2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

# Adaptive Trajectory Database Learning for Nonlinear Control with Hybrid Gradient Optimization

Kuan-Yu Tseng[1], Mengchao Zhang[1], Kris Hauser[2], and Geir E. Dullerud[1]

***Abstract*— This paper presents a novel experience-based technique, called *EHGO*, for sample-efficient adaptive control of nonlinear systems in the presence of dynamical modeling errors. The starting point for EHGO is a database seeded with many trajectories optimized under a reference estimate of real system dynamics. When executed on the real system, these trajectories will be suboptimal due to errors in the reference dynamics. The approach then leverages a hybrid gradient optimization technique, GRILC, which observes executed trajectories and computes gradients from the reference model to refine the control policy without requiring an explicit model of the real system. In past work, GRILC was applied in a restrictive setting in which a robot executes multiple rollouts from identical start states. In this paper, we show how to leverage a database to enable GRILC to operate across a wide envelope of possible start states in different iterations. The database is used to balance between start state proximity and recentness-of-experience via a learned distance metric to generate good initial guesses. Experiments on three dynamical systems (pendulum, car, drone) show that the proposed approach adapts quickly to online experience even when the reference model has significant errors. In these examples EHGO generates near-optimal solutions within hundreds of epochs of real execution, which can be orders of magnitude more sample efficient than reinforcement learning techniques.**

## I. Introduction

Although robot control algorithms are initially developed in ideal settings with laboratory data, robots should adapt their behavior as they receive real-world experience to address the dynamical gap between the laboratory and the deployment environments. In particular, this paper considers the setting of dynamical mismatch, which could come from calibration errors of system dynamics parameters or non-parametric variations in environmental conditions. Model-free, model-based, and hybrid methods have been proposed to address such settings. Model-free approaches, like Deep Reinforcement Learning (DRL) [1], [2], [3], enable robots to learn policies by directly interacting with a simulated environment, and researchers have developed different techniques to mitigate performance degradation when transferring a learned policy to the real system. However, these algorithms are typically data inefficient and require extensive hyperparameter tuning on a case-by-case basis. Model-based approaches, such as adaptive Model Predictive Control (MPC) [4], [5], learn the dynamics of the real world from experience for use in optimization. The disadvantage of such techniques is that they require solving a difficult system identification problem, and flexible models like neural networks can be challenging for trajectory optimizers to solve in real time. Hybrid approaches, such as Iterative Learning Control (ILC) [6], utilize a model of the simulated system to compute optimal control policies, which are then applied to the real system and refined based on execution feedback. ILC does not require an accurate model of the true system, but it does require repeated restarts from the same initial state, and hence it can only be applied to episodic problems.

This paper presents Experience-Based Hybrid Gradient Optimization (EHGO), a technique that generalizes ILC to remove the constraint that the system needs to be restarted from the same state across iterations. Our approach stores an experience set of trajectories, first generated during simulation, and uses the experience to seed an ILC optimizer. As new experience is gathered from the real world, the newly optimized trajectories are added to the database. We use a hybrid gradient optimizer, Gradient-based Iterative Learning Control (GRILC) [7], to perform the ILC step. Notably, GRILC combines gradients from the *simulated dynamics* and states and costs from the *real system* to optimize trajectories for the real dynamics. In this fashion, our method can obtain numerical stability and fast convergence.

A key question is how to combine experience on the simulated system with real experience. Reminiscent of Nearest Neighbors Optimal Control [8] our approach selects nominal trajectories from the experience database that are "relevant" to the query initial state. Relevance is based on a distance metric that captures both the deviation of the initial states between the query and the neighbor, as well as the number of iterations which the neighbor has been optimized for the real system (recentness). As the real system deviates further from simulation, it becomes better to use recent (more optimized) experience. We learn the parameters of the distance metric offline to optimize relevance over a simulation dataset.

We evaluate EHGO on three control problems, demonstrating that it consistently generates high-quality trajectories with a few hundreds of trials on the real system, even in the presence of severe dynamical mismatch. Our experience compares it with the DRL algorithm Proximal Policy Optimization (PPO) [3], showing that in these instances our proposed method achieves superior performance over PPO and substantially better data efficiency.

[1]Kuan-Yu Tseng, Mengchao Zhang, and Geir E. Dullerud are with the Department of Mechanical Science and Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801, USA {kuanyut2, mz17, dullerud} @illinois.edu

[2]Kris Hauser is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA, kkhauser@illinois.edu

