IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

Kalman Filter-based One-shot Sim-to-Real Transfer Learning

Qingwei Dong[1,2,3,4], Peng Zeng[1,2,3,*], Guangxi Wan[1,2,3,4], Yunpeng He[1,2,3,4], Xiaoting Dong[1,2,3,4]

***Abstract*—Deep reinforcement learning algorithms offer a promising method for industrial robots to tackle unstructured and complex scenarios that are difficult to model. However, due to constraints related to equipment lifespan and safety requirements, acquiring a number of samples directly from the physical environment is often infeasible. With the development of increasingly realistic simulators, it has become feasible for industrial robots to acquire complex motion skills within simulated environments. Nonetheless, the "reality gap" frequently results in performance degradation when transferring policies trained in simulators to physical systems. In this paper, we treat the reality gap between a physical environment (target domain) and a simulated environment (source domain) as a Gaussian perturbation and utilize Kalman filtering to reduce the discrepancy between source and target domain data. We refine the source domain controller using target domain data to enhance the controller's adaptability to the target domain. The efficacy of the proposed method is demonstrated in reaching tasks and peg-in-hole tasks conducted on PR2 and UR5 robotic platforms.**

***Index Terms*—Transfer Learning, Reinforcement Learning, Kalman filter, Reality Gap, Sim-to-Real.**

I. Introduction

In industry, numerous robots are programmed to fulfill specific needs in distinct scenarios. Manually crafting trajectories for a particular task is a labor-intensive, intricate process that is prone to errors [1]. When a robot's operating environment contains uncertainties such as dynamic obstacles [2], randomly placed items, workpieces of varying geometric shapes, and wear and tear of robot components, the robot's workspace becomes complex and cannot be modeled or programmed in advance. To circumvent collisions and successfully complete tasks, industrial robots require adaptive capabilities to handle unstructured and uncertain environments. Deep reinforcement learning (DRL) algorithms, which have demonstrated remarkable progress in highly flexible and unmodelable complex scenarios, are an ideal solution.

Manuscript received: May 10, 2023; Revised: August 9, 2023; Accepted: October 19, 2023. This paper was recommended for publication by Editor Jens Kober upon evaluation of the Associate Editor and Reviewers' comments. This research was funded in part by the State Key Laboratory of Robotics of China (2023-Z15) and in part by the National Natural Science Foundation of China under Grants U1908212, 92067205 and 92267205. *(Corresponding author: Peng Zeng)*

Qingwei Dong, Peng Zeng, Guangxi Wan, Yunpeng He and Xiaoting Dong are with [1]the State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences; [2]Key Laboratory of Networked Control Systems, Chinese Academy of Sciences; [3]Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences; Qingwei Dong, Guangxi Wan, Yunpeng He and Xiaoting Dong are also with [4]the University of Chinese Academy of Sciences. ShenYang 110016, China (e-mail: dongqingwei@sia.cn; zp@sia.cn; wanguangxi@sia.cn; heyunpeng@sia.cn; dongxiaoting@sia.cn).

Digital Object Identifier (DOI): see top of this page.

In recent years, DRL algorithms have accomplished remarkable feats. For instance, DRL-trained agents can defeat the world's top Go players [3], master Atari games directly from pixels with superhuman performance [4], exhibit impressive performance in multiagent interactions in StarCraft II [5], and successfully navigate various challenging obstacle environments such as Quadruped, Planar Walker, and Humanoid [6]. However, these successes are predominantly restricted to 2D and 3D video games and virtual environments, with learning complex strategies on physical systems remaining a significant challenge [7] [8]. Google's team proposed a large-scale algorithm for hand-eye coordination, showcasing the prowess of DRL algorithms in physical robot grasping tasks. The experiment entailed approximately 800,000 grasps over two months, employing 6-14 parallel robots [9]. Although the results were striking, directly training on physical robots may not be a prudent choice due to device lifespan limitations and safety requirements. Consequently, researchers are increasingly focusing on the sim-to-real domain for robots to learn intricate tasks [10].

Acquiring complex robotic motor skills in simulation environments has been proven to be rapid and secure. However, this transfer from simulation to reality is effective only when simulators can accurately model the actual robot and environment [11]. To address this gap in transfer learning, various methods have been invented, including zero-shot transfer, domain randomization [12], domain adaptation [13], and learning with disturbances [14]. These approaches concentrate on enhancing simulators, randomizing simulations, unifying the feature spaces of source and target domains, and introducing environmental disturbances. These methods can be categorized into two categories. The first category consists of methods that randomize environmental parameters to emulate real environments, enabling sim-to-real transfer. This approach requires substantial computational resources, resulting in high computational costs. The second category consists of methods that rely on designing robust controllers with empirical rules in the target domain to ensure their reliability and reduce the gap between the source and target domains. However, this approach leads to performance degradation in the target domain, and the performance of the controllers is suboptimal.

We aim to reduce the computational cost of methods of the first category while minimizing the reduction in controller performance in the target domain compared to methods of the second category. The crux of the sim-to-real problem lies in the discrepancies in dynamic models between the source and target domains. This difference can be regarded as a disturbance with a specific covariance [15] [16] and mean,

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

which can be simulated using observed state disturbances. In conventional control, filtering methods can be employed to eliminate this disturbance. Consequently, we examine whether filtering techniques can be utilized to rectify this discrepancy. As a result, we transform the transfer problem into a control issue and employ filtering methods to minimize the disparity between the source and target domains.

Our primary contributions are as follows: (1) A promising perspective. We treat the differences between the source and target domains as perturbations with specific covariance and mean values and address these differences using Kalman filtering; (2) KFOTL method. We introduce a unique sim-to-real approach that, by considering the differences between the source and target domains, refines the source domain policy to improve its transferability; (3) Diverse experimental validation. We demonstrate the effectiveness of our proposed method in reaching tasks and peg-in-hole tasks using UR5 and PR2 robots.

II. RELATED WORK

To reduce the demand for samples from real environments, various novel methods have been proposed. A method combining model predictive control (MPC) [17] with learning task representations adapts policies from simulations for unknown tasks using pretrained operations as predictive tools. Reference [7] introduced a neural-augmented simulation (NAS) method for policy learning. This method uses real robot data to train a recurrent network, predicting the simulation-reality gap and improving transferability through reduced errors. Hanna, J.P. [18] presented the Grounded Action Transformation (GAT) algorithm, refining simulators to align more closely with reality, thus enhancing the performance of robotic control policies in real-world situations. Justin Fu [19] formulated a model-based reinforcement learning algorithm, blending prior task knowledge with online dynamic model adaptation to craft effective control strategies for intricate robotic controls.

However, these methods all incur significant computational costs when transferring policies to the real world. For instance, the dynamics randomization (DR) [20] method can generalize policies to the real world without training on physical systems, it entails substantial training costs. In a pushing object experiment, the algorithm needs to randomize 95 dynamic parameters and utilize approximately 100 million samples, requiring approximately 8,000 iterations. Its computational load involves training on a 100-core cluster for 8 hours.

