

Received 3 May 2023, accepted 16 May 2023, date of publication 19 May 2023, date of current version 30 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3278208

## APPLIED RESEARCH

# Seg-CURL: Segmented Contrastive Unsupervised Reinforcement Learning for Sim-to-Real in Visual Robotic Manipulation

**BINZHAO XU<sup>1</sup>**, **TAIMUR HASSAN<sup>2</sup>**, AND **IRFAN HUSSAIN<sup>1,3</sup>**

<sup>1</sup>Khalifa University Center for Autonomous Robotic Systems (KUCARS), Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates

<sup>2</sup>Department of Electrical and Computer Engineering, Abu Dhabi University, Abu Dhabi, United Arab Emirates

<sup>3</sup>Department of Mechanical Engineering, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates

Corresponding author: Irfan Hussain (irfan.hussain@ku.ac.ae)

This work was supported by the Khalifa University of Science and Technology under Award FSU-2021-019 and Award RC1-2018-KUCARS.

**ABSTRACT** Training image-based reinforcement learning (RL) agents are sample-inefficient, limiting their effectiveness in real-world manipulation tasks. Sim2Real, which involves training in simulations and transferring to the real world, effectively reduces the dependence on real data. However, the performance of the transferred agent degrades due to the visual difference between the two environments. This research presents a low-cost segmentation-driven unsupervised RL framework (Seg-CURL) to solve the Sim2Real problem. We transform the input RGB views to the proposed semantic segmentation-based canonical domain. Our method incorporates two levels of Sim2Real: task-level Sim2Real, which transfers the RL agent to the real world, and observation-level Sim2Real, which transfers the simulated U-nets to segment real-world scenes. Specifically, we first train contrastive unsupervised RL(CURL) with segmented images in the simulation environment. Next, we employ two U-Nets to segment robotic hand-view and side-view images during real robot control. These U-Net are pre-trained with synthetic RGB and segmentation masks in the simulation environment and fine-tuned with only 20 real images. We evaluate the robustness of the proposed framework in both simulation and real environments. Seg-CURL is robust to the texture, lighting, shadow, and camera position gap. Finally, our algorithm is tested on a real Baxter robot with a dark hand-view in the cube lifting task with a success rate of 16/20 in zero-shot transfer.

**INDEX TERMS** Reinforcement learning, robotic manipulation, Sim2Real, zero-shot learning.

## I. INTRODUCTION


In recent years, deep reinforcement learning (DRL) has achieved remarkable results in robot control, ranging from robotic manipulation to robotic locomotion [1], [2], [3]. Among kinds of DRL methods, visual-based DRL can give more potential for solving complex robotics tasks [1], [4] since the image contains a large amount of hidden information, such as contacting, reaching, and slipping.

However, DRL with high dimensional observations, like images, requires significantly more samples than the physical state-based features [5]. It usually needs millions of steps to train an agent in high-dimensional tasks [6]. Also, at the

initial training stage, the robot acts randomly to explore the workspace fully, which causes damage to the real robot. [7].

Transferring a well-trained agent from the simulation to the real world (Sim2Real) is an effective solution to avoid these problems [7], [8]. But transferring a vision-based control policy to the real world usually meets a low success rate due to the gap between reality and simulation environments. It is hard to set the parameters in the simulation environment the same as in the real world, causing visual gap such as rendering, texture, lighting, and camera position.

There are two methods to close the gap of Sim2Real: 1) Make the simulation more like reality; 2) Learn invariant representations between simulation and reality, and then train the RL agent by this representation. Benefiting from the recent development of the computer's hardware like GPU, a computer could render a 3D scene which is hard for a

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

human to distinguish from if it is from the real world or simulation [9]. But these methods are designed for the general simulation environment and need huge numbers of real data to train. It is challenging to transfer the simulation rendering to a specific real lab scenario using a limited real-world vision sample.

As for methods related to learning invariant representation, domain randomization is a common way [10]. By adding random noise in the simulation parameters, such as camera position, and light color, the RL agent can learn a representation invariant to these parameters. When transferred to the real environment, the trained agent is robust with these parameters. But this method will increase the complexity of the problem since the agent has to be trained in a wider observation space. Another way is to define a canonical observation space, then transform simulation observation and real observation into the same canonical space to reduce the gap between reality and simulation environment [11].

Our method belongs to the second class, which transfers the observation space to a canonical space – segmented image. Firstly, we train visual grasping strategies with CURL [12] algorithm (contrastive unsupervised for reinforcement learning) using the segmented images in the simulation environment with sparse reward. To tackle the problem of sparse reward during RL training, we design a specialized environment that captures visual observations from an expert agent trained with low-dimension features and dense reward. Secondly, we pre-train the U-nets with simulation images and corresponding segmented images, then fine-tune these U-Net with a small real segmentation data set (20 real images). In this way, we greatly reduce the need for real segment labels, which are time-consuming to obtain. Finally, we control the robotic manipulator with well train agent from the simulation with segmented real data. To summarize, the main contributions of this work are:

- This paper proposes a low-cost Sim2Real framework via self-contrastive learning and semantic segmentation. In our framework, only 20 real images are needed to achieve image segmentation and control of a real robot. It takes around 2 hours to train a stable RL grasping and lifting agent.
- Create an “imitation environment” to give expert demonstrations for the visual-based sparse reward agent. In this way, the cumbersome human demonstration collection process is avoided when solving the sparse reward problem in the RL training.
- We demonstrate the robustness of the segmented CURL algorithm with the texture, shadows, and camera position offset in both simulation and real Baxter on the lifting task.

## II. RELATED WORK

### A. LEARNING INVARIANT REPRESENTATION FOR *Sim2Real*

Although some physic-state based RL methods [13] can transfer to the real world directly, image-base RL algorithms

need to find an invariant representation to achieve successful Sim2Real transfer. There exist two primary categories of invariant representation: 1) Feature-level invariant representation, which aims to identify a low-dimensional representation that contains only task-related information. 2) Pixel-level invariant presentation, which involves transferring simulation and real vision into the same canonical visual space.

