifier. Utilizing a workstation with a graphics processing unit (NVIDIA GeForce RTX 3070) was sufficient to execute RL training and test in real time. During real-time control, the workstation ran a Matlab/Simulink code for RL algorithm execution, while the controller ran a Labview code to operate the rotary inverted pendulum and acquire measurements. We performed real-time data exchange between Matlab/Simulink and Labview programs running on separate devices through a User Datagram Protocol (UDP) connection [47]. The illustration of the data flow in the setup is presented in Fig. 4.

The rotary inverted pendulum was made by the company Quanser. This setup consisted of a static base, an arm actuated with a servo motor, and a pendulum that freely rotated perpendicular to the arm at which it was pivoted. The angles of the pendulum and the arm were recorded by encoders. Figure 5 shows a schematic diagram of the rotary inverted

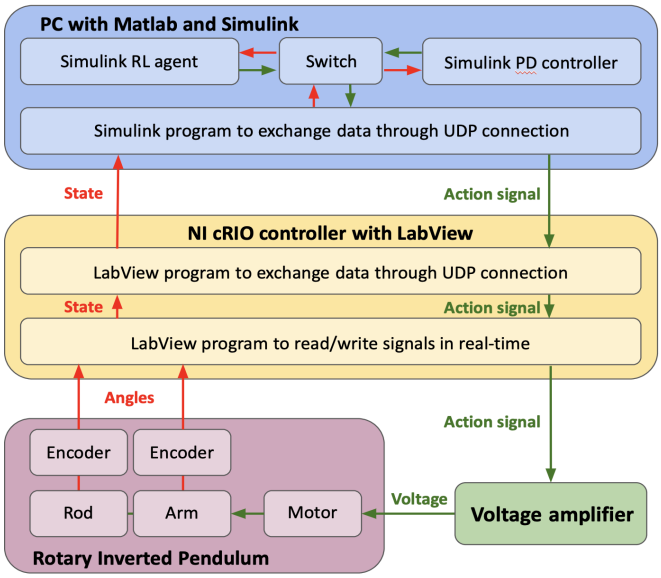


FIGURE 4: Signal flow in the experimental setup.

TABLE I: Nominal parameters of the rotary inverted pendulum model

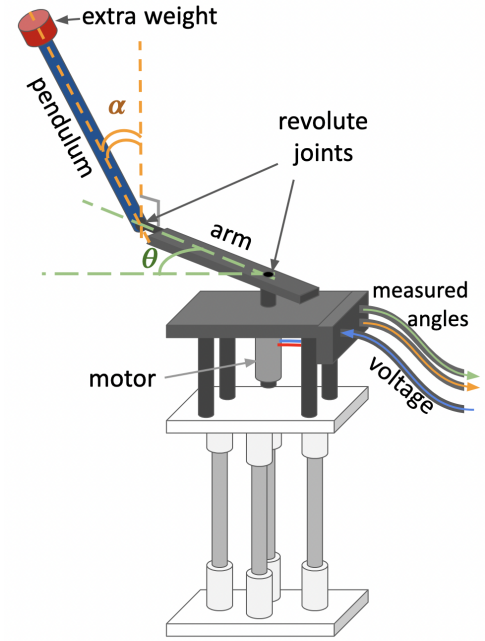
Symbol	Description	Nominal Value
m_r	Pendulum rod mass	0.127 kg
L_p	Total length of pendulum	0.337 m
J_p	Moment of inertia of the pendulum	0.0012 kg·m^2
B_p	Pendulum viscous damping coef.	0.0024 N·m·s/rad
L_r	Arm length	0.216 m
J_r	Moment of inertia of the pendulum	0.001 kg·m^2
B_r	Arm viscous damping coef.	0.0024 N·m·s/rad
R_m	Motor armature resistance	2.6 Ohm
k_m	Motor back-emf	0.0077 V·s/rad
k_t	Motor current-torque const.	0.0077 N·m/A
K_g	Total gear ratio	70
ν_g	Gearbox efficiency	0.9
ν_m	Motor efficiency	0.69

pendulum along with an actual photograph showcasing the physical arrangement of the setup.

2) RL Agent Training

For our experiments, we performed training of RL agents in simulated models of the setup. The first simulated model had a static nominal configuration of the setup. The parameters of the nominal model are summarized in Table I. The values of the parameters are taken from the Quanser manuals [48] and [49].The second case involved a process shown in the Algorithm 1. $M_{sim} = 2000$ episodes, whereas $M_{real} = 50$.

The mass of the pendulum m_p is the mass of the rod m_r and additional mass m_w placed on the end of the rod. The randomized parameter was implemented by randomly varying the additional mass $p = m_w$ in range R between 0 kg to 0.1 kg at each training episode. The equations governing



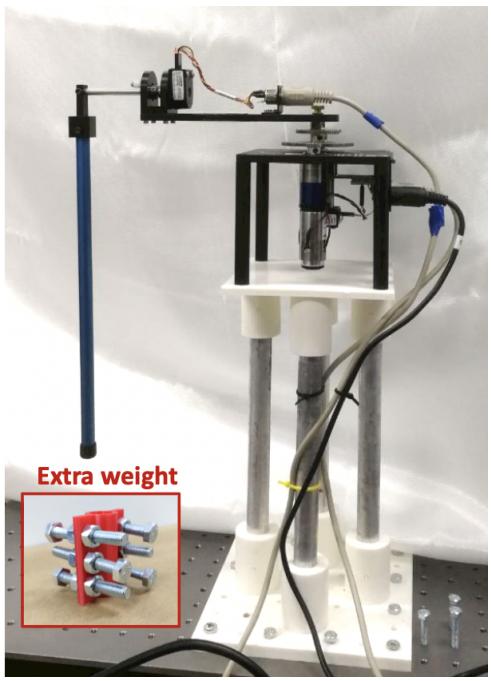


FIGURE 5: Schematics and photograph of the rotary inverted pendulum with the extra weights used in sim-to-real experiments.