In fact, such extensive training is computationally expensive and necessitates substantial simulation-based training. For some tasks, acceptable performance may be possible without a perfect pretrained model. Krishan Rana proposed a multiplicative controller fusion (MCF) [21] method based on this idea. This approach designs a prior controller capable of performing preliminary obstacle avoidance tasks through traditional methods, albeit with longer trajectories and lower efficiency, resulting in a suboptimal solution. A better controller is then trained based on this prior controller, and both are combined during deployment to the target domain. When state uncertainty is low, control actions are closer to the trained policy, while control actions are closer to the prior when state uncertainty is high. This method is a relatively conservative approach since it reverts to the suboptimal solution of the prior controller when the current policy's uncertainty is high. In contrast, we aim to propose a proactive method that directly constructs the reality gap and narrows the difference via filtering.

For scenarios with relatively minor dynamic parameter changes, a block-shaped mass rigidly attached to a robotic arm link causing a change in the dynamics. If the only difference between the source domain and the target domain is the presence or absence of this mass block, then the difference is traceable. The performance of the trained source domain policy in the target domain can be understood as the result of disturbances caused by the differences. Therefore, a possible hypothesis is that if the trajectory differences between the source and target domains are treated as Gaussian noise with a specific covariance and mean, the differences can be corrected with a filtering approach on the trained source domain policy. Consequently, with minimal corrections and limited pretraining, the source domain policy can be adaptively transferred to the target domain.

III. PRELIMINARIES

In this paper, we denote the agent's state at time step t as $\boldsymbol{x}_t$ and represent the agent's action at time step t by $\boldsymbol{u}_t$. The stochastic policy trained in the source domain is defined as the source domain policy, denoted as $\pi_{src}(\boldsymbol{u}_t|\boldsymbol{x}_t^{src};\boldsymbol{\theta}_{src})$. The policy adapted for the target domain is referred to as the target domain policy, denoted as $\pi_{tgt}(\boldsymbol{u}_t|\boldsymbol{x}_t^{tgt};\boldsymbol{\theta}_{tgt})$. Here, $\boldsymbol{x}_t^{src}$ and $\boldsymbol{x}_t^{tgt}$ represent the agent's states in the source and target domains at time step t, respectively. $\boldsymbol{\theta}_{src}$ and $\boldsymbol{\theta}_{tgt}$ denote the parameters of the source and target domain policies, respectively. In the subsequent text, the two policies are abbreviated as $\pi(\boldsymbol{\theta}_{src})$ and $\pi(\boldsymbol{\theta}_{tgt})$. In the proposed method, the source domain policy is obtained through a reinforcement learning method, and the target domain policy is obtained by adapting the source domain policy.

We formulate the learning of robotic motion skills in the source domain as a policy search problem, aiming to find the optimal parameter $\boldsymbol{\theta}_{src}^*$ that minimizes the expected loss of the trajectory $E_{\pi_{src}}[l(\boldsymbol{\tau}_{src})]$.

$$\boldsymbol{\theta}_{src}^* = argmin_{\boldsymbol{\theta}_{src}} E_{\pi_{src}}[l(\boldsymbol{\tau_{src}})] \tag{1}$$

In this equation, $\boldsymbol{\theta}_{src}$ represents the parameters of the probability distribution of action $\boldsymbol{u}_t$ taken by the robot in source state $\boldsymbol{x}_t^{src}$. $\boldsymbol{\tau}_{src}$ denotes a fixed-length trajectory of robot states and actions in the source domain. $l(\tau_{src})$ represents the loss of a trajectory from the source domain.

$\boldsymbol{\tau}_{src} = \{\boldsymbol{x}_0^{src}, \boldsymbol{u}_0^{src}, \boldsymbol{x}_1^{src}, \boldsymbol{u}_1^{src}, ..., \boldsymbol{x}_T^{src}, \boldsymbol{u}_T^{src}\}, T \in \mathbb{N}$.

Here, $\mathbb{N}$ represents the set of natural numbers. The trajectory distribution $p(\boldsymbol{\tau}_{src})$ is jointly described by the stochastic policy $\pi_{src}(\boldsymbol{u}_t|\boldsymbol{x}_t)$ and the system dynamics model $p_{src}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. The system dynamics describe the process in which, at time step t, the robot takes action $\boldsymbol{u}_t^{src}$ in the source domain state $\boldsymbol{x}_t^{src}$, and the system transitions to the next source domain state $\boldsymbol{x}_{t+1}^{src}$. Given that the linear approximation of the dynamics at each time step provides the required precision for our design, the system dynamics in the source domain are described using a time-varying linear Gaussian

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

distribution, $p_{src}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = N(f_{\boldsymbol{x}t}\boldsymbol{x}_t + f_{\boldsymbol{u}t}\boldsymbol{u}_t + f_{\boldsymbol{c}t}, \boldsymbol{F}_t)$, where $f_{\boldsymbol{x}t}$, $f_{\boldsymbol{u}t}$ and $f_{\boldsymbol{c}t}$ are the mean parameters of the system dynamics model, and $\boldsymbol{F}_t$ represents the variance. To enhance the generalization and representation capabilities of the stochastic policy, we express the stochastic policy with a neural network model.

Considering the drawbacks of slow convergence and lengthy learning cycles of direct policy search methods, we introduce the GPS algorithm [22]–[24] to accelerate the learning of source domain policies.

IV. METHOD

A. Architecture of KFOTL

Discrepancies in trajectories between the source and target domains arise due to the accumulation of system model errors. For instance, at the initial timestep, the robots in both the source and target domains possess identical initial states, denoted as $\boldsymbol{x}_0^{src}$ and $\boldsymbol{x}_0^{tgt}$, respectively. Consequently, the control actions transmitted to the robot's joints by the same controller will be identical in the next time step. However, inherent differences between the source and target domains lead to the robot in the source domain reaching a new state, $\boldsymbol{x}_1^{src}$, after executing the first set of control actions, while the robot in the target domain reaches a new state, $\boldsymbol{x}_1^{tgt}$. As the source domain policy continues to issue action commands, the trajectory discrepancy between the robots in the source and target domains grows progressively larger due to the accumulation of errors. Thus, by correcting the newly issued control actions in the target domain based on the trajectory differences between the source and target domains after each set of control actions, the robot may be able to successfully complete the designated task in the target domain.

Since acquiring the optimal source domain policy is typically challenging, the modified policy parameters should comprise two components: the approximate optimal source domain policy parameters, $\boldsymbol{\theta}^*_{src}$, and the source domain policy modification parameters, $\boldsymbol{\kappa}$, such that $\boldsymbol{\theta}_{tgt} = \{\boldsymbol{\theta}^*_{src}, \boldsymbol{\kappa}\}$. We refer to the source domain policy that allows the robot to complete the specified task in the source domain as the approximate optimal source domain policy. The target domain policy should adhere to the following mathematical form:

$$\boldsymbol{u}_t \sim \pi_{tgt}(\boldsymbol{u}_t|\boldsymbol{x}_t^{tgt}; \boldsymbol{\theta}^*_{src}, \boldsymbol{\kappa}), t \in \mathbb{N} \tag{2}$$

where $\mathbb{N}$ is the set of natural numbers. The control action, executed by the controller at time step t, adheres to the probability distribution collaboratively determined by parameters $\boldsymbol{\theta}^*_{src}$ and $\boldsymbol{\kappa}$. Our approach addresses two central challenges: acquiring the approximate optimal source domain policy and adapting the approximate optimal source domain policy for the target domain. To tackle these challenges, we develop the controller adaptive transfer structure, as shown in Figure 1.