Regarding feature-level representation learning, Puang et al. [14] adopt an auto-encoder to extract key points from stereo cameras. They subsequently employ behavior cloning (BC) to control the robot using these key points. Since this method is based on the auto-encoder structure, it requires a substantial amount of data to train the encoder (approximately 35,000 images). Given that self-contrastive learning can be more data-efficient than auto-encoders, Jeong et al. [15] employs time contrastive learning to obtain the encoder for the observed image. The corresponding action is derived from DRL rather than BC, making this method more stable in unseen situations.

To use well-developed visual-based control methods directly, pixel-level invariant presentation is a good option. James et al. [11] define a canonical pixel observation where different colors represent different joints and objects. They employ a GAN-based generator called Randomized to Canonical Adaptation Network to translate the randomized simulation image into the canonical space. Similarly, Rao et al. [16] utilize CycleGAN, which consists of two generators - Sim2Real and Real2Sim. Furthermore, they introduce a Q-function loss, which ensures that paired simulation and real observations have similar Q values. However, GAN-based generators may face the issue of instability, slow convergence, and high computational costs.

On the other hand, Pashevich et al. [17] consider the depth image as the canonical space and train agents on the depth image augmented with random noise. This method depends on the depth camera. Similarly to our work, Hong et al. [18] also use segmented images in robot control. But their only show the basic reaching task on the mobile robot.

### B. RL BASED VISUAL CONTROL

Image-based observations can implicitly provide crucial information, such as object position and the contact state of the end-effectors, which are often difficult or expensive to obtain in the real world [19]. As a result, this approach has immense potential for solving complex robotic tasks. Nonetheless, training visual control through Deep Reinforcement Learning (DRL) needs millions of data points. Furthermore, these frameworks are plagued by the problem of sparse rewards [20], only receiving rewards upon successful task completion. A common approach for addressing the sparse reward issue involves providing expert demonstrations.

To address the data inefficiency of image-based RL, a preprocessing step is required to reduce the observation’s dimensionality. In this respect, self-supervised learning has attracted considerable attention due to its data efficiency and

generalization capabilities. There are three primary methods in self-supervised learning [21]: 1) Generative self-supervised learning methods, such as PixelRNN [22] and GraphRNN [23]; 2) Adversarial self-supervised learning methods, like RoCL [24]; and 3) Contrastive self-supervised learning methods, including SimCLR [25], CURL [12], and the most recent version of self-contrastive learning even extracts features along the time dimension [26]. Among these algorithms, contrastive self-supervised learning methods require fewer data and exhibit greater stability during training.

Based on the discussion above, we choose the contrastive unsupervised algorithm as the backbone for reducing the dimensionality of visual observations. CURL derives representations from images using self-contrastive representation learning, which requires less data for training compared to auto-encoding. To address the sparse reward problem, several expert trajectories are incorporated into the replay buffer. However, CURL, and more generally, self-contrastive representation learning methods, exhibit limited performance in domain adaptation, particularly in bridging the Sim2Real gap [27]. To overcome these challenges, we combine CURL with segmentation-driven domain shifts to enhance its Sim2Real transferability.

### III. METHODS

In this section, we introduce the primary workflow of our study on visual robotic manipulation. The proposed Seg-CURL framework is depicted in Figure 1 and consists of three essential components: 1) Expert Demonstration 2) Contrastive Unsupervised Representation for RL. 3) Sim2Real Transfer Module, where real image observations are segmented using a Double U-Net.

#### A. COLLECT EXPERT DEMONSTRATION

To address the sparse reward issue while training an image-based RL agent, we gather 20 expert trajectories with image observations. As illustrated in Figure 2, we initially train an expert agent using physics-informed observations (robot end-effector states and object states), which are readily available in the simulation environment. Subsequently, a specialized environment—referred to as the imitation environment—is employed to collect image-based observations.

##### 1) TRAIN EXPERT AGENT

In general, RL algorithms can be classified into on-policy and off-policy. On-policy algorithms typically exhibit more stable convergence properties as the agent learns from data collected by the current policy. On the other hand, off-policy learning allows the policy to be updated using experiences from older policies, making it more data-efficient. In this paper, we address a relatively stable task (manipulating a static object). Off-policy algorithms have the potential to learn faster without compromising stability. The RL algorithm we choose is Soft Actor-Critic (SAC) [28]. Compared to other

off-policy RL algorithms, this method is relatively less sensitive to hyperparameters such as learning rate and reward discount factor. The main difference between SAC and other actor-critic algorithms (such as TD3 [29]) is that SAC is based on maximum entropy RL. This adaptation transforms the Deep RL problem into finding a network-policy  $\pi$ —that maximizes accumulated reward as defined in Equation (1). The entropy measures the randomness of the policy. High entropy means high exploring rates. By incorporating entropy, the robot keeps exploring the action space while maintaining high rewards during training.

$$\pi^*(a) = \arg \max_{\pi} E_{\tau \sim \pi} \left[ \sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot | s_t))) \right] \quad (1)$$

where:  $\pi$  is the policy,  $\tau$  is the trajectory,  $\gamma$  is the reward discount factor,  $s_t$  is the observed state at time  $t$ ,  $a_t$  is action at time  $t$ ,  $H(\pi(\cdot | s_t))$  is the entropy of the policy at  $s_t$ .  $\alpha$  is the trade-off coefficient.

In this application, we employ an auto-tuning version of SAC [28] to determine the trade-off coefficient  $\alpha$ . Task-related variables—observation space  $s$ , action space  $a$ , and reward  $r$ —are defined as follows: The observation at a time  $t$  is:

$$s_t = \{p_t^{obj}, p_t^{gripper}, g, d_{o,p}\} \quad (2)$$

where:  $p_t^{obj}$ ,  $p_t^{gripper}$  is the  $x, y, z$  position of the object and gripper in the world coordinate.  $g$  is the gripper's open state, normalized into the range  $[-1, 1]$ , where  $-1$  present close and  $1$  present open.  $d_{o,p}$  is the distance from the cube to the gripper.