the pendulum's dynamics can be expressed as:

$$(m_p L_r^2 + \frac{1}{4} m_p L_p^2 - \frac{1}{4} m_p L_p^2 cos(\alpha)^2 + J_r)\ddot{\theta} - (\frac{1}{2} m_p L_p L_r cos(\alpha))\ddot{\alpha} + (\frac{1}{2} m_p L_p^2 sin(\alpha) cos(\alpha))\dot{\theta}\dot{\alpha} + (\frac{1}{2} m_p L_p L_r sin(\alpha))\ddot{\alpha}^2 = \tau - B_r\dot{\theta}, \quad (6)$$

$$-\frac{1}{2}m_pL_pL_rcos(\alpha)\ddot{\theta}+(\frac{1}{4}m_pL_p^2+J_p)\ddot{\alpha}$$
$$-(\frac{1}{4}m_pL_p^2sin(\alpha))\dot{\theta}^2-(\frac{1}{2}m_pL_pgsin(\alpha))=-B_p\dot{\alpha}, \quad (7)$$

where θ is the angle of the arm and α is the pendulum angle. The torque τ is calculated from the supplied voltage V_m and measured angular velocity of the arm $\dot{\theta}$:

$$\tau=\frac{\nu_gK_g\nu_mk_t(V_m-K_gk_m\dot{\theta})}{R_m}. \quad (8)$$

The RL agent outputs voltage as an action while taking the vector consisting of the arm angle θ_t, pendulum angle α, the angular velocity of the arm $\dot{\theta}_t$, and angular velocity of the pendulum $\dot{\alpha}_t$ at a given time step t as input state $s_t=[\theta_t,\alpha_t,\dot{\theta}_t,\dot{\alpha}_t]$.

Since the equations for motion represent a nonlinear system with continuous action space, we applied the DDPG algorithm for the swing-up task. When the pendulum arm angle α approached zero degrees (vertical upright position), we switched to the linearized balancing model by substituting all $sin(\alpha)$ terms with α and used state-feedback control [50]. Specifically, the state-feedback control loop was activated when the pendulum reached within 0.2 radians of the vertical upright position.

To ensure a successful control delegation, the RL agent must learn to bring the pendulum to this position with minimal velocity, while also executing a swift swing-up maneuver with minimal overall effort. To train the agent in mastering this behavior, we devised the following reward function:

$$r_t=-\alpha_t^2-0.1(\theta_t-\alpha_t)^2-0.1u_{t-1}+F, \quad (9)$$

where $F=100$ was a positive reward for keeping the pendulum angle less than 0.2 radians and the pendulum angular velocity less than 5 rad/s.

The state-feedback control was in the form $u=K\,(x_d-x)$ where u is the control action, $x_d=\mathbf{0}$ is the reference state vector, and x is the measured state vector. The gain vector K was set as $K=[-0.2,30,-1.5,2.1]$ through pole placement and manual fine-tuning.

3) Experimental Procedure

To have a considerable sim-to-real gap, we utilize two versions of the setup. While the first version held the nominal configuration, the second involved an additional mass on the end of the pendulum rod built from six metal screws with two nuts on each screw. The screws were fixed on the end of the rod with a 3D-printed plastic holder (see Fig. 5). The overall mass of the additional weight m_w was 0.082 kg.

Overall, to verify the efficiency of the proposed approach six experiments were conducted:

1) Experiment A: An agent trained with the nominal model being tested on a real plant equivalent to the nominal model.
2) Experiment B: An agent trained with the model with domain randomization being tested on a real plant equivalent to the nominal model.
3) Experiment C: An agent trained with the model with domain randomization being trained additional episodes and tested on a real plant equivalent to the nominal model.
4) Experiment D: An agent trained with the nominal model being tested on a real plant with additional weights.
5) Experiment E: An agent trained with the model with domain randomization is tested on a real plant with additional weights.
6) Experiment F: An agent trained with the model with domain randomization is additionally trained and tested on a real plant with additional weights.

For these experiments, we initially aimed to conduct comprehensive real-world training from scratch to gauge the benchmark score under real-world conditions. However, after approximately four hours and 100 training episodes, we made the decision to halt the training. The reason behind this decision was that the agent was still making a substantial number of random moves, posing a risk to the setup's integrity. This experiment further supports why domain randomization in a simulation environment is beneficial for training RL agents for real-world applications.

Additional training in the real world is conducted as follows: We first load the agent, already trained in the simulation, into our Simulink program, which is designed for training RL agents. This program ensures that the reward function, state formulation, and set of available actions are consistent with those in the simulated environment. However, the actions are transmitted to the motor, whereas the state is computed from measurements of the sensors. After each training episode is executed, we pause to reset the setup to its initial state, with θ set to 0 and α set to π. Subsequently, a new episode is initiated.

The system is designed to prevent self-collision and is elevated to avoid table collisions. Cables are securely fixed to avoid twisting during arm rotations. User safety is ensured by keeping them out of the pendulum's workspace, with risk-free manual resets when the system is powered down.

According to our hypothesis, it is expected that the performance of agents in experiment F will be higher than in experiments D and E. Moreover, we look forward to observing that the performance of agents in experiment C is higher than in experiment B, although we admit that it may not exceed scores in experiment A due to the absence of the reality gap.

IV. RESULTS AND DISCUSSION

This chapter presents the outcomes of experiments previously introduced in the preceding chapter, along with a comprehensive analysis of the significant findings. The initial subsection of the results section presents an evaluation of the efficacy of our approach in sim-to-sim gap experiments. The second subsection reports the results of the approach in sim-to-real gap experiments. The section concludes with a discussion of the primary discoveries in the last subsection.