## II. Related Work

Model-based methods are widely employed to compute optimal control policies for dynamical systems. This approach entails modeling the actual system, synthesizing control policies using this model, and subsequently implementing these policies on the real system. To enhance the performance of model-based controllers, a variety of methods leveraging data acquired from the real system have been developed. For instance, in the context of MPC [9], certain methods involve utilizing real-world data to refine the model. In [5], Gaussian process regression is employed to compensate for the nonlinear part of the car dynamics. In [10], a system identification technique is utilized to update the system model while running MPC. Generally, MPC utilizes an explicit model to compute an optimal policy for the system. Control performance may degrade significantly when a large error exists between the model and the system. Another notable strategy is ILC [6], which distinctively updates the control policy directly with real-world data, bypassing adjustments to the model itself. In [11], the authors implement an optimization-based ILC on a quadcopter's trajectory tracking problem by linearizing the system's model around a previous trajectory and updating the control policy using the linearized model. In [7], a gradient-based ILC combining trajectories gathered from real robotic operations and gradients computed via a simulated model—our closest representation of the real system—to refine the control policy. Similarly, in [12], an iterative online optimal feedback control method is proposed to compute near-optimal control actions of a system with an inexact model. The main limitation of ILC methods is their requirement for the robot to be reset to the same initial state in each iteration. This requirement can lead to inefficiencies or, in some cases, may not be feasible to implement.

In recent years, DRL [1], [2], [3] has shown its capability to learn an approximated optimal control policy in various robot control problems by letting an agent directly interact with the environment and using the collected experiences to update the policy [13], [14], [15]. Due to safety concerns and the time efficiency of training on a real robot platform, a control policy is often learned in a simulated environment and then applied to the real robot. However, the mismatch between the simulated model and the real system could hinder the performance of the learned policies, which is known as the sim-to-real gap [16]. Different techniques are utilized to bridge the gap, such as dynamics randomization [15], [14] and domain randomization [17]. Other approaches include training an ensemble of policies with randomized kinematic parameters and adapting to the real platform using Bayesian optimization [18]. In [19], the progressive net approach is proposed to transfer the learned policies from simulation to the real robot by laterally connecting a new network to an existing trained network. In [20], [21], the simulation parameters are adapted using real-world rollouts during policy training. In general, training a DRL policy involves tuning plenty of hyperparameters and requires a large amount of data samples to achieve desirable performance. Additionally, the performance of a trained policy might significantly degrade when fine-tuning on a new task, known as catastrophic forgetting [22].

## III. Approach

In this section, we describe the formulation of a learning control problem and our approach. Then, we introduce the GRILC algorithm, and in the following subsections extend its capability to learn a policy with an experience database and without starting from the same initial states over iterations.

### A. Problem Formulation

We consider discrete-time dynamical systems in the form

$$\boldsymbol{x}(i+1) = \boldsymbol{f}_R(\boldsymbol{x}(i), \boldsymbol{u}(i)), \tag{1}$$

where $\boldsymbol{f}_R(\cdot,\cdot)$ describes the dynamics of the real system. $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{u} \in \mathbb{R}^m$ denote the system state and control action, respectively. Given a query initial state $\boldsymbol{x}_0$, we aim to find the optimal trajectory from the initial state to a target $\boldsymbol{x}_{\text{tar}}$ as evaluated by an objective function. An equidistant time grid of size $N+1$ is used for discretizing the trajectory from time 0 to some final time $t_f$. The minimization problem is formulated as

$$J = \sum_{i=0}^{N-1} \ell(\boldsymbol{x}(i), \boldsymbol{u}(i), \boldsymbol{x}_{\text{tar}}) + \Phi(\boldsymbol{x}(N), \boldsymbol{x}_{\text{tar}}) \tag{2a}$$

$$\text{s.t.} \quad \boldsymbol{x}(i+1) = \boldsymbol{f}_R(\boldsymbol{x}(i), \boldsymbol{u}(i)) \ \forall i = \{0, \dots, N-1\} \tag{2b}$$

$$\boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{2c}$$

with $\ell$ the running cost and $\Phi$ the terminal cost.

We assume the real system dynamics $\boldsymbol{f}_R$ is usually unknown or partially known, and the state can be directly observed. The goal of the control problem is to achieve high-quality performance on the real system after observing a small amount of experience on the real system dynamics $\boldsymbol{f}_R$ from varying initial states $\boldsymbol{x}_0$.

### B. EHGO Overview

We assume that we have access to a nominal (simulation) model $\boldsymbol{f}_S$ of the system that approximates $\boldsymbol{f}_R$. Specifically, the GRILC method requires that the gradients of $\boldsymbol{f}_S$ and $\boldsymbol{f}_R$ have positive dot product across the state and control space. Since it is in simulation, we have access to a large amount of experience on $\boldsymbol{f}_S$, and can generate large numbers of optimal trajectories [8] or perform extensive reinforcement learning [3].