The agent's action space consists of the changes in the end-effector's  $x, y, z$  direction and the state of the gripper, as shown in Equation (3). These values are normalized to  $[-1, 1]$  and multiplied by a scale factor when controlling the real robot.

$$a = \{\Delta x, \Delta y, \Delta z, g\} \quad (3)$$

As for the reward: We design dense rewards for the grasping task, similar to those in Robosuite [30], to accelerate the learning process. The lifting task consists of three stages: 1) reaching stage; 2) grasping stage; 3) lifting stage. The corresponding rewards are designed as follows:

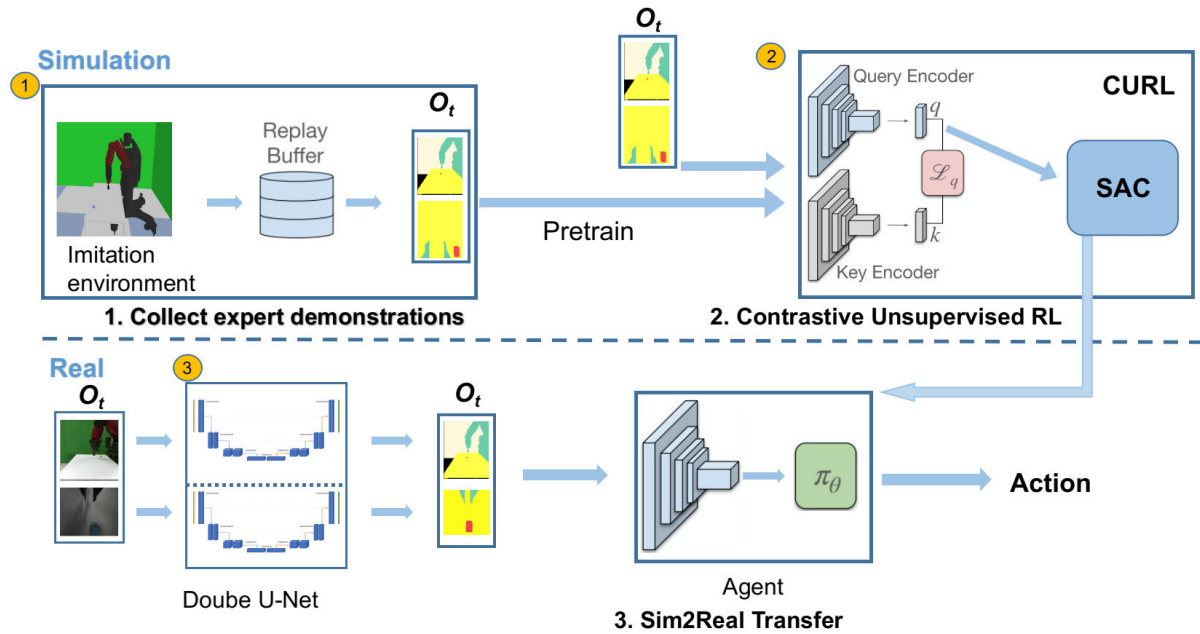
- 1) Reaching reward.

$$r_{reaching} = 1 - \tanh(10 * d_{o,p}) \quad (4)$$

where:  $d_{o,p}$  is the distance from the end-effector to the object.

- 2) Grasping reward. If two grippers touch the object, the agent will get the grasping reward.

$$r_{grasping} = \begin{cases} 0.25 & \text{contact} \\ 0 & \text{non-contact} \end{cases} \quad (5)$$



**FIGURE 1.** The Framework for Seg-CURL. 1. Expert demonstration collection. 2. Contrastive Unsupervised Representation for Reinforcement Learning. 3. Sim2Real Module based on a Double U-Net.

- 3) Lifting reward. When the object is above the table more than 5cm, the agent gets the lifting reward.

$$r_{lifting} = \begin{cases} 2.5 & z_{obj} > 0.05m \\ 0 & z_{obj} \leq 0.05m \end{cases} \quad (6)$$

where:  $z_{obj}$  is the lift height of the object.

- 4) Acting reward. To restrain the random movement, the agent gets a small negative reward in each step.

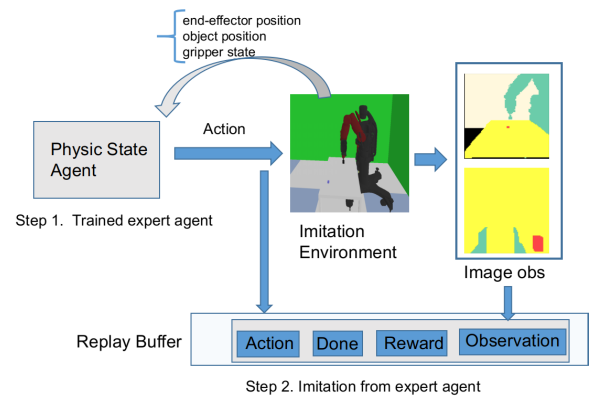
$$r_{act} = -0.1 \quad (7)$$

The total reward  $r$  is:

$$r = (r_{reaching} + r_{grasping} + r_{lifting} + r_{act}) \quad (8)$$

## 2) IMITATION ENVIRONMENT

Once the physics-based agent is trained, we need to store visual observations of this agent. For this purpose, we design an imitation environment, as shown in Figure 2. This environment has both physical feature observations and image observations (segmented images). In each step of the RL process, the physical observation  $obs_t^p$  is sent to the expert agent to generate expert action  $a_t$ , while the image observation  $obs_t$  is stored in the replay buffer. Then, the robot executes action  $a_t$ , and the environment transitions to a new state with  $obs_{t+1}^p, obs_{t+1}$ , repeating the above process until the end of the trajectory. During this process, action, reward, done information, and image-based observations are stored in the replay buffer. These image-based expert trajectories are used to pre-train the contrastive network and address the sparse reward problem of RL.



**FIGURE 2.** Collect expert demonstration. Step 1: train an expert agent with physical state-based observation. Step 2: Record image observations using the imitation environment. This environment executes the action from the expert agent while recording the image-based observation, action, and sparse reward.