Our proposed architecture involves learning the approximate optimal source domain policy within the source domain and subsequently adjusting this policy to derive the target domain policy in the target domain. The agent's learning process for the approximate optimal source domain policy is akin to conventional model-based GPS methods. Initially, the robot gathers trajectory sample data in the source domain through environmental interactions. These data are then employed to model the source domain dynamics. Leveraging the known source domain dynamics, LQR guides the search direction of the source domain policy into high reward regions. After iterating this procedure, the approximate optimal source domain policy is generated. Next, we deploy the approximate optimal source domain policy in the target domain. Utilizing the loss of target domain trajectories in comparison to source domain trajectories, we compute the source domain policy modification parameter $\boldsymbol{\kappa}$ via Kalman filtering. Ultimately, we derive the target domain policy that is seamlessly adapted to the target domain. In the following subsection, we delve into the details of these two processes.

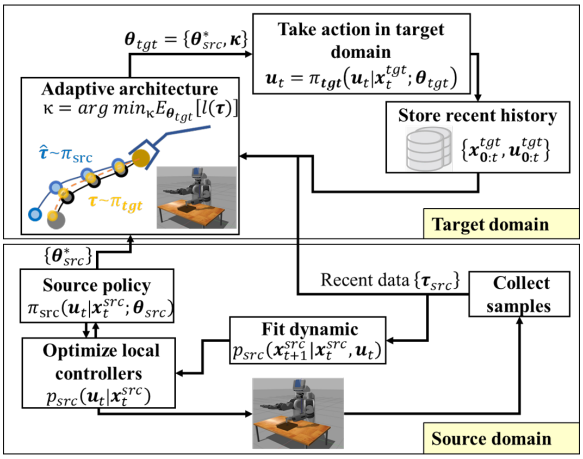

Fig. 1. Illustration of KFOTL method. The architecture is composed of source domain approximate optimal policy learning and target domain policy refinement.

B. Calculation of source domain dynamic parameters

Prior to learning this policy, it is necessary to fit the parameters of the dynamic model. The source domain trajectory samples are segmented for processing, with each timestep data point presented in the form $\boldsymbol{y}_t = [\boldsymbol{x}_t^{src}, \boldsymbol{u}_t^{src}, \boldsymbol{x}_{t+1}^{src}]$. Since the robot system has a high-dimensional state space, a large number of samples are needed to fit the dynamics well at each iteration. To mitigate sample complexity, the dynamics at every timestep in the source domain are linearized, and a multivariate linear Gaussian distribution is utilized for their representation. As there are strong correlations between the dynamics of nearby timesteps, prior information from other timesteps or previous iterations is incorporated to enhance sample utilization efficiency when computing the dynamic model of the current timestep.

Gaussian mixture models (GMMs) possess robust expressive power, making them particularly suitable for scenarios where system dynamics can be linearized in segments. Hence, we select GMM as our preferred method for fitting global dynamic models. Fitting a global dynamic model fundamentally entails discovering a suitable probability model that effectively characterizes the global data distribution. Statistically, this involves identifying a set of parameters that maximize the probability of sample occurrence. The probability model we seek is formulated as follows:

$$p(D_{src}|\xi) = \textstyle\sum_k \alpha_k p(D_{src}|\xi_k)$$

where D_{src} denotes the source domain samples, and ξ represents the model parameters generating the global samples.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

α_k is the weight coefficient, signifying the current component's contribution to the mixed model. K refers to the number of clusters in the GMM, typically an empirical value. $p(D_{src}|\xi_k)$ is the Gaussian distribution density function, with $\xi_k = (\mu_k, \sigma_k^2)$. The functional form is expressed as:

$$p(D_{src}|\xi_k) = \frac{1}{\sqrt{2\pi}\sigma_k} exp(-\frac{(D_j^{src}-\mu_k)^2}{2\sigma_k^2})$$

In this paper, we directly present the GMM solution formula, whose derivation can be found in the book [25]. In the E-step, the response of component k to the observed data D_j^{src} is calculated based on the current parameter model:

$$\hat{\gamma}_{jk} = \frac{\alpha_k p(D_j^{src}|\xi_k)}{\sum_k \alpha_k p(D_j^{src}|\xi_k)}, j = 1, ..., N; k = 1, ..., K$$

Here, j represents the sample number. In the M-step, the model parameters are computed:

$$\hat{\mu}_k = \frac{\sum_k \hat{\gamma}_{jk} D_j^{src}}{\sum_k \hat{\gamma}_{jk}}, k = 1, ..., K$$
$$\hat{\sigma}_k^2 = \frac{\sum_k \hat{\gamma}_{jk} (D_j^{src}-\mu_k)^2}{\sum_k \hat{\gamma}_{jk}}, k = 1, ..., K$$
$$\hat{\alpha}_k = \frac{\sum_k \hat{\gamma}_{jk}}{N}, k = 1, ..., K$$

where $\hat{\mu}_k$ is the estimated mean of the k-th component, $\hat{\sigma}_k^2$ is the estimated variance of the k-th component, and $\hat{\alpha}_k$ is the weight coefficient of the k-th model. The E-step and M-step are iterated until convergence, at which point the GMM parameters $\{\hat{\mu}_k, \hat{\sigma}_k^2, \hat{\alpha}_k\}$ are obtained. The GMM characterized by these parameters represents the maximum likelihood probability model for all known source domain samples. The maximum likelihood mean is the weighted sum average of the component means of the GMM, and the maximum likelihood variance is the weighted sum average of the component variances of the GMM. The equations for the maximum likelihood mean and variance are as follows:

$$\hat{\mu} = \frac{1}{K}\sum_k \hat{\alpha}_k \hat{\mu}_k, \ \hat{\Sigma} = \frac{1}{K}\sum_k \hat{\alpha}_k \hat{\sigma}_k^2$$

In accordance with Bayesian theory, the posterior probability distribution of the local model at timestep t is obtained by multiplying the prior with the likelihood distribution. Given the local model is a multivariate normal distribution with an unknown mean and covariance matrix, a normal-inverse-Wishart distribution, defined by parameters $\hat{\Sigma}$, $\hat{\mu}$, m and n_0, serves as the prior. Here, m and n_0 represent the number of samples.

Utilizing maximum likelihood estimation theory, the maximum likelihood estimation mean of the sample at timestep t is $\mu_{ll,t} = \frac{1}{N}\sum_n y_t^n$, and the maximum likelihood estimation variance is $\Sigma_{ll,t} = \frac{1}{N}\sum_n (y_t^n - \mu_{ll,t})(y_t^n - \mu_{ll,t})^T$. Under this prior and likelihood distribution, the maximum a posteriori estimation mean μ_t and covariance Σ_t of the local model are provided by the following equation:

$$\mu_t = \frac{m\mu_0 + n_0\mu_{ll,t}}{m + n_0} \tag{3}$$

$$\Sigma_t = \frac{\hat{\Sigma} + N\Sigma_{ll,t} + \frac{Nm}{N+m}(\mu_{ll,t} - \hat{\mu})(\mu_{ll,t} - \hat{\mu})^T}{N + n_0} \tag{4}$$

where N is the number of samples points. To calculate the parameters of the dynamic, we decompose the mean and covariance of the multivariate normal distribution as follows:

$$\mu_t = \begin{bmatrix} \mu_{xu} \\ \mu_{x'} \end{bmatrix}, \Sigma_t = \begin{bmatrix} \Sigma_{xu,xu} & \Sigma_{xu,x'} \\ \Sigma_{x',xu} & \Sigma_{x',x'} \end{bmatrix}$$

where μ_{xu} denotes the vector composed of the current state and action at time t, while $\mu_{x'}$ represents the state reached by the system at time $t+1$. The covariance matrix is expressed as a block matrix, maintaining the same structure as previously mentioned. Given that the local data are $[x_t^{src}, u_t^{src}, x_{t+1}^{src}]^T$, the conditional distribution of the dynamics is $\boldsymbol{x}_{t+1}^{src} = N(f_{\boldsymbol{xt}}\boldsymbol{x}_t^{src} + f_{\boldsymbol{ut}}\boldsymbol{u}_t^{src} + f_{\boldsymbol{ct}}, \boldsymbol{F}_t)$. We let $\boldsymbol{F}_{mt} = [f_{\boldsymbol{xt}}, f_{\boldsymbol{ut}}]$, $\mu_{xu} = [\boldsymbol{x}_t^{src}, \boldsymbol{u}_t^{src}]^T$ and $\mu_{x'} = [\boldsymbol{x}_{t+1}^{src}]$.

Consequently, we obtain $\mu_{x'} = N(\boldsymbol{F}_{mt}\mu_{xu} + f_{\boldsymbol{ct}}, \boldsymbol{F}_t)$. $\boldsymbol{F}_{mt}$ represents the noise-free transition matrix from μ_{xu} to $\mu_{x'}$. Therefore, we assume that the noise-free transition of the covariance also applies to the matrix $\boldsymbol{F}_{mt}$, resulting in $\Sigma_{xu,xu}\boldsymbol{F}_{mt} = \Sigma_{xu,x'}$. According to the theory of conditional probability, the parameters of the local dynamic model of the system are as follows:

$$\boldsymbol{F}_{mt} = \Sigma_{xu,xu}^{-1}\Sigma_{xu,x'} \tag{5}$$

$$f_{\boldsymbol{ct}} = \mu_{x'} - \boldsymbol{F}_{mt}\mu_{xu} \tag{6}$$

$$\boldsymbol{F}_t = \Sigma_{x',x'} - \boldsymbol{F}_{mt}\boldsymbol{\Sigma}_{xu,xu}\boldsymbol{F}_{mt}^T \tag{7}$$

C. Learning Approximate Optimal Source Domain Policies

Upon determining the system dynamic model parameters, the LQR algorithm effectively directs the policy search in the source domain toward regions with higher rewards. The optimization objective for approximating the optimal source domain policy parameters is defined as follows:

$$min_{p,\boldsymbol{\theta}_{src}} E_p[\Sigma_t l(\boldsymbol{x}_t, \boldsymbol{u}_t)], s.t. \pi_{\boldsymbol{\theta}_{src}}(\boldsymbol{u}_t|\boldsymbol{x}_t) = p(\boldsymbol{u}_t|\boldsymbol{x}_t) \tag{8}$$

The algorithm seeks to identify the linear Gaussian controller parameters and the neural network parameters $\boldsymbol{\theta}_{src}$ that minimize the expected loss. The equality constraint is imposed to ensure that the distribution of the controller guaranteed by the source domain policy parameters is close to that of the linear Gaussian controller. For optimization problems with equality constraints such as this one, it is common to formulate a Lagrange equation from the optimization problem and constraints and subsequently solve it using the dual gradient descent method. The alternating update formulas for the linear Gaussian controller parameters p, neural network parameters $\boldsymbol{\theta}_{src}$, and Lagrange multipliers $\lambda_{\mu t}$ are provided below. A detailed derivation of these formulas can be found in [26].

$$\begin{aligned} \theta_{src} \leftarrow \arg\min_{\theta_{src}} \sum\nolimits_t E_{p(\boldsymbol{x}_t)\pi_{\theta_{src}}(\boldsymbol{u}_t|\boldsymbol{x}_t)}[\boldsymbol{u}_t^T\lambda_{\mu t}] + \\ v_t E_{p(\boldsymbol{x}_t)}[D_{KL}(\pi_{\theta_{src}}(\boldsymbol{u}_t \mid \boldsymbol{x}_t) \parallel p(\boldsymbol{u}_t \mid \boldsymbol{x}_t))]. \end{aligned} \tag{9}$$

$$\begin{aligned} p \leftarrow \arg\min_{p} \sum\nolimits_t E_{p(\boldsymbol{x}_t,\boldsymbol{u}_t)}[l(\boldsymbol{x}_t, \boldsymbol{u}_t) - \boldsymbol{u}_t^T\lambda_{\mu t}] + \\ v_t E_{p(\boldsymbol{x}_t)}[D_{KL}(p(\boldsymbol{u}_t \mid \boldsymbol{x}_t) \parallel \pi_{\theta_{src}}(\boldsymbol{u}_t \mid \boldsymbol{x}_t))]. \end{aligned} \tag{10}$$

$$\lambda_{\mu t} \leftarrow \lambda_{\mu t} + \alpha v_t (E_{\pi_{\theta_{src}}(\boldsymbol{u}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t)}[\boldsymbol{u}_t] - E_{p(\boldsymbol{u}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t)}[\boldsymbol{u}_t]). \tag{11}$$

Upon convergence of the algorithm, we acquire the approximate optimal source domain policy parameters tailored for the current task.

D. Learning Target Domain Policy

When the target domain policy can be deployed in the target domain, the generated trajectory from the target domain should exhibit minimal loss values. Therefore, we define the objective of determining the source domain policy adjustment parameter as follows:

$$\boldsymbol{\kappa}^* = argmin_{\boldsymbol{\kappa}} E_{\boldsymbol{\theta}_{tgt}}[l(\boldsymbol{\tau}_{tgt})] \tag{12}$$

where $l(\boldsymbol{\tau}_{tgt})$ characterizes the proximity of the trajectory in the target domain to the task objectives and serves as a

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

performance metric for the task. We define the robot state information obtained from sensors as the target domain observation state, denoted as z_t^{tgt}. The target domain state is x_t^{tgt}. Our objective is to estimate the true value of x_t^{tgt} using z_t^{tgt} and the source domain state x_t^{src}.

We partition the robot's trajectory while it performs a task in the target domain uniformly into T segments. When T is sufficiently large, adjacent target domain states can be approximated as linearly related. Thus, the target domain state information to be estimated adheres to the mathematical form of variable values processed by traditional Kalman filter methods:$\boldsymbol{x}_{t+1}^{tgt} = \boldsymbol{\Phi}\boldsymbol{x}_t^{tgt} + \boldsymbol{w}_t$. where $\boldsymbol{\Phi}$ represents the mapping matrix from $\boldsymbol{x}_t^{tgt}$ to $\boldsymbol{x}_{t+1}^{tgt}$, typically assumed to be time-invariant; $\boldsymbol{w}_t$ is Gaussian noise with known variance.