TABLE II: Sim-to-sim experiments with the cart pole system

Training environment	Training episodes	Testing environment	Test score
real plant	1000	real plant	500
nominal model	1000	nominal model	500
nominal model	1000	real plant	200
nominal model + real plant	1000 + 20	real plant	63
nominal model + real plant	1000 + 100	real plant	392
nominal model + real plant	1000 + 200	real plant	395
nominal model + real plant	1000 + 500	real plant	500
randomized model	1000	real plant	467
randomized model + real plant	1000 + 20	real plant	325
randomized model + real plant	1000 + 100	real plant	455
randomized model + real plant	1000 + 200	real plant	481
randomized model + real plant	1000 + 500	real plant	491

TABLE III: Sim-to-sim experiments with the simple pendulum

Training environment	Training episodes	Testing environment	Test score
real plant	400	real plant	-716
nominal model	400	nominal model	-698
nominal model	400	real plant	-974
nominal model + real plant	400 + 20	real plant	-983
nominal model + real plant	400 + 100	real plant	-1108
nominal model + real plant	400 + 200	real plant	-735
randomized model	400	real plant	-754
randomized model + real plant	400 + 20	real plant	-745
randomized model + real plant	400 + 100	real plant	-730
randomized model + real plant	400 + 200	real plant	-694

A. SIM-TO-SIM EXPERIMENTS RESULTS

In summary, our sim-to-sim experimental results validate our assumptions and hypotheses. Agents trained on a nominal model perform well in simulated real-world environments when there are no discrepancies between the plant and model simulations; however, their performance drops significantly when discrepancies are present, confirming the existence of the reality gap. On the other hand, our approach effectively bridges the sim-to-sim gap.

The experimental results reveal improved performance in uncertain "real world" environments after applying domain randomization during the training process. Furthermore, additional training in the "real plant" simulation generally shows considerable improvement after just 20 episodes. Agents trained with domain randomization and subsequent adaptation achieve superior results after 100-200 episodes in the simulation emulating a real plant.

In Tables II, III IV, and VI, "nominal model" denotes the model with nominal parameters used for training, while "real plant" refers to the testing environment diverging from the nominal model. "Randomized model" signifies the simulation implementing domain randomization. "Nominal model + real plant" and "randomized model + real plant" indicate that domain adaptation is applied to agents. In these instances, the number of training episodes is represented by addition. For instance, "400 + 20" implies that 20 episodes on the "real plant" are performed as domain adaptation in addition to the 400 episodes of training in either the nominal or domain-randomized model.

1) Experiments with cart pole system

The experimental results for the cart pole system, as presented in Table II, demonstrate that domain randomization provides generalization, while domain adaptation enables better fitting to specific conditions. By combining these techniques, we achieve a relatively good overall performance score.

When there is no mass discrepancy, the performance remains consistent across testing scenarios. However, when there is a gap, performance drops dramatically. In contrast, the agent trained in the model with a varying mass parameter experiences a comparatively smaller drop. After a sufficient number of episodes of domain adaptation on the "real plant", the methodology showed excellence by approaching the benchmark score. However, domain adaptation caused higher performance drops when we applied it independently of domain randomization.

2) Experiments with simple pendulum

As shown in Table III, when there are discrepancies between the training and testing simulations, the agent attains a drop of score. Domain randomization improves this score, and adding domain adaptation episodes in the "real plant" simulation further enhances the score to benchmark values. For this agent, we have not observed any performance drop when combining domain randomization with adaptation. However, in general, domain adaptation negatively impacts the "real-world" results of agents trained without domain randomization.

3) Experiments with quadruped robot

Table IV demonstrates the existence of the gap between training and testing environments. When discrepancies between training and testing environments arise, the score drops. The table also displays the effectiveness of combining domain randomization and domain adaptation. Domain randomization alone achieves a relatively higher score, but adding domain adaptation in the "real plant" simulation increases the score to the benchmark score. Domain adaptation, however, leads to an initial drop in performance for agents trained without randomization. It recovers the performance after some number of episodes but does not reach the benchmark score.

4) Experiments with ant robot

The consistent results of the experiments with the ant robot supported our hypothesis, aligning with the outcomes of our previous studies (see table V). Notably, we observed that agents trained with nominal models exhibited a more significant drop in performance when confronted with the diverging characteristics of the "real plant" model compared to those trained in randomized environments. Nevertheless, unlike previous experiments, we observed no drop in performance when directly applying domain adaptation on agents trained on the nominal model. Such a difference may be caused by the fact of using the PPO algorithm which has the