## B. CONTRASTIVE UNSUPERVISED REINFORCEMENT LEARNING

After obtaining the demonstrations from the expert agent in a replay buffer, we use them to train two backbone models, the query and key encoders, as shown in the CURL module section of Figure 1. The trained query encoder is then used as the observation for the SAC-based RL. A detailed discussion of each step within CURL is presented below:

### 1) TRAINING BACKBONE ENCODERS

The expert demonstrations are used to train the key and query backbone encoders. Initially, both encoders are initialized with the same parameters. During training, the parameters

of the key encoder are updated by constraining it with the InfoNCE loss function [31]. A batch of image observations  $X = \{x_1, \dots, x_k\}$  is sampled from the replay buffer. The data augmentation is performed on every image in the batch with the random crops to generate queries  $Q = \{q_1, \dots, q_K\}$  and keys  $K = \{k_1, \dots, k_K\}$ . The InfoNCE loss is defined in equation 9.

$$L_{q_i} = -\log \frac{\exp(q_i^T W k_i)}{\sum_{j=1}^K \exp(q_i^T W k_j)} \quad (9)$$

where:  $q^T W k$  is the bi-linear inner-product similarity, and  $W$  is a learned parameter matrix.

To minimize this InfoNCE loss, the bi-linear inner-product similarity of the query  $q_i$  and key  $k_i$  is maximized if the query and key come from the same image. Through this, they will be encoded close in the low-dimension representation space. Once the parameters in the key encoder are updated, the query encoder is updated with momentum:

$$\theta_k = m\theta_k + (1 - m)\theta_q \quad (10)$$

where:  $\theta_k, \theta_q$  are the network parameters of key and query, respectively.

## 2) TRAIN SAC WITH ENCODER

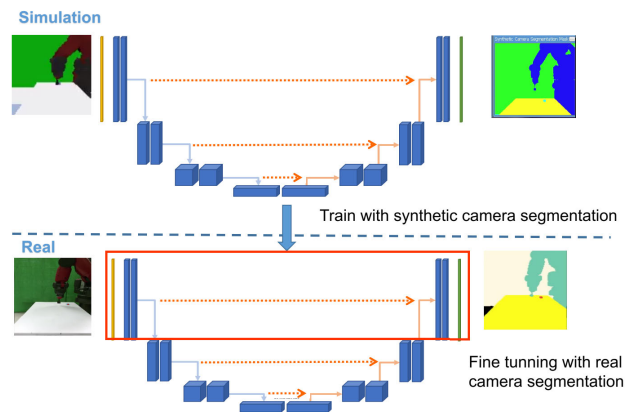
After tuning the key and query encoders, the SAC-based RL agent is trained using the query encoder, as shown in the CURL section of Figure 1. Since the agent is trained with image-based observations, defining a dense reward during grasping is challenging. The only reward is the lifting reward (with a reward of 1) when the object is elevated more than 5cm above the table.

## C. TRANSFER TO REAL ENVIRONMENT

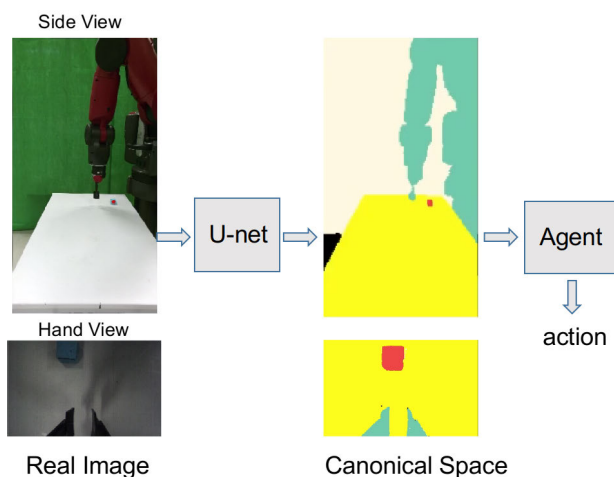
To achieve robust task transfer, in other words, to utilize the well-trained agent from simulation, the real-world images need to be mapped into the proposed canonical space via segmentation.

We chose U-Net [32] for semantic segmentation of real images. U-Net is designed for fast and accurate image segmentation. As depicted in Figure 3, it comprises three types of layers: max-pooling (blue arrow), up-sampling (yellow arrow), and copy-crop concatenation (red arrow). U-Net has been extensively used in biomedical image segmentation due to its ability to provide robust, precise segmentation with a reasonable number of training samples. However, training a U-Net still necessitates hundreds to thousands of images with segmentation masks, which can be time-consuming to label.

We utilize another transfer learning approach for segmentation to reduce the demand for real data. This transfer is the observation-level Sim2Real. We first pre-train the U-Net with synthetic images and segmentation masks provided directly by the simulation environment. Then, we fine-tune the U-Net with a small amount of real data, as depicted in Figure 3. To efficiently use this limited real data set, we must determine which layers need to be fine-tuned in the U-Net. In [33],



**FIGURE 3.** Train segmentation for the real world scans. First, U-nets are trained using synthetic RGB scans and ground truth masks from the simulation environment. Then, the networks' shadow layers (shown in the red box) are fine-tuned with real data from the side and hand views.

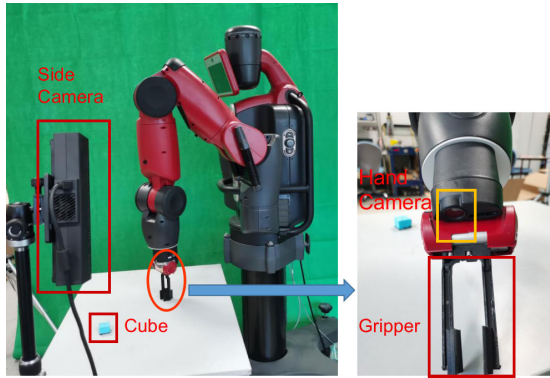


**FIGURE 4.** Control the real robot with segmented image. First, segment the real camera view with U-Net. Then the robot is controlled with the transferred agent based on the segmented image.

Amiri investigates the performance of U-Net on a new dataset when different layers are fine-tuned. We find that in our case, tuning the shallow layers (the layers in the red box in Figure 3) yields the best performance on the real image data set, which is consistent with their findings.

Since the primary differences between synthetic and real images lie in texture, rendering, and lighting, these features are extracted by the shallow layers of the U-Net. Thus, by fine-tuning these layers, U-Net can better adapt to the real data set. To achieve robust segmentation, a mere 15 side-view and 25 hand-view images are sufficient for fine-tuning each U-Net pair. The segmentation masks for this small data set are generated using the Matlab ImageLabeler tool.