The target domain observation state can be modeled as: $\boldsymbol{z}_t^{tgt} = \boldsymbol{H}\boldsymbol{x}_t^{tgt} + \boldsymbol{v}_t$, where $\boldsymbol{z}_t^{tgt}$ is the target domain observation state at time t. $\boldsymbol{H}$ is a time-invariant, noise-free transfer matrix between the target domain state and the target domain observation state; $\boldsymbol{v}_t$ is the corresponding observation error, and its cross-correlation with the state noise $\boldsymbol{w}_t$ is 0. Assuming the covariance of the two noise models remains stable over time, the covariance matrices of the noise can be defined as $\boldsymbol{Q} = E[\boldsymbol{w}_k\boldsymbol{w}_k^T]$, $\boldsymbol{R} = E[\boldsymbol{v}_k\boldsymbol{v}_k^T]$. To ensure the accuracy of target domain state estimation, we need to describe the deviation between the target domain state estimate and the target domain state. Therefore, we use $\boldsymbol{P}_t^{tgt}$ to describe the mean square error between the target domain state $\boldsymbol{x}_t^{tgt}$ and the target domain state estimate $\hat{\boldsymbol{x}}_t^{tgt}$ at time step t. $\boldsymbol{P}_t^{tgt}$ has the following quadratic form:

$$\boldsymbol{P}_t^{tgt} = E[(\boldsymbol{x}_t^{tgt} - \hat{\boldsymbol{x}}_t^{tgt})(\boldsymbol{x}_t^{tgt} - \hat{\boldsymbol{x}}_t^{tgt})^T]$$

We consider the target domain state estimate to be most accurate when the extremum of $\boldsymbol{P}_t^{tgt}$ is reached. Therefore, we identify the extremum of $\boldsymbol{P}_t^{tgt}$. The prior estimation of the target domain state $\hat{\boldsymbol{x}}_t^{tgt}$ is denoted as $(\hat{\boldsymbol{x}}_t^{tgt})'$, which is obtained from the prior knowledge of the source domain system model. To make the target domain state estimation more reliable, we incorporate the target domain state prior estimate $(\hat{\boldsymbol{x}}_t^{tgt})'$ and the target domain observed state $\boldsymbol{z}_t^{tgt}$, as shown in the equation:

$$\hat{\boldsymbol{x}}_t^{tgt} = (\hat{\boldsymbol{x}}_t^{tgt})' + \boldsymbol{K}_t(\boldsymbol{z}_t^{tgt} - \boldsymbol{H}(\hat{\boldsymbol{x}}_t^{tgt})')$$

here, $\boldsymbol{z}_t^{tgt} - \boldsymbol{H}(\hat{\boldsymbol{x}}_t^{tgt})'$ is the observation residual. It represents the difference between the actual observation and the expected observation based on the prior estimated state $(\hat{\boldsymbol{x}}_t^{tgt})'$. If this value is large, it suggests that our prediction might be inaccurate. $\boldsymbol{K}_t$ is the Kalman gain coefficient, which determines how much "weight" we should give to the observation residual. If the Kalman gain is large, it means we trust our observation more; if it's small, it means we trust our prediction more. The purpose of the entire state update equation is to combine the predicted state with the weighted value of the observation residual to obtain the best estimate of the current state. This optimal estimate takes into account both the model's prediction and the actual observation data. The value of z_t^{tgt} has already been given in the previous paragraph. Now, by substituting the value of z_t^{tgt} into the equation, we get:

$$\hat{\boldsymbol{x}}_t^{tgt} = (\hat{\boldsymbol{x}}_t^{tgt})' + \boldsymbol{K}_t(\boldsymbol{H}\boldsymbol{x}_t^{tgt} + \boldsymbol{v}_t - \boldsymbol{H}(\hat{\boldsymbol{x}}_t^{tgt})') \tag{13}$$

Thus, we can simplify the expression of P_t^{tgt} to:

$$\boldsymbol{P}_t^{tgt} = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})(\boldsymbol{P}_t^{tgt})'(\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H}) + \boldsymbol{K}_t\boldsymbol{R}(\boldsymbol{K}_t)^T$$

where $(\boldsymbol{P}_t^{tgt})'$ represents the prior estimate of $\boldsymbol{P}_t^{tgt}$. The trace, i.e., the sum of the diagonal elements, of the covariance matrix is the total variance of the data. To determine the extremum of the covariance matrix $\boldsymbol{P}_t^{tgt}$ with respect to the Kalman gain $\boldsymbol{K}_t$, we compute the derivative of the trace of $\boldsymbol{P}_t^{tgt}$ with respect to $\boldsymbol{K}_t$. When the derivative is zero, we obtain:

$$\boldsymbol{K}_t = (\boldsymbol{P}_t^{tgt})'\boldsymbol{H}^T(\boldsymbol{H}(\boldsymbol{P}_t^{tgt})'\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{14}$$

Then, we can obtain the target domain state estimation error covariance matrix under the optimal Kalman gain:

$$\boldsymbol{P}_t^{tgt} = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})(\boldsymbol{P}_t^{tgt})' \tag{15}$$

At each time step t, the target domain policy's correction parameter is $\boldsymbol{\kappa} = \{\boldsymbol{H}, \boldsymbol{\Phi}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{K}_t\}$. After correcting the observed state in the target domain, the approximate action output in the target domain is given by:

$$\hat{\boldsymbol{u}}_t^{tgt} = \pi(\boldsymbol{u}_t|\boldsymbol{z}_t^{tgt}; \boldsymbol{\theta}_{src}^*, \boldsymbol{\kappa}) \tag{16}$$

In the target domain, the robot's prior estimation at time $t+1$ in the target domain is given by the source domain system:

$$(\hat{\boldsymbol{x}}_t^{tgt})' = f_{\boldsymbol{xt}}\hat{\boldsymbol{x}}_t^{tgt} + f_{\boldsymbol{ut}}\hat{\boldsymbol{u}}_t^{tgt} \tag{17}$$

Upon determining the error covariance matrix for the next instant, a recursive form is established. At time $t+1$, the target domain state prior estimation error takes the following form:

$$\boldsymbol{e}_{t+1}^{'} = \boldsymbol{\Phi}\boldsymbol{x}_t^{tgt} + \boldsymbol{w}_t - (f_{\boldsymbol{xt}}\hat{\boldsymbol{x}}_t^{tgt} + f_{\boldsymbol{ut}}\hat{\boldsymbol{u}}_t^{tgt})$$

Finally, the error covariance of the target domain state at the next moment can be obtained.

$$(\boldsymbol{P}_{t+1}^{tgt})' = E[\boldsymbol{e}_{t+1}^{'}(\boldsymbol{e}_{t+1}^{'})^T] \tag{18}$$

E. Pseudocode

Algorithm 1 outlines the procedure for the KFOTL method, which comprises two main stages. First, the source domain policy is trained within the source domain utilizing a model-based approach. Subsequently, the discrepancy between the source and target domain states is treated as Gaussian noise, with the Kalman filtering technique employed to minimize the distance between them. This process ultimately results in the acquisition of the target domain controller.

V. EXPERIMENTS

A. Experimental Setup

We designed four experimental scenarios in both simulated environments and real-world settings. The first two scenarios are purely simulation experiments, with environments based on Gazebo + ROS Kinetic. According to the widely referenced paper [18], factors significantly impacting transfer performance and related to the robot's intrinsic properties include the robot's link mass and joint damping. Consequently, we created sim-PR2 and real-PR2 robots within the Gazebo environment, with the primary distinctions being the joint damping and link mass parameters. The real-PR2 robot employs the default URDF parameters, whereas the sim-PR2 robot utilizes parameter ranges of $[0.1, 4.0] * \boldsymbol{DefaultDamping}$, $[0.1, 3.0] * \boldsymbol{DefaultMass}$. Scenario 3 and 4 involves an experiment with UR5 in a real-world setting.