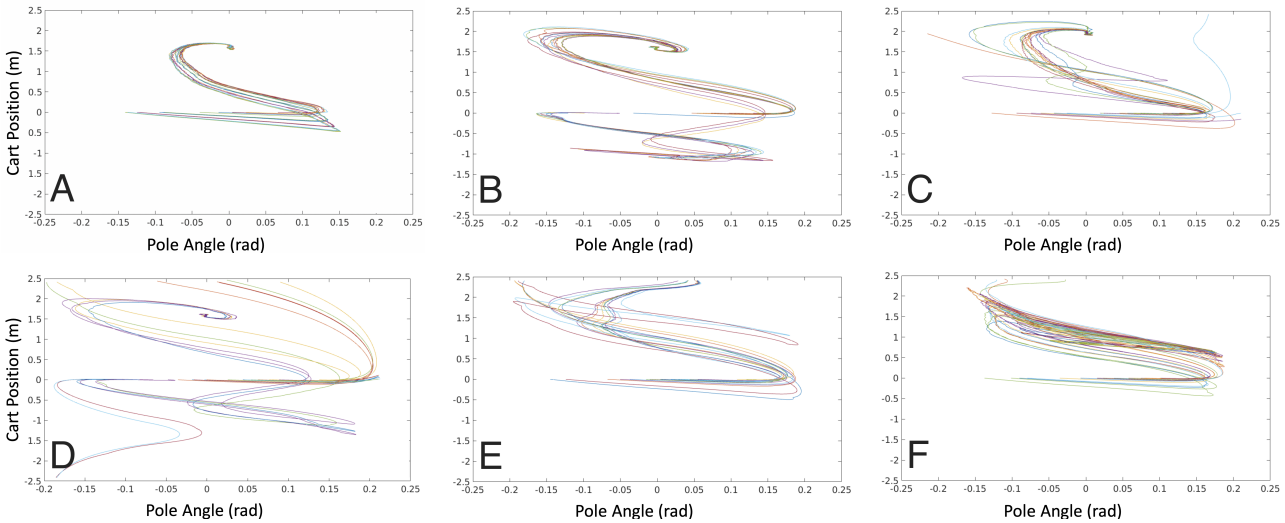


FIGURE 6: Captured trajectories of cart pole agents. (a) Distribution of optimal behavior for the "real plant." (b) Distribution of optimal behavior for the nominal model. (c) Distribution of optimal behavior for the randomized model includes the distributions of the behavior of both the nominal model and the "real plant." (d) The agent trained on the nominal model is destabilized on the "real plant," as the trajectories are not explored. (e) After 200 training episodes of domain adaptation on the "real plant," the agent trained with the nominal model has unstable behavior. (f) After 200 training episodes of domain adaptation on the "real plant," the initially randomized agent converged to a stable behavior. The behavior is considered successful if an agent does not move the cart outside the area from -2.5 m to 2.5 m along the Y-axis (the cart moves away from the center), and -0.26 rad to 0.26 rad along the X-axis (the pole falls).

TABLE IV: Sim-to-sim experiments with the quadruped robot

Training environment	Training episodes	Testing environment	Test score
real plant	5000	real plant	327
nominal model	5000	nominal model	321
nominal model	500	real plant	285
nominal model + real plant	5000 + 20	real plant	237
nominal model + real plant	5000 + 100	real plant	222
nominal model + real plant	5000 + 200	real plant	209
nominal model + real plant	5000 + 300	real plant	278
nominal model + real plant	5000 + 400	real plant	308
nominal model + real plant	5000 + 500	real plant	314
randomized model	5000	real plant	270
randomized model + real plant	5000 + 20	real plant	280
randomized model + real plant	5000 + 100	real plant	315
randomized model + real plant	5000 + 200	real plant	320
randomized model + real plant	5000 + 300	real plant	322
randomized model + real plant	5000 + 400	real plant	333
randomized model + real plant	5000 + 500	real plant	340

TABLE V: Sim-to-sim experiments with the ant system

Training environment	Training episodes	Testing environment	Test score
real plant	20	real plant	891
real plant	100	real plant	3078
real plant	200	real plant	5054
real plant	300	real plant	5532
real plant	1000	real plant	6112
nominal model	1000	nominal model	11865
nominal model	1000	real plant	2242
nominal model + real plant	1000 + 20	real plant	4683
nominal model + real plant	1000 + 100	real plant	4975
nominal model + real plant	1000 + 200	real plant	5108
nominal model + real plant	1000 + 300	real plant	5408
randomized model	1000	real plant	5810
randomized model + real plant	1000 + 20	real plant	6201
randomized model + real plant	1000 + 100	real plant	5804
randomized model + real plant	1000 + 200	real plant	6142
randomized model + real plant	1000 + 300	real plant	6519

advantage of maintaining a more diverse policy distribution than deterministic algorithms such as DQN and DDPG.

Furthermore, our experiments revealed that domain adaptation was particularly effective when applied to agents trained with randomized models. This can be attributed to the fact that randomized agents encompass a wide distribution of policies, which includes configurations resembling those encountered in real-world scenarios.

Remarkably, the combination of domain randomization and domain adaptation yielded even better results than benchmark scores obtained by training agents directly on the "real plant" model. This finding underscores the effectiveness of our approach in bridging the gap between simulated training and real-world deployment.

5) Experiments with cart pole trajectories

Figure 6 illustrates the advantage of combining domain adaptation with domain randomization rather than applying domain adaptation directly shown in the Cart Pole experiment. We can see in Fig. 6a and 6b the distributions of optimal behavior for the "real plant" and the nominal model, respectively, are not matching, implying a possibility of a distributional shift problem when transferring the experience

TABLE VI: Sim-to-real experiments with the Rotary Inverted Pendulum

Training environment	Extra weight	Average swing-up time	Success
nominal model	none	0.47 s	10/10
randomized model	none	2.81 s	10/10
randomized model + real plant	none	3.39 s	10/10
nominal model	82 g	6.62 s	4/10
randomized model	82 g	2.28 s	10/10
randomized model + real plant	82 g	2.04 s	10/10

from one model to another.

On the other hand, the distribution of optimal behavior for the randomized model in Fig. 6c includes the distributions of the behavior of both the nominal model and the "real plant", demonstrating a chance to overcome the distributional shift problem. Indeed, as seen in Fig. 6d, the agent trained on the nominal model is destabilized on the "real plant," as the trajectories are not explored. While after 200 training episodes of domain adaptation on the "real plant," the initially randomized agent converged to a stable behavior (see Fig. 6e), the agent trained with the nominal model has unstable behavior even after 200 training episodes of domain adaptation on the "real plant" (see Fig. 6f).

B. SIM-TO-REAL EXPERIMENTS RESULTS

The outcomes of the experiments are presented in Table VI, showcasing the results of conducting 10 repetitions of each experiment (labeled A to F) on the actual setup, with each trial lasting 10 s.

As seen in Table VI and Figure 7a, the agent trained on the nominal model performs perfectly on an equivalent real plant, with 10 successful trials out of 10 and an average swing-up time of half a second. However, when the additional mass not included in the model is added, performance drops to 4 successful trials out of 10 (see Fig. 7d). In contrast, domain randomization allows agents to consistently swing up the pendulum successfully in both scenarios (see Fig. 7b and Fig. 7e). The agent that underwent domain adaptation achieved the best result among the others in the scenario with extra mass not included in the nominal model (see Fig. 7f). However, it did not improve the performance obtained after domain randomization in the scenario without the additional mass (see Fig. 7c). A video showing the experiments is provided as supplementary material.