Once the U-Net segments the real image observation in real-time, the proposed framework transfers the RL agent (trained in the simulation environment) to control the actual Baxter robot for the desired manipulation tasks, as illustrated in Figure 4. The original image dimension of hand view and



**FIGURE 5.** Experiment setup. A Baxter robot with parallel two fingers gripper and two RGB cameras from side view and hand view.

side view are  $3 \times 540 \times 960$  and  $3 \times 640 \times 400$ . These images are reduced to the dimension of  $3 \times 96 \times 96$  before putting into the image-based RL agent.

#### IV. EXPERIMENT

In this section, we evaluate the performance of our proposed method on both simulated and real Baxter robots. First, we describe the task and experimental setup. Next, we provide details of the training process in the simulation environment and compare our method's performance with vanilla CURL and CURL with domain randomization (DR-CURL) under various camera position calibration errors, illumination, and light color conditions. Finally, we test our method on the Baxter robot with a dark hand view in the cube lifting task.

##### A. HARDWARE SETUP

The hardware setup is shown in Figure 5. The task is to lift the cube above the table more than 5cm. This setup consists of a 7-DoF Baxter manipulator (Rethink Robotics, US) equipped with a parallel two-finger gripper, a side camera (Microsoft Kinect2) featuring a resolution of  $540 \times 960$ , and an original Baxter camera on hand with a resolution. Both cameras have a sampling rate of 30 Hz. A cube-shaped object with the dimension ( $50 \times 50 \times 50$ )mm is placed on the table. The manipulator's workspace is the table's surface with dimensions ( $600 \times 800$ )mm. The exchange of messages between all the different devices is managed by the ROS framework, an open-source Robot Operating System running on a personal computer with an i5-1060 CPU, 32GB RAM, and NVIDIA RTX 3070 GPU.

##### B. SIMULATION EXPERIMENT

To accomplish successful Sim2Real transfer, we must create a simulation environment in Pybullet that closely resembles the real world. We utilize the robot, table, and camera URDF files to construct the simulation environment. Additionally, we use the intrinsic and extrinsic camera parameters to enable accurate camera simulation. Camera calibration tools, such as MoveIt and OpenCV, can be employed to obtain these

**TABLE 1.** Parameters for SAC and encoder.

name	value	name	value	name	value
$\gamma$	0.99	$lr$	$1e-3$	layers	4
$\alpha$	0.5	$\alpha_{lr}$	$1e-4$	out_dim	50
$B$	256	$\tau$	0.01	filters	32

parameters. One episode contains 100 steps. When the agent reaches the maximum step, the episode will stop. The goal of RL is to maximize the accumulated reward in 100 steps.

The simulation and demonstration collection are conducted in the Pybullet. We present implementation details regarding expert SAC and the self-contrastive encoder. We compare the performance of our proposed Seg-CURL with vanilla CURL and CURL with domain randomization (DR-CURL) under various camera position calibration errors, illumination, and light color conditions. We demonstrate that segmented images contain sufficient information to complete the required task robustly.

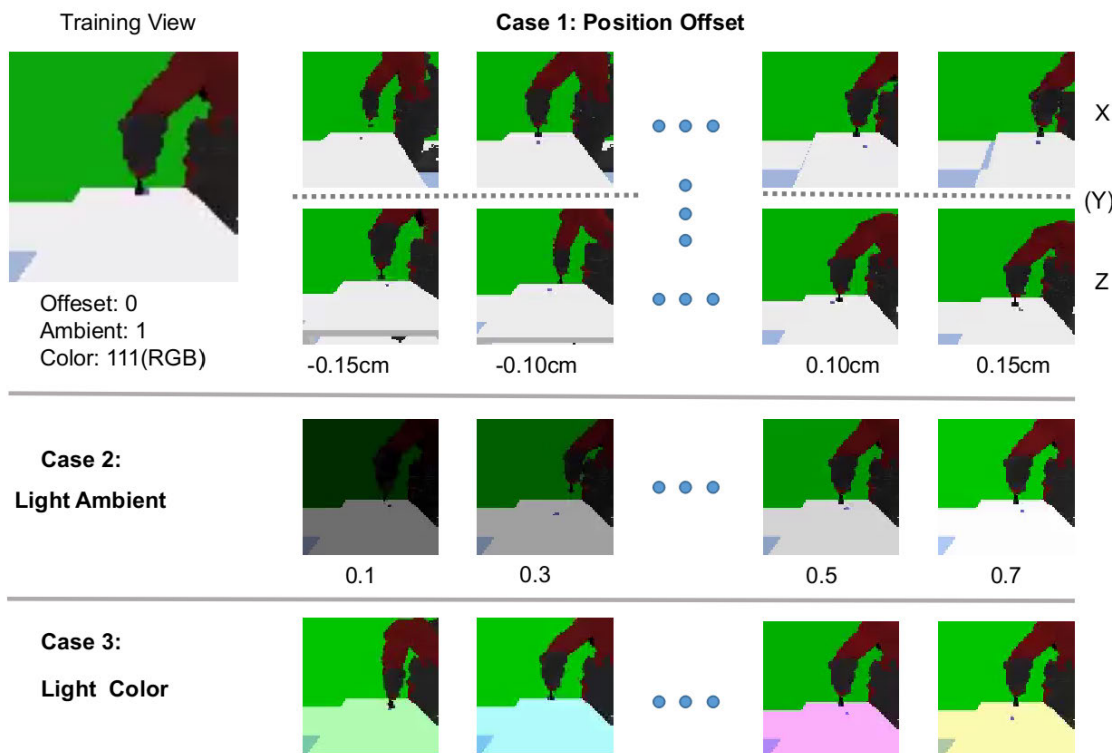
As described in the methods section, we first need to train an expert agent that has a physics state-based observation, as in Eq. (2), and a predefined dense reward, as expressed in Eq. (1). The hyperparameters of the SAC (the expert agent) are shown in Table 1, where  $\gamma$  is the reward discount factor,  $lr$  is the learning rate of the  $Q$  and  $V$  networks.  $\alpha$  is the initial temperature parameter,  $\alpha_{lr}$  is the corresponding learning rate.  $B$  is the batch size, and  $\tau$  is the smoothing coefficient for the target  $Q$  network. These hyperparameters are chosen in a manner consistent with Zhan's work [5]. They can be optimized using random search, grid search, or Bayesian optimization, aided by open-source tools such as Ray Tune [34] and Optuna [35]. The training takes approximately 2 hours (500 episodes) to produce a stable agent, as illustrated in Figure 7.