The state variables in the experiments are $[\boldsymbol{x}_{ja}, \boldsymbol{x}_{jv}, \boldsymbol{x}_p, \boldsymbol{x}_v]$, representing the robot's joint angles and joint velocities and the end effector's position and velocities, respectively. In **Scenario 1** and **Scenario 2**, we employed torque control, with the actuation based on the torques of its seven joints. The robot

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

Algorithm 1: KFOTL method

Input: initialization parameters
Output: $\pi(\boldsymbol{\theta}_{tgt})$

for *iteration = 1 to Num* **do**
  Collect $\boldsymbol{D}_{src}$;
  Update $f_{\boldsymbol{x}t}$, $f_{\boldsymbol{u}t}$, f_{ct} and $\boldsymbol{F}_t$;
  Update linear-Gaussian controllers p;
  Update source domain policy $\boldsymbol{\theta}_{src}$;
  Update $\lambda_{\mu t}$;
end
Collect $\boldsymbol{D}_{tgt}$ using $\pi(\boldsymbol{\theta}^*_{src})$;
Calculate $\boldsymbol{H}$ and $\boldsymbol{v}_t$;
for t *= 0 to* T **do**
  $(\hat{\boldsymbol{x}}_t^{tgt})' \leftarrow f_{\boldsymbol{x}t}\hat{\boldsymbol{x}}_t^{tgt} + f_{\boldsymbol{u}t}\hat{\boldsymbol{u}}_t^{tgt}$;
  $\boldsymbol{P}_t^{tgt} \leftarrow (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})(\boldsymbol{P}_t^{tgt})'$;
  $\boldsymbol{K}_t \leftarrow (\boldsymbol{P}_t^{tgt})'\boldsymbol{H}_T(\boldsymbol{H}(\boldsymbol{P}_t^{tgt})'\boldsymbol{H}_T + \boldsymbol{R})^{-1}$;
  $\hat{\boldsymbol{x}}_t^{tgt} \leftarrow (\hat{\boldsymbol{x}}_t^{tgt})' + \boldsymbol{K}_t(\boldsymbol{H}\boldsymbol{x}_t^{tgt} + \boldsymbol{v}_t - \boldsymbol{H}(\hat{\boldsymbol{x}}_t^{tgt})')$;
  Update the error covariance $(\boldsymbol{P}_{t+1}^{tgt})'$;
  $\boldsymbol{u}_{tgt} \leftarrow \pi_{tgt}(\boldsymbol{u}_t|\boldsymbol{z}_t^{tgt}; \boldsymbol{\theta}^*_{src}, \boldsymbol{\kappa})$
end
Return optimized policy parameters $\boldsymbol{\theta}^*_{tgt} = \{\boldsymbol{\theta}^*_{src}, \boldsymbol{\kappa}\}$;

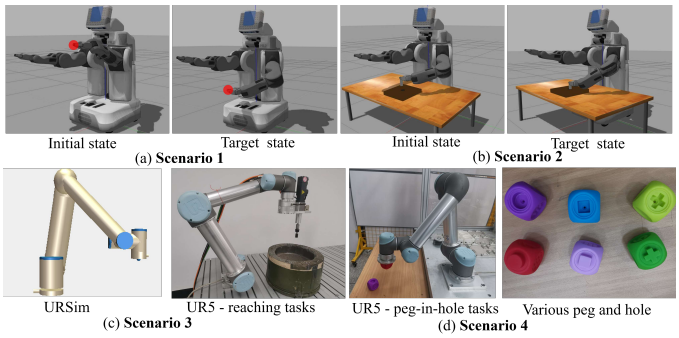

Fig. 2. Three experimental evaluation scenarios are designed for the algorithm. (a) **Scenario 1** involves controlling the PR2 robot in Gazebo to complete reaching tasks; (b) **Scenario 2** consists of controlling the PR2 robot in Gazebo to complete peg-in-hole tasks; (c) **Scenario 3** involves a UR5 robot cutting plastic materials, in which we control the real-UR5 to complete reaching tasks prior to the cutting task. (d) **Scenario 4** involves a peg-in-hole experiment with the UR5 robot, featuring various peg and hole components. In **Scenario 1** and **Scenario 2**, the differences between the source and target domains are due to the robot's varying joint damping and link mass, which are artificially set. In **Scenario 3** and **Scenario 4**, the distinction between the source and target domains lies in the dynamic differences between the robot simulator and the real robot. https://youtu.be/C_dFFmXZ5HU

interacts with its environment through the $< rostopic >$ and $< rosservice >$ mechanisms. In **Scenario 3** and **Scenario 4**, we utilized position control mode, directing the motion based on the angular positions of its six joints. The connection between the supervisory computer and the actual UR5 robot machine was achieved using socket communication protocols and the URScript Programming Language. In all scenarios, the sampling frequency is set at 20 Hz, with the sampling trajectory consisting of a fixed 100 time steps. For both reaching tasks and peg-in-hole tasks, we employ the same objective function:

$$l(\boldsymbol{\tau}) = \boldsymbol{w}_u \sum_t l(\boldsymbol{u}_t) + \boldsymbol{w}_x \sum_t l(\boldsymbol{x}_t) + \boldsymbol{w}_{xf} l_f(\boldsymbol{x}_N) \quad (19)$$

where $\boldsymbol{w}_u$, $\boldsymbol{w}_x$ and $\boldsymbol{w}_{xf}$ represent the torque cost coefficient, state cost coefficient, and final time robot state cost coefficient, respectively. In **Scenario 1** and **Scenario 2**, the parameters are set as $w_u = 1.0$,$w_x = 1.0$, and $w_{xf} = 1.0$. In **Scenario 3** and **Scenario 4**, the parameters are $w_u = 0.0$, $w_x = 1.0$, and $w_{xf} = 1.0$. Furthermore, global policy is composed of a 32-40-40-7 fully connected neural network. We utilize the Adam solver with a learning rate of 0.001 and a batch size of 25.

B. Evaluation in the Target Domain

To demonstrate the effectiveness of the proposed KFOTL method, we carried out two sets of experiments in both **Scenario 1** and **Scenario 2**, with each set comprising 10 trials. The controller's performance was evaluated by determining the L1 norm of the robot's Tool Center Point (TCP) at the final time step relative to the target position. We use box plots to illustrate the discrete distribution of the robot's TCP in relation to the target position. As depicted in Figure 3, the red box plots represent the source domain policy's performance in the source domain, the blue box plots indicate the performance of the source domain policy when directly applied to the target domain, and the purple box plots illustrate the performance of the target domain policy achieved through the KFOTL method in the target domain.