In the offline phase, EHGO begins by generating an experience database $\mathcal{S}_{(S+\Delta S)}$ of trajectories using $\boldsymbol{f}_S$ and perturbed dynamics $\boldsymbol{f}_{(S+\Delta S)}$ as the dynamics models. We also learn parameters of a distance metric $\text{Dist}(\Delta\boldsymbol{x}_0, \Delta r)$, which determines the relevance of an existing trajectory to a new query based on the deviation of the initial state $\boldsymbol{x}_0$ and the recentness of the experience $r$. In the online phase, our approach starts with $\mathcal{S} \leftarrow \mathcal{S}_S$ containing trajectories from $\boldsymbol{f}_S$ and then expands $\mathcal{S}$ as it receives more experience: After EHGO observes an initial state, it queries $\mathcal{S}$ to estimate a

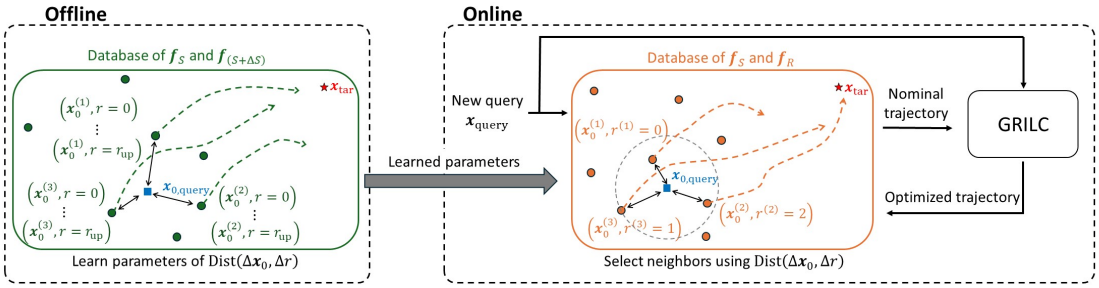


Fig. 1. Overview of EHGO. In the offline phase, we generate an experience database containing trajectories from $\boldsymbol{f}_S$ and $\boldsymbol{f}_{(S+\Delta S)}$. The initial states and trajectories are represented as dots and dashed curves in the database. We then learn parameters for the distance metric $\text{Dist}(\Delta \boldsymbol{x}_0, \Delta r)$, which measures the deviation of the initial state ($\boldsymbol{x}_0$) and the recentness of the experience ($r$). During the online phase, upon receiving a query, we generate a nominal trajectory by selecting neighbors from the database with trajectories of $\boldsymbol{f}_S$ and $\boldsymbol{f}_R$. This trajectory undergoes one iteration of the GRILC method, and the optimized trajectory is then added back into the database.

trajectory. The trajectory is then optimized by the GRILC method for one iteration and stored back in $\mathcal{S}$. The workflow of EHGO is summarized in Fig. 1. Below we will present details of this method.

### C. Hybrid Gradient Iterative Learning Control

To address the mismatch between $\boldsymbol{f}_S$ and $\boldsymbol{f}_R$, the GRILC [7] method employs a hybrid optimization that only requires gradients from $\boldsymbol{f}_S$. Combining trajectories collected on the real system $\boldsymbol{f}_R$ and gradient information calculated using the simulated system $\boldsymbol{f}_S$, the evolution of the real state trajectory can be approximated with the deviation of control variables and the initial state,

$$\boldsymbol{X}' \approx \boldsymbol{X} + \boldsymbol{M}_{\boldsymbol{u}}\hat{\boldsymbol{U}} + \boldsymbol{M}_{\boldsymbol{x}(0)}\hat{\boldsymbol{x}}(0), \tag{3}$$

where $\boldsymbol{X}' = \left[\boldsymbol{x}'(1), \boldsymbol{x}'(2), \ldots, \boldsymbol{x}'(N)\right]$ is the vectorized state trajectory predicted in the next iteration, $\boldsymbol{X} = \left[\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(N)\right]$ is the state trajectory measured on $\boldsymbol{f}_R$, and $\hat{\boldsymbol{U}} = \left[\hat{\boldsymbol{u}}(0), \hat{\boldsymbol{u}}(1), \ldots, \hat{\boldsymbol{u}}(N-1)\right]$ is the vectorized deviations of control variables between the current and the next iterations. $\boldsymbol{M}_{\boldsymbol{u}}$ and $\boldsymbol{M}_{\boldsymbol{x}(0)}$ are lifted matrices containing the Jacobian matrices of $\boldsymbol{f}_S$. $\hat{\boldsymbol{x}}(0)$ denotes the deviation of the initial state. Then the optimization problem (2a) can be transformed to

$$\min_{\hat{\boldsymbol{u}}(\cdot)} \sum_{i=0}^{N-1} \ell(\boldsymbol{x}'(i), \boldsymbol{u}(i) + \hat{\boldsymbol{u}}(i), \boldsymbol{x}_{\text{tar}}) + \Phi(\boldsymbol{x}'(N), \boldsymbol{x}_{\text{tar}}) \tag{4}$$

subject to (2c) and a trust region constraint to bound the control deviation $|\hat{\boldsymbol{u}}| \leq \hat{\boldsymbol{u}}_{\max}$. For a comprehensive derivation, we direct the reader to [7].

Therefore, starting from a nominal trajectory generated by $\boldsymbol{f}_S$, GRILC derives the state trajectory of the next iteration (3), and solves the optimization problem (4). The updated input trajectory is then executed on the real system, and the state trajectory is measured.

In a normal GRILC iteration, if the cost improves, the measured trajectory is fed back as the nominal trajectory for a new iteration. Otherwise, GRILC tightens the trust region $\hat{\boldsymbol{u}}_{\max}$ and solves the optimization again. Learning is terminated if $\hat{\