Once we get the physic state-based expert agent, we collect 20 trajectories from this agent with segmented observation. Subsequently, the self-contrastive learning encoders are pre-trained using the segmented images stored in the replay buffer. The parameters of the SAC agent are updated based on the encoder's representation when the robot interacts with the environment. The hyperparameters of the image-based SAC remain consistent with the parameters listed in Table 1. The encoder consists of four layers, each comprising a filter with 32 channels. The training process takes approximately three hours (800 episodes) to achieve a stable agent, as depicted by the blue line in Figure 7.

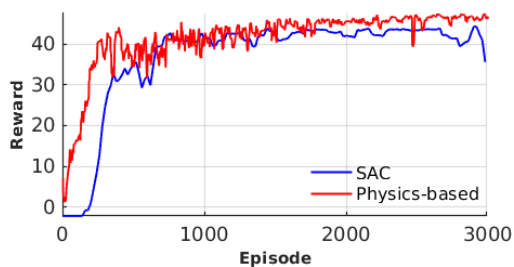
In order to systematically evaluate our method, we test Seg-CURL under three types of Sim2Real gaps, as depicted in Figure 6.

**1) Camera Position Offset:** This represents the position calibration error. We evaluate RL agents on the X, Y, and Z axes with position offset ranges from -0.15 cm to 0.15 cm.

**2) Illumination of Light:** This effect can be adjusted by the light ambient coefficient (LAC) in the Pybullet simulator. As shown in Figure 6 case 2, the LAC is set from 0.1 to 0.7.



**FIGURE 6.** Experiments in the simulation. **Case 1:** Test the grasping in camera position offset. The camera position calibration error ranges from  $-0.15\text{cm}$  to  $0.15\text{cm}$  in the X, Y, and Z axis. **Case 2:** Test grasping on different illumination. The range for the light ambient coefficient is 0.1 to 0.7, corresponding from dark to bright. **Case 3:** Test on different light colors. RL agents are tested on 8 colors, where color 0 presents gray (RGB: 000), color 1 presents purple (RGB: 001), and so on.



**FIGURE 7.** Training process. The red line is the training reward for the physic feature based agent. The blue line is the reward for the segmented image-based agent.

**3) Object Texture:** We simply alter the light color to demonstrate this effect. We select eight kinds of colors, where 0 represents gray (RGB 000), 1 represents purple (RGB 001), and so forth.

The baselines we have chosen are vanilla CURL and CURL with domain randomization (DR-CURL). The vanilla CURL is trained using RGB views with a 0 position offset, a LAC value of 1, and pure white light (RGB 111). In contrast, for DR-CURL, the camera position offset is randomly set within the range of  $[-0.05, 0.05]$  at each step. The LAC is randomly set within the range of  $[0.3, 0.7]$ , and the light color is randomly set from  $[0, 0, 0]$  to  $[1], [1], [1]$  at each step.

1) EVALUATION IN POSITION OFFSET

As illustrated in Figure 8, all RL agents achieve a success rate of approximately 0.95 when the camera position offset is less than  $0.05\text{ cm}$  in the X, Y, and Z axes. This robustness results from the data augmentation in the encoding process, wherein  $84 \times 84$ -pixel images are randomly cropped from the original  $96 \times 96$ -pixel images. Consequently, the learned representation vector remains invariant to changes at the image edges. DR-CURL outperforms both CURL and Seg-CURL, as it is trained with a wider field of view by randomizing the camera position. In most cases (except in the X-axis), our Seg-CURL attains a performance similar to that of DR-CURL, even though it is trained with a narrower field of view. However, Seg-CURL is more sensitive to camera position changes in the X-axis when the offset exceeds 0.05. It delivers the worst performance when the offset is  $+0.1\text{ cm}$  and  $+0.15\text{ cm}$  in the X-axis.

2) EVALUATION IN LIGHT AMBIENT

Figure 9 presents the performance of RL agents under various illumination conditions. Our Seg-CURL outperforms other RL agents, achieving a success rate higher than 0.95 in all conditions. DR-CURL also attains a success rate of approximately 1 when the light ambient is greater than 0.1. However, it can only achieve a success rate of around 0.5 when the light ambient is 0.1, a case exclusively included in domain

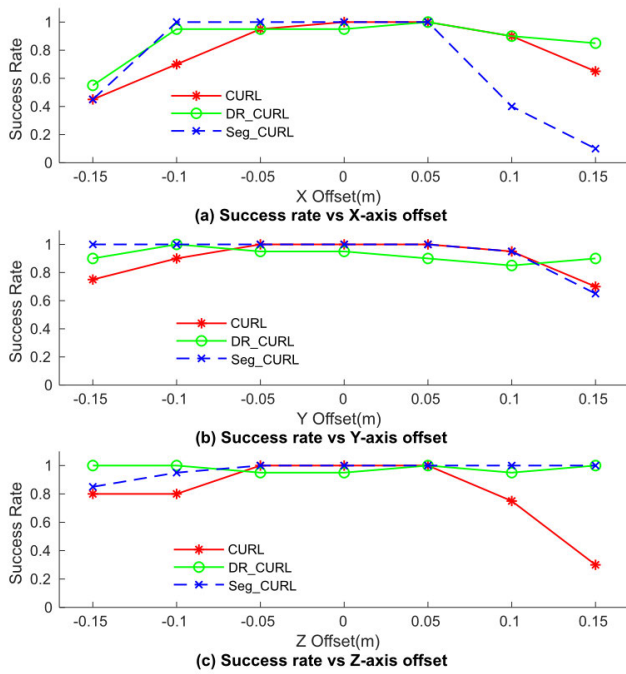


FIGURE 8. Grasping success rate in different camera position offset.

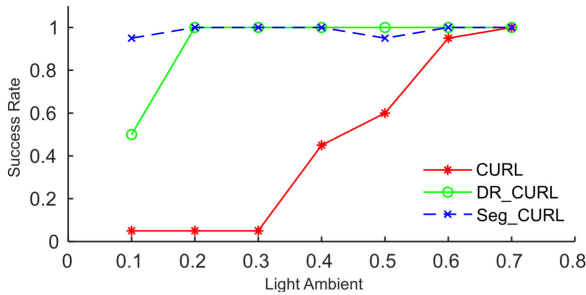


FIGURE 9. Grasping success rate in different light ambient coefficient.

randomization. The vanilla CURL is highly sensitive to the LAC and can only achieve a performance similar to Seg-CURL when the LAC value is greater than 0.6.