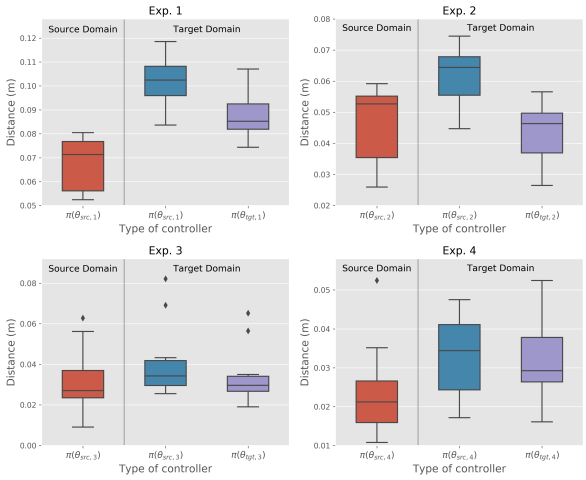

Fig. 3. Evaluate the performance of the target domain policy modified by the KFOTL method in the target domain. Exp.1 and Exp.2 represent the source domain policy obtained under different random seeds in **Scenario 1**. Exp.3 and Exp.4 represent the source domain policy obtained under different random seeds in **Scenario 2**. In the same scenario, the only difference between the two sets of experiments is the random seed.

We utilized the performance of the source domain policy in the source domain for each experiment as the evaluation benchmark. The upper edge, lower edge, upper quartile, and lower quartile of the blue box are all greater than the values of the red box. This implies that when the source domain policy is directly implemented in the target domain without modification, the robot's TCP deviation from the target position increases, and the sample points become more dispersed. In contrast, it is evident that the values of the four indicators of the purple box are all lower than those of the blue box. This signifies that the target domain policy, modified by the KFOTL method, improves the ability to transfer to the target domain.

C. Success Rate in Manipulation Tasks

To assess the efficacy of the KFOTL approach in specific manipulation tasks, we examined the algorithm's success rate

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

TABLE I
SUCCESS RATES OF KFOTL METHOD ON EACH SCENARIO. SUCCESS IS DEFINED AS INSERTING THE PEG INTO THE HOLE WITH A FINAL DISTANCE OF LESS THAN 0.03 AND REACHING THE GOAL WITH A FINAL DISTANCE OF LESS THAN 0.06. S. D. DENOTES THE SOURCE DOMAIN AND T. D. DENOTES THE TARGET DOMAIN

Iter.	Scenario 1			Scenario 2			Scenario 3		
	$\boldsymbol{\theta}_{src}$ in S.D.	$\boldsymbol{\theta}_{src}$ in T.D.	KFOTL in T.D.	$\boldsymbol{\theta}_{src}$ in S.D.	$\boldsymbol{\theta}_{src}$ in T.D.	KFOTL in T.D.	$\boldsymbol{\theta}_{src}$ in S.D.	$\boldsymbol{\theta}_{src}$ in T.D.	KFOTL in T.D.
8	0/10	0/10	0/10	6/10	3/10	5/10	1/10	0/10	0/10
9	3/10	0/10	0/10	3/10	1/10	2/10	2/10	0/10	2/10
10	10/10	6/10	10/10	8/10	3/10	6/10	10/10	4/10	7/10
11	10/10	5/10	9/10	9/10	2/10	6/10	9/10	7/10	10/10

in all three scenarios. In **Scenario 1**, the source domain policy achieved a success rate of 3/10 in the source domain by the 9th iteration. Following the 10th iteration, the task consistently maintained a high success rate of 10/10. Nevertheless, when implementing the unaltered source domain policy in the target domain environment, performance deterioration transpired in all three scenarios. In **Scenario 2**, the source domain policy at the 11th iteration achieved a success rate of $2/10$ in the target domain, which is a decrease of $7/10$ compared to its success rate in the source domain. In contrast, the target domain policy derived using the KFOTL method attained a success rate of $6/10$ in the target domain, only a decline of $3/10$ from the source domain policy's performance in its native domain. This is significantly superior to the performance of the unmodified source domain policy in the target domain. Furthermore, across all three scenarios, the policies refined using the KFOTL method demonstrated consistent performance improvements.The experimental findings corroborate the fact that our method accounts for disparities between the source and target domains, thereby mitigating the reality gap.

D. Comparison with Other Methods

We carried out experiments comparing the recently proposed MCF and DR method. These experiments were conducted under Scenario 1. When deploying the MCF method, we utilized the Linear Gaussian Controller trained in the source domain as a prior policy, and on the basis of this policy, executed the MCF Deployment operation, resulting in the MCF policy. Additionally, when replicating the DR method, the joint damping and link mass of the sim-PR2 robot were discretized within the ranges of $[0.1, 4.0] * DefaultDamping$ and $[0.1, 3.0] * DefaultMass$, respectively. This is due to the fact that entirely random damping and mass parameters pose significant challenges for algorithm convergence, substantially increasing the training time for the controller. Testing was conducted on a real-PR2 robot with default parameters. The learning objective for all three methods was to minimize the cost of the sampled trajectory. Each assessment was executed 10 times. Figure 4 illustrates the L1 norm of the robot's TCP relative to the target position, indicating the proximity of the robot's TCP to the target. To quantify each algorithm's performance, we further established the trajectory quality score (TQS) for the entire sampled trajectory, which is defined as follows:

$$TQS = -\sum_{0}^{T} \|\boldsymbol{x}_{TCP} - \boldsymbol{x}_p\|_2 + log(\|\boldsymbol{x}_{TCP} - \boldsymbol{x}_p\|_2 + \epsilon) \quad (20)$$

where $\|\boldsymbol{x}_{TCP} - \boldsymbol{x}_p\|_2$ is the distance of the robot's TCP from the target position, and $\epsilon = 0.0001$. A high trajectory quality score is assigned when the TCP is near the target position.

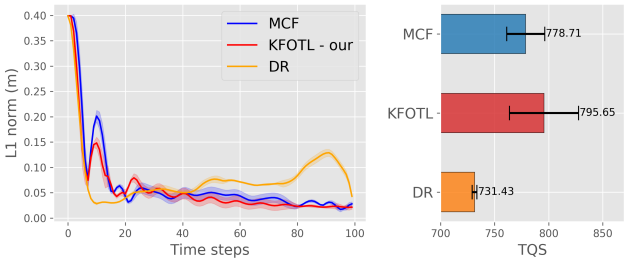

Fig. 4. Comparison of our policy with the MCF policy and the DR policy. The color of the area is the confidence interval for the performance of the policy. These experiments are in the **Scenario 1**.

As depicted in Figure 4, for the same number of training iterations in the source domain policy, the target domain trajectories produced by the KFOTL method are superior to those generated by the DR and MCF approaches. For instance, the TQS score of KFOTL is 795.65, which is 2.18% greater than that of the MCF method and 8.78% higher than that of the DR method. These findings demonstrate that, with equal computational power, the KFOTL method yields superior results in the target domain.

E. Evaluation in the Real World

Next, we assessed the performance of the KFOTL approach on an actual UR5 robotic arm by conducting four distinct reaching tasks in Scenario 3. Figure 5 illustrates the results, where the vertical axis denotes the L1 distance between the UR5's TCP and the target position, while the horizontal axis denotes the control timestep. In this experiment, the term 'Baseline' refers to the performance of the unmodified source domain policy in the target domain. The configuration for the MCF method remains consistent with the prior section. Within the DR method, dynamic adjustment is exclusively applied to the timestep parameter to address dynamic delays [20]. During policy testing, the protocol was set to cease sending control actions to the robotic arm once the distance between the robot's TCP and the target position did not decrease for two consecutive timesteps. A trial was terminated when the distance ceased to decrease. Each task was subjected to ten repetitions, with each curve in the figure representing the mean and variance of the ten test samples.