C. DISCUSSION

In our discussion, we analyze the major observations from the experiments in order to qualitatively estimate the efficacy of our methodology for the sim-to-real transfer of trained RL policy.

First of all, it is important to indicate that the experimental results from both sim-to-sim and sim-to-real experiments validate our assumptions about sim-to-real transfer problems that our approach is designed to solve. The experiments demonstrate the reality gap's negative impact on the performance of agents trained on a simulated model when discrepancies with the real plant exist. Furthermore, they also prove the existence of a distributional shift problem in domain adaptation of agents trained in deterministic environments, as well as the insufficiency of using domain randomization alone to reach benchmark scores.

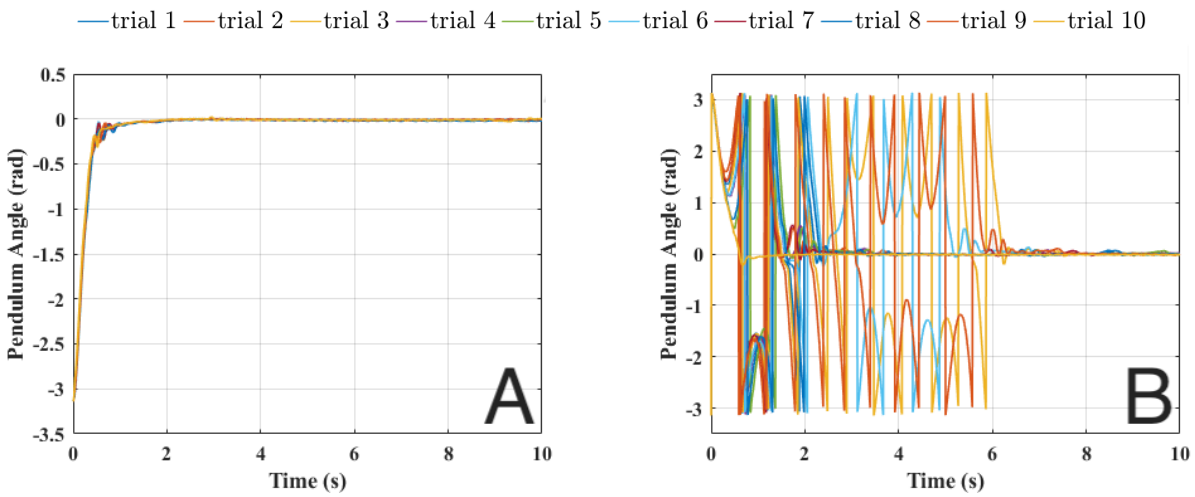
Secondly, the experiments illustrate that our approach effectively bridges the reality gap, showing improved performances in uncertain real-world environments. Domain randomization, as demonstrated in our experiments, contributes to a better generalization of learned policies. Meanwhile, domain adaptation provides a more tailored fit to specific conditions. By combining these techniques, we achieved a relatively good overall performance score in the cart pole system, simple pendulum, quadruped robot, ant robot, and rotary inverted pendulum scenarios. In our sim-to-real experiments with the real rotary inverted pendulum, we demonstrate the potential of our approach to be applied in real robotics that require balancing control.

In most experiments, the number of real-world training episodes required after the domain randomization is significantly less than those needed for direct training on the real plant. This reduction to around 200 training episodes offers a substantial advantage over training an agent directly on the plant from scratch, which varies from 400 to 5000 training episodes. Furthermore, a pre-trained agent obtained after simulation training will perform fewer random actions to explore policies, thereby reducing the occurrence of potentially harmful moves.

Although the feasible amount of real-world training may vary across applications, there might be scenarios when the real-world training is not affordable, e.g. space rockets. We recommend exploiting our approach when at least 200 such episodes can be performed.

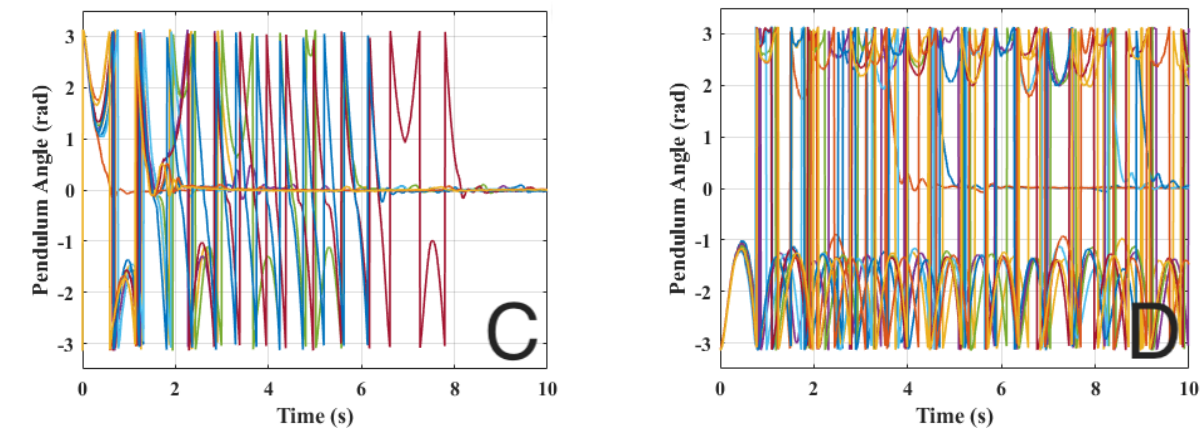
Thirdly, our experiments consistently underscore the distributional shift problem. For instance, when subjecting agents trained in static nominal models to real-world conditions for just twenty episodes, a considerable decline in performance becomes evident. Intriguingly, it takes a substantially larger number of episodes, often several hundred, to restore their initial scores. In stark contrast, when fine-tuning a randomized agent to adapt to real-world conditions, we observe either a minor performance dip after a mere 20-50 real-world episodes (as seen in the Cart Pole experiment and Rotary Inverted Pendulum experiment), or no performance drop at all (as demonstrated in the Simple Pendulum, Quadruped Robot experiments and Ant Robot experiments). This discrepancy can be attributed to the fact that the randomization ranges encompassed the actual real-world conditions, rendering the agents more adaptable to novel scenarios.

Additionally, our results indicate that domain adaptation has the potential to be applied independently of domain randomization in stochastic RL algorithms such as PPO. Although the performance scores were generally lower compared to the combined approach, domain adaptation alone exhibited stable improvement over time in the experiments with ant robots (subsection IV-A4). This finding suggests that



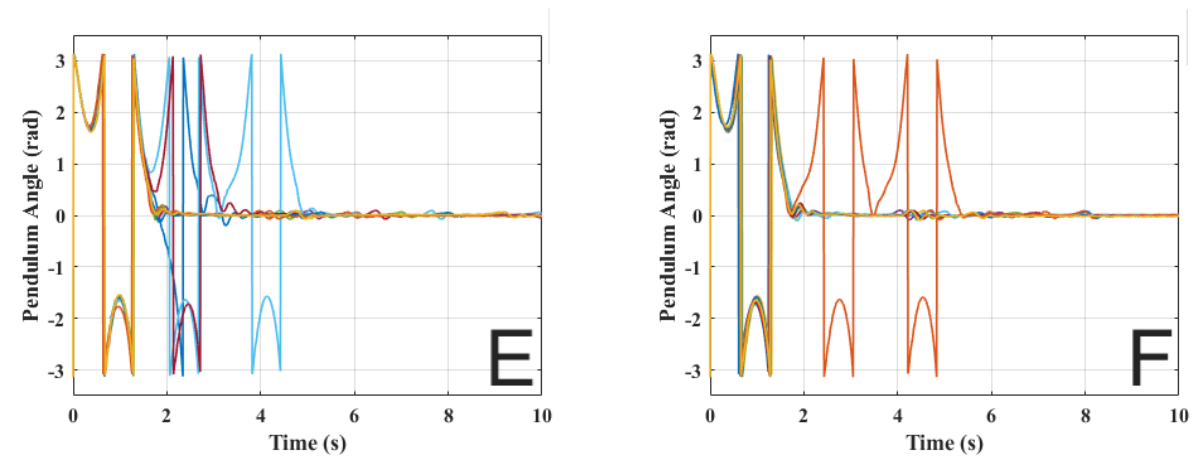
Agent trained with the nominal model is tested on the equivalent real plant. 10 out of 10 successful swing-ups with an average time of 0.47 s.

Agent trained with domain randomization is tested on a real plant equivalent to the nominal model. 10 out of 10 successful swing-ups with an average time of 2.81 s.



Agent trained with domain randomization and adaptation is tested on a real plant equivalent to the nominal model. 10 out of 10 successful swing-ups with an average time of 3.39 s.

Agent trained with the nominal model is tested with additional weights. 4 out of 10 successful swing-ups with an average time of 6.62 s.



Agent trained with domain randomization is tested with additional weights. 10 out of 10 successful swing-ups with an average time of 2.28 s.

Agent trained with domain randomization and adaptation is tested with additional weights. 10 out of 10 successful swing-ups with an average time of 2.04 s.

FIGURE 7: Pendulum angle during real-world experiments A-F. Each experiment was repeated 10 times.

domain adaptation could serve as a viable alternative to the combined approach when using a stochastic RL. Nevertheless, our observation regarding the intricate relationship between domain randomization and real-world adaptability significantly aligns with our hypothesis that the hybrid approach successfully addresses the distributional shift problem.

Finally, since our experiments display a considerable variety of environments and deep RL algorithms, the research highlights the generalization of our method. Through a comprehensive set of experiments spanning diverse problem domains, including the cart pole, simple pendulum, quadruped robot, ant robot, and rotary inverted pendulum, we have showcased the adaptability of our methodology. Furthermore, we utilized two distinct simulation platforms: Matlab with Simulink and Isaac Sim. Finally, our choice to employ three different deep reinforcement learning algorithms, DQN, DDPG, and PPO underscores the versatility of our approach, demonstrating that its benefits extend beyond specific problem instances. While the application of domain randomization and adaptation may require tailored adjustments for individual scenarios, our research provides compelling evidence that this combination serves as a promising foundation for addressing the challenges of parameter mismatches and uncertainties across a wide range of robotic control tasks. We believe that our findings open avenues for future research in leveraging this approach across various problem domains and reinforcement learning algorithms, potentially leading to more robust and versatile robotic systems.

V. FUTURE WORKS AND CONCLUSION

In order to address the limitation of affordability of real-world training, we can try to decrease the level of underfitting in the domain randomization process in order to improve adaptability during domain adaptation. One promising direction is the incorporation of active domain randomization [22], [33], which can adaptively select parameter variation range to maximize policy learning. For example, [22] proposed applying elements of system identification as a Gaussian process to find a black-box relationship between randomization parameters and the performance of an agent in a real plant during periodic roll-outs on the real plant. This approach could potentially lead to even better generalization of learned policies and reduced training time in real-world environments. However, this approach would involve periodic roll-outs on a real setup to estimate the efficiency of the applied range. Therefore, adjustments to the randomization ranges should be done carefully.

Domain adaptation can also utilize alternative methods such as adversarial-based training or metric learning to increase data efficiency [34], [35]. The former refines discriminators to distinguish between the target and training domains, whereas the latter technique teaches agents to measure similarities between inputs, enabling the model to predict the output for an unseen target domain input based on its similarity to inputs from the training domain.

On the other hand, enhancing the data efficiency of the real-world training might be possible if we apply model-based RL [51]. In model-based RL an agent models the dynamics of an environment by performing transitions and recording resultant rewards and observations, while a policy is generated by a planner algorithm. This might be a dynamic programming, tree search, or other decision-making algorithms from optimal control theory. This approach is considered to utilize obtained data more efficiently, although the learned model does not guarantee accuracy. The algorithms that are exploited in this paper belong to model-free RL which builds policy without knowledge of a model, but by mapping transitions to estimated values of possible cumulative reward.

Although our agents were able to generalize to the provided randomization ranges, there might be cases when the level of parameter uncertainty leads to a vast range of randomization to which an agent might not be able to generalize. In this case, a curriculum learning approach might be useful [52]. These methods can help to prevent convergence to suboptimal behaviors by gradually increasing the difficulty of a task.

Another insight for future work is solving the challenge of selecting the optimal learning rate, as excessively high values can cause divergence, while overly low values hinder fast convergence, significantly affecting domain adaptation quality. Moreover, monotonously decreasing learning rates may trap neural networks in suboptimal states, possibly causing premature stagnation [53]. Implementing an attenuation strategy, which involves dynamically adjusting the learning rate based on gradient observations [53] or utilizing reinforcement learning agents to adapt it to observed states [54], is essential for robustness. Although not employed in our current research due to its relatively uncommon use and tool limitations, we acknowledge its importance. In future work, we plan to incorporate a learning rate scheduler to optimize training and enhance agent adaptability in real-world scenarios.

Finally, expanding the range of environments to dexterous manipulation tasks can validate the versatility of our method. Investigating other applications where dynamic mismatches between simulation and reality are critical, such as autonomous vehicles, aerial robotics, and industrial automation, can provide valuable insights into the efficacy of our approach in diverse settings. By exploring these potential improvements and extending the scope of our methodology, we can continue to refine sim-to-real transfer techniques and contribute to the broader field of RL.

To conclude, our sim-to-real transfer methodology, which combines domain randomization and domain adaptation by training in a simulation with randomization of unreliable parameters followed by additional real-world training, demonstrates promising results in bridging the reality gap and improving the performance of RL agents in uncertain environments. This approach holds potential for further development and application in various real-world scenarios, where robust and adaptable agents are crucial for success.

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT press, 1998.

[2] I. Sarker, "Machine learning: algorithms, real-world applications and research directions," *Social Netw. Comput. Sci.*, vol. 2, p. 160, 2021.

[3] E. Gross, "On the Bellman's principle of optimality," *Phys. A Stat. Mech. Appl.*, vol. 462, pp. 217–221, 2016.

[4] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Syst.*, vol. 12, no. 2, pp. 19–22, 1992.

[5] D. E. Kirk, *Optimal Control Theory: An Introduction*. Upper Saddle River, NJ, USA: Prentice-Hall, 1970.

[6] Google DeepMind, *ANYmal B Description (MJCF)*, 2023. [Online]. Available: https://github.com/deepmind/mujoco_menagerie/tree/main/anybotics_anymal_b

[7] Open Source Robotics Foundation, Inc., *Gazebo Simulation Software Showcase*, 2023. [Online]. Available: https://gazebosim.org/showcase

[8] NVIDIA Corporation, *Omni.Anim.People*, 2023. [Online]. Available: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/ext_omni_anim_people.html

[9] Cyberbotics Ltd., *Webots User Guide*, 2023. [Online]. Available: https://cyberbotics.com/doc/guide/youbot

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[12] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[13] D. Dutta and S. R. Upreti, "A survey and comparative evaluation of actor-critic methods in process control," *The Canadian Journal of Chemical Engineering*, vol. 100, no. 9, pp. 2028–2056, 2022.

[14] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola, "Elastica: a compliant mechanics environment for soft robotic control," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3389–3396, 2021.

[15] J. Yue, "Learning locomotion for legged robots based on reinforcement learning: A survey," in *Proc. Int. Conf. Electr. Eng. Control Technol (CEECT)*, 2020, pp. 1–7.

[16] H. Sun, W. Zhang, R. Yu, and Y. Zhang, "Motion planning for mobile robots—focusing on deep reinforcement learning: a systematic review," *IEEE Access*, vol. 9, pp. 69 061–69 081, 2021.

[17] D. Görges, "Relations between model predictive control and reinforcement learning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920–4928, 2017.

[18] W. Zhao, J. Peña Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, 2020, pp. 737–744.

[19] L. Ljung, *System Identification: Theory for the User*. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.

[20] ——, "Perspectives on system identification," *Annu. Rev. Control*, vol. 34, no. 1, pp. 1–12, 2010.

[21] 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.*, 2017, pp. 23–30.

[22] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Bayesian domain randomization for sim-to-real transfer," *arXiv preprint arXiv:2003.02471*, 2020.

[23] N. Jakobi, "Evolutionary robotics and the radical envelope-of-noise hypothesis," *Adapt. Behav.*, vol. 6, no. 2, pp. 325–368, 1997.

[24] J. Morimoto and K. Doya, "Robust reinforcement learning," *Neural Comput.*, vol. 17, no. 2, pp. 335–359, 2005.

[25] H. Zhang, H. Chen, C. Xiao, B. Li, M. Liu, D. Boning, and C.-J. Hsieh, "Robust deep reinforcement learning against adversarial perturbations on state observations," in *Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 21 024–21 037.

[26] J. Moos, K. Hansel, H. Abdulsamad, S. Stark, D. Clever, and J. Peters, "Robust reinforcement learning: A review of foundations and recent advances," *Mach. Learn. Knowl. Extr.*, vol. 4, no. 1, pp. 276–315, 2022.

[27] O. Levy and L. Wolf, "Live repetition counting," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 3020–3028.

[28] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[29] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 12 627–12 637.

[30] R. Antonova, S. Cruciani, C. Smith, and D. Kragic, "Reinforcement learning for pivoting task," *arXiv preprint arXiv:1703.00472*, 2017.

[31] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1808.00177

[32] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: a survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153 171–153 187, 2021.

[33] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," in *Conf. Robot Learn.*, 2020, pp. 1162–1176.

[34] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[35] X. Guo and H. Yu, "On the domain adaptation and generalization of pretrained language models: A survey," *arXiv preprint arXiv:2211.03154*, 2022.

[36] H. Niu, Y. Qiu, M. Li, G. Zhou, J. HU, X. Zhan *et al.*, "When to trust your simulator: Dynamics-aware hybrid offline-and-online reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 36 599–36 612, 2022.

[37] C. J. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. learn.*, vol. 8, pp. 279–292, 1992.

[38] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[39] Y. Mohan, S. G. Ponnambalam, and J. I. Inayat-Hussain, "A comparative study of policies in q-learning for foraging tasks," in *Proc. IEEE Nat. Biol. Inspired Comput.*, 2009, pp. 134–139.

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

[41] S. P. Bhattacharyya, "Robust control under parametric uncertainty: an overview and recent results," *Annu. Rev. Control*, vol. 44, pp. 45–77, 2017.

[42] MathWorks, *Train DQN Agent to Balance Cart-Pole System*, 2023. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/train-dqn-agent-to-balance-cart-pole-system.html

[43] ——, *Train DDPG Agent to Swing-Up and Balance Pendulum*, 2023. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/train-ddpg-agent-to-swing-up-and-balance-pendulum.html

[44] ——, *Quadruped Robot Locomotion using DDPG*, 2023. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/quadruped-robot-locomotion-using-ddpg-gent.html

[45] Z. Mandi, P. Abbeel, and S. James, "On the effectiveness of fine-tuning versus meta-reinforcement learning," *arXiv preprint arXiv:2206.03271*, 2022.

[46] H. Xu, S. Ebner, M. Yarmohammadi, A. S. White, B. Van Durme, and K. Murray, "Gradual fine-tuning for low-resource domain adaptation," *arXiv preprint arXiv:2103.02205*, 2021.

[47] J. Mohyla, "Inverted pendulum remote real-time controller implemented in matlab-simulink and labview environments," 2016. [Online]. Available: https://core.ac.uk/download/pdf/44402200.pdf

[48] Quanser Inc., *Quanser Rotary Inverted Pendulum Manual*, 2023. [Online]. Available: https://www.quanser.com/products/rotary-inverted-pendulum/

[49] ——, *Quanser Servo Base Unit Manual*, 2023. [Online]. Available: https://www.quanser.com/products/rotary-servo-base-unit/

[50] S.-E. Oltean, "Swing-up and stabilization of the rotational inverted pendulum using PD and Fuzzy-PD controllers," *Procedia Technol.*, vol. 12, pp. 57–64, 2014.

[51] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "Model-based reinforcement learning: A survey," *" Found. Trends Mach. Learn.*, vol. 16, no. 1, pp. 1–118, 2023.

[52] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 7382–7431, 2020.

[53] D. Vidyabharathi, V. Mohanraj, J. S. Kumar, and Y. Suresh, "Achieving generalization of deep learning models in a quick way by adapting t-htr

learning rate scheduler," *Personal and Ubiquitous Computing*, vol. 27, no. 3, pp. 1335–1353, 2023.

[54] Z. Xu, A. M. Dai, J. Kemp, and L. Metz, "Learning an adaptive learning rate schedule," *arXiv preprint arXiv:1909.09712*, 2019.

AIDAR SHAKERIMOV received the B.S. degree in computer science and M.S. degree in robotics from Nazarbayev University, Astana, Kazakhstan, in 2021 and 2023, respectively. He is currently working as a Research Assistant at the Institute of Smart Technologies and Artificial Intelligence (ISSAI) at Nazarbayev University, Astana, Kazakhstan. His research interest includes reinforcement learning and dexterous control of manipulators.

TOHID ALIZADEH received a B.Sc. degree in electrical engineering from the Sahand University of Technology, Tabriz, Iran, an M.Sc. degree in automation and instrumentation engineering from the Petroleum University of Technology, Tehran, Iran, and a Ph.D. degree in robotics, cognition and interaction technologies from the Italian Institute of Technology (IIT), Genoa, Italy, in 2004, 2007 and 2014, respectively. He is currently an Assistant Professor at the School of Engineering and Digital Sciences, Robotics and Mechatronics Department, Nazarbayev University, Astana, Kazakhstan. His research interests include robotics, programming by demonstration, statistical machine learning, industrial automation, and smart manufacturing.

HUSEYIN ATAKAN VAROL (M"09, SM"16) received his B.S. degree in mechatronics engineering from Sabanci University, Istanbul, Turkey, in 2005, and M.S. and Ph.D. degrees in electrical engineering from Vanderbilt University, Nashville, TN, USA, in 2007 and 2009, respectively. He was a postdoctoral research associate and then a research assistant professor at Vanderbilt University from 2009 to 2011. In 2011, he joined the faculty of Nazarbayev University, Astana, Kazakhstan, where he currently chairs the Department of Robotics and Mechatronics and directs the Institute of Smart Systems and Artificial Intelligence (ISSAI). His research interests include soft robotics, machine learning, computer vision, sensor fusion, and tensegrity. He has published over 100 technical papers on related topics in international journals and conferences.

Dr. Varol was a finalist for the KUKA Innovation Award in 2014. He was also the recipient of the IEEE International Conference on Rehabilitation Robotics Best Paper Award in 2009, and the IEEE Engineering in Medicine and Biology Society Outstanding Paper Award in 2013. He served as a technical and associate editor for prominent journals and conferences.