### 3) EVALUATION IN LIGHT COLOR

Lastly, we evaluate the performance of RL agents under various light colors, as demonstrated in Figure 10. Our method delivers the best performance, as the segmented image is invariant to color changes in lighting. DR-CURL and CURL do not exhibit stable performance across various light colors. CURL achieves a success rate of only 0.25 in light color 1 (RGB: 001), while DR-CURL attains a success rate of merely 0.1 in light color 6 (RGB: 110).

### C. TRANSFER TO THE REAL ROBOT

In this section, we report the performance of the transferred agent on a real robot. The experiment setup details are described in Section IV-A. There are two RGB images from side-view and hand-view perspectives, respectively. Figure 11 displays the real images and their corresponding

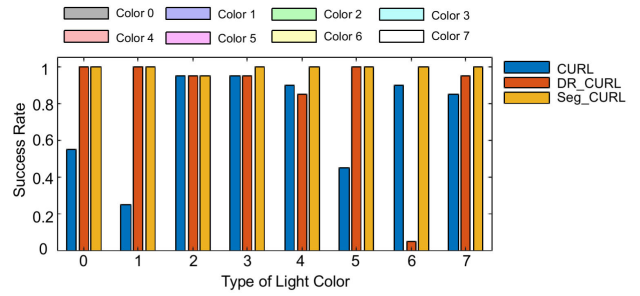


FIGURE 10. Grasping success rate in various light colors. The Sim2Real performance is tested on 8 kinds of light.

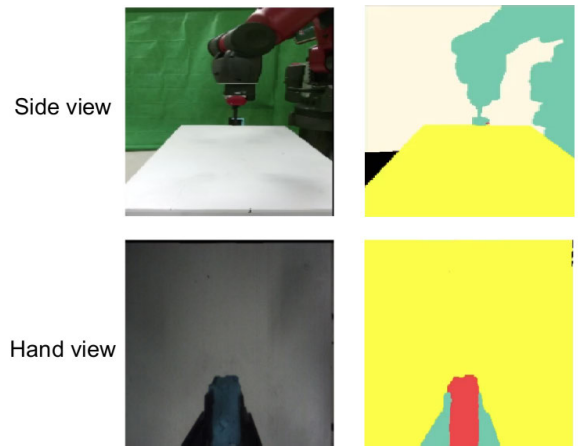


FIGURE 11. Real image, segmented image.

segmented images. It takes around 18 minutes to train the U-net on our customized data set that contains 5000 pairs of synthetic RGB and segmentation masks. From the real image view, we can observe that some shadows change during the robot’s movement. Furthermore, the hand camera of the Baxter robot can only provide dark images, which increases the task’s difficulty.

Table 2 presents the success rate of tasks when transferring the RL agents to the real robot using zero-shot learning. We randomly place the cube in a 15cm × 15cm area below the gripper. The vanilla CURL can only touch the cube with a success rate of 3/20. The DR\_CURL can achieve a success rate of 12/20 and 7/20 on the task touch and lifting task respectively. Our Seg-CURL demonstrates good Sim2Real performance, achieving 20/20 on touching and 16/20 on lifting in zero-shot transfer to the real world. It is not surprising that vanilla CURL and DR-CURL exhibit poor performance in Sim2Real, as the real view domain differs from the simulation. Real images encompass varying texture details, shadows, and lighting conditions. Particularly when the manipulator is near the table, there will be irregular shadows that are not present in the simulation environment. Consequently, the transferred encoder provides a different representation than the simulated view, even when the real robot and real object are in the same position as the simulated ones. Without bridging the gap between the image



**TABLE 2. Success rate for different algorithms in Sim2Real.**

Success rate	Seg-CURL	DR_CURL	CURL
touch rate	20/20	12/20	3/20
grasp rate	16/20	7/20	0/20

representations of simulation and reality, the transferred actor cannot take optimal action in the real world.

## V. CONCLUSION

This work introduces the Seg-CURL framework for reinforcement learning-based visual grasping. We employ segmented images as the canonical observation space. Compared to GAN-based Sim2Real methods [11], [16], Seg-CURL necessitates only about 20 real images with segmentation masks to control a real robot. In the majority of cases, Seg-CURL attains higher success rates than vanilla CURL and DR-CURL in the object-lifting task under various light colors, illumination levels, and camera position offsets. It can be applied to multi-object grasping by segmenting different objects with different masks. In the future, we will also test this method on the long horizon task, such as pick and place, and peg-in-hole. The main drawback of our method is the loss of information about texture when transferring RGB observation to segmented observation. Our method is suitable for manipulating objects where texture does not play a crucial role in the task.