The experimental outcomes are depicted in Figure 5. Observing the sample distance curves, within the same timeframe, the KFOTL method exhibits a faster approach speed towards the target and superior TQS values. When calculating the TQS values for both methods based on the stopping time of the KFOTL approach, the KFOTL approach exhibits improve-

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

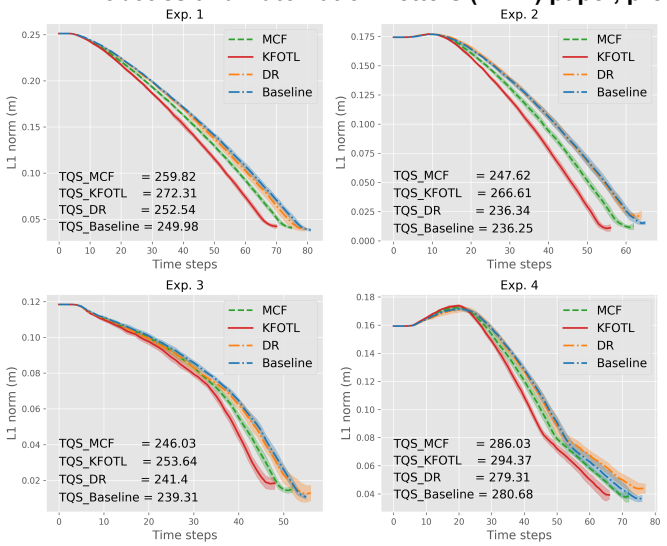

Fig. 5. Evaluation of the KFOTL method on the real UR5 robot. The baseline refers to the performance of deploying the source domain policy trained through URSim directly onto the real UR5. Exp.1, Exp.2, Exp.3, and Exp.4 are the experiments in **Scenario 3**, with four different target positions.

ments of 8.20%, 11.39%, 5.65%, and 4.65% over the baseline method for the four experiments. The trajectory improvement is quantified by $(TQS_{KFOTL} - TQS_{Baseline})/TQS_{Baseline}$. These results suggest that the KFOTL outperforms the baseline, MCF and DR methods when implemented on the actual UR5 robotic arm.

In Exp.3 and Exp.4, we observed a higher final error with the KFOTL method. Far from the target position, sample points are relatively sparse. The KFOTL method can adeptly leverage target domain data to amend the source domain policy, leading to quicker proximity to the objective. However, near the target with dense samples, it risks overfitting.

TABLE II
PEG-IN-HOLE TASKS USING UR5

-	DR	MCF	KFOTL (our)
Circular hole	4/10	3/10	6/10
Square hole	2/10	4/10	5/10
Cross-shaped hole	0/10	1/10	3/10

Finally, we conducted tests in **Scenario 4**. The engagement with the cross-shaped hole presented a more complex contact scenario, resulting in a comparatively lower success rate for this task. Across all three experimental conditions, the KFOTL method demonstrated superiority over two foundational methodologies. This suggests that enhancing the adaptability of source domain strategies to target domains via the KFOTL method is feasible and aligns with our initial design expectations.

VI. CONCLUSIONS

In this paper, we treat the reality gap as a disturbance and propose a solution to reduce the mismatch between source and target domain data using Kalman filtering. First, we model the difference between the physical and simulated environments as a filtering problem with disturbance; second, we utilize target domain data to refine the source domain policy, minimizing the reality gap and improving the controller's adaptability to the target domain. Our method is evaluated in three scenarios. In future work, we will further extend this research to a broader range of industrial robotic tasks, aiming to address unstructured, unmodelable complex scenarios in industrial production applications.

REFERENCES

[1] Z. X. Pan, J. Polden, N. Larkin *et al.*, "Recent progress on programming methods for industrial robots," *Rob. Comput.-Integr. Manuf.*, vol. 28, no. 2, pp. 87–94, 2012.
[2] D. Hoeller, L. Wellhausen *et al.*, "Learning a state representation and navigation in cluttered and dynamic environments," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5081–5088, 2021.
[3] D. Silver, J. Schrittwieser *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[4] V. Mnih, A. P. Badia *et al.*, "Asynchronous methods for deep reinforcement learning," in *33rd Int. Conf. Mach. Learn.* PMLR, 2016, pp. 1928–1937.
[5] O. Vinyals, T. Ewalds, S. Bartunov *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.
[6] N. Heess, D. Tb, S. Sriram *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.
[7] F. Golemo, A. A. Taiga, A. Courville *et al.*, "Sim-to-real transfer with neural-augmented robot simulation," in *Proc. 2nd Conf. Robot Learn.* PMLR, 2018, pp. 817–828.
[8] T. Yoneda, G. Yang, M. R. Walter *et al.*, "Invariance through latent alignment," *arXiv preprint arXiv:2112.08526*, 2021.
[9] S. Levine, P. Pastor *et al.*, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *Int. J. Robot. Res.*, vol. 37, no. 4-5, pp. 421–436, 2018.
[10] W. Zhao, J. P. Queralta *et al.*, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *IEEE Symp. Ser. Comput. Intell.* IEEE, 2020, pp. 737–744.
[11] M. Breyer, F. Furrer, T. Novkovic *et al.*, "Flexible robotic grasping with sim-to-real transfer based reinforcement learning," *arXiv preprint arXiv:1803.04996*, 2018.
[12] M. Andrychowicz, B. Baker, M. Chociej *et al.*, "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.
[13] K. Bousmalis, A. Irpan, P. Wohlhart *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *IEEE Int. Conf. Robot. Autom.* IEEE, 2018, pp. 4243–4250.
[14] J. Wang, Y. Liu *et al.*, "Reinforcement learning with perturbed rewards," in *34th AAAI Conf. Artif. Intell.*, vol. 34, no. 04, 2020, pp. 6202–6209.
[15] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *J. Stat. Plan. Inference*, vol. 90, no. 2, pp. 227–244, 2000.
[16] Y. Pan, C.-A. Cheng *et al.*, "Agile autonomous driving using end-to-end deep imitation learning," *arXiv preprint arXiv:1709.07174*, 2017.
[17] Z. He, R. Julian, E. Heiden *et al.*, "Zero-shot skill composition and simulation-to-real transfer by learning task representations," *arXiv preprint arXiv:1810.02422*, 2018.
[18] J. Hanna *et al.*, "Grounded action transformation for robot learning in simulation," in *31th AAAI Conf. Artif. Intell.*, vol. 31, no. 1, 2017.
[19] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* IEEE, 2016, pp. 4019–4026.
[20] X. B. Peng, M. Andrychowicz *et al.*, "Sim-to-real transfer of robotic control with dynamics randomization." IEEE, 2018, pp. 3803–3810.
[21] K. Rana, V. Dasagi, B. Talbot *et al.*, "Multiplicative controller fusion: Leveraging algorithmic priors for sample-efficient reinforcement learning and safe sim-to-real transfer," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* IEEE, 2020, pp. 6069–6076.
[22] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," Seattle, WA, 2015, pp. 156–163.
[23] W. Montgomery and S. Levine, "Guided policy search via approximate mirror descent," vol. 29, Barcelona, Spain, 2016.
[24] I. Mordatch *et al.*, "Combining the benefits of function approximation and trajectory optimization." in *Robot. Sci. Syst.*, vol. 4, 2014, p. 23.
[25] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.
[26] S. Levine, C. Finn, T. Darrell *et al.*, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 39, 2016.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.