## REFERENCES

- [1] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation," 2018, *arXiv:1806.10293*.
- [2] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Sci. Robot.*, vol. 4, no. 26, pp. 1–20, Jan. 2019.
- [3] F. Munguia-Galeano, S. Veeramani, J. D. Hernández, Q. Wen, and Z. Ji, "Affordance-based human-robot interaction with reinforcement learning," *IEEE Access*, vol. 11, pp. 31282–31292, 2023.
- [4] S. Song, A. Zeng, J. Lee, and T. Funkhouser, "Grasping in the wild: Learning 6DoF closed-loop grasping from low-cost demonstrations," *IEEE Robot. Autom. Lett.*, vol. 5, no. 3, pp. 4978–4985, Jul. 2020.
- [5] A. Zhan, R. Zhao, L. Pinto, P. Abbeel, and M. Laskin, "Learning visual robotic control efficiently with contrastive pre-training and data augmentation," 2020, *arXiv:2012.07975*.
- [6] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 1395–1476, 2021.
- [7] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2020, pp. 737–744.
- [8] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers Robot. AI*, vol. 9, pp. 1–19, Apr. 2022.
- [9] R. Burgert, J. Shang, X. Li, and M. Ryoo, "Neural neural textures make Sim2Real consistent," 2022, *arXiv:2206.13500*.
- [10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 23–30.
- [11] 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. (CVPR)*, Jun. 2019, pp. 12619–12629.
- [12] A. Srinivas, M. Laskin, and P. Abbeel, "CURL: Contrastive unsupervised representations for reinforcement learning," 2020, *arXiv:2004.04136*.
- [13] S. Pareek, H. Nisar, and T. Kesavadas, "AR3n: A reinforcement learning-based assist-as-needed controller for robotic rehabilitation," 2023, *arXiv:2303.00085*.
- [14] E. Y. Puang, K. P. Tee, and W. Jing, "KOVIS: Keypoint-based visual servoing with zero-shot sim-to-real transfer for robotics manipulation," 2020, *arXiv:2007.13960*.
- [15] R. Jeong, Y. Aytar, D. Khosid, Y. Zhou, J. Kay, T. Lampe, K. Bousmalis, and F. Nori, "Self-supervised sim-to-real adaptation for visual robotic manipulation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 2718–2724.
- [16] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, "RL-CycleGAN: Reinforcement learning aware simulation-to-real," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11154–11163.
- [17] A. Pashevich, R. Strudel, I. Kalevatykh, I. Laptev, and C. Schmid, "Learning to augment synthetic images for Sim2Real policy transfer," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 2651–2657.
- [18] Z.-W. Hong, Y.-M. Chen, H.-K. Yang, S.-Y. Su, T.-Y. Shann, Y.-H. Chang, B. H.-L. Ho, C.-C. Tu, T.-C. Hsiao, H.-W. Hsiao, S.-P. Lai, Y.-C. Chang, and C.-Y. Lee, "Virtual-to-real: Learning to control in visual semantic segmentation," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 4912–4920.
- [19] G. Du, K. Wang, S. Lian, and K. Zhao, "Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: A review," *Artif. Intell. Rev.*, vol. 54, no. 3, pp. 1677–1734, Mar. 2021.
- [20] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5049–5059.
- [21] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 857–876, Jan. 2023.
- [22] A. V. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1747–1756.
- [23] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 5708–5717.
- [24] M. Kim, J. Tack, and S. J. Hwang, "Adversarial self-supervised contrastive learning," in *Proc. Neural Inf. Process. Syst. (NIPS)*, 2020, pp. 2983–2994.
- [25] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 1597–1607.
- [26] Y. Liu, K. Wang, L. Liu, H. Lan, and L. Lin, "TCGL: Temporal contrastive graph for self-supervised video representation learning," *IEEE Trans. Image Process.*, vol. 31, pp. 1978–1993, 2022.
- [27] M. Thota and G. Leontidis, "Contrastive domain adaptation," 2021, *arXiv:2103.15566*.
- [28] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," 2018, *arXiv:1812.05905*.
- [29] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 4, 2018, pp. 2587–2601.
- [30] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "Robosuite: A modular simulation framework and benchmark for robot learning," 2020, *arXiv:2009.12293*.
- [31] A. Van Den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2018, *arXiv:1807.03748*.
- [32] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," 2015, *arXiv:1505.04597*.
- [33] M. Amiri, R. Brooks, and H. Rivaz, "Fine-tuning U-Net for ultrasound image segmentation: Different layers, different outcomes," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 67, no. 12, pp. 2510–2518, Dec. 2020.
- [34] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," 2018, *arXiv:1807.05118*.
- [35] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2623–2631.



**BINZHAO XU** received the B.S. degree from China Three Gorges University, and the M.S. degree (Hons.) in electrical engineering from Queen's University Belfast, in 2018. He is currently pursuing the Ph.D. degree in robotics with the Khalifa University of Science and Technology. His research interests include reinforcement learning and learning from demonstration.



**TAIMUR HASSAN** received the B.S. degree in computer engineering from Bahria University, Islamabad, Pakistan, in 2013, the M.S. degree in computer engineering from the University of Engineering and Technology (UET), Taxila, Pakistan, in 2015, and the Ph.D. degree in computer engineering from the National University of Sciences and Technology (NUST), Islamabad, in 2019.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Abu Dhabi University, United Arab Emirates. Prior to that, he was a Postdoctoral Fellow with the Khalifa University Center for Autonomous Robotic Systems (KUCARS) and the Center for Cyber-Physical Systems (C2PS), Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates. He has worked on many local and foreign funded research projects as a principal investigator, a co-principal investigator, and a lead scientist/engineer. His research interests include robotic vision, medical imaging, deep learning, signal processing, and computer vision. He was a recipient of various national and international awards.



**IRFAN HUSSAIN** received the B.E. degree in mechatronics engineering from Air University, Pakistan, the first M.S. degree in mechatronics engineering from the National University of Sciences and Technology, Pakistan, the second master's degree in automatica and control technologies from Politecnico di Torino, Italy, and the Ph.D. degree in robotics from the University of Siena, Italy.

From 2008 to 2011, he was an Assistant Manager of engineering with Trojans, Pakistan. He was a Research Assistant with Gyeongsang National University, South Korea, from 2012 to 2013. He was a Visiting Researcher with Centro Ricerche Fiat (CRF), Italy. He was a Postdoctoral Researcher with the Robotics Institute, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates, and the Siena Robotics and System Laboratory (SIRSLab), Italy. He is currently an Assistant Professor of robotics and mechanical engineering with Khalifa University, Abu Dhabi. He is the author of one book, more than 70 articles, and three inventions. His research interests include embodied intelligence, exoskeletons, extra robotic limbs, soft robotic hands, wearable haptics, grasping, and manipulation. He is an Associate Editor of *Proceedings of the Institution of Mechanical Engineers—C: Journal of Mechanical Engineering Science*, a Research Topic Editor for a Special Issues on Wearable Robots and Sensorimotor Interfaces: Augmentation, Rehabilitation, Assistance or Substitution of Human Sensorimotor Function (ID 21096) of *Frontiers in Neurorobotics*, and a Guest Editor for the Special Issue on Design and Development of Vision-Based Tactile Sensors of *Sensors* (ISSN 1424-8220). He is an Associate Editor of RAS/EMBS BioRob, in 2018, ICRA, in 2021, ICRA, in 2022, and IEEE/RAS ICRA, in 2023.

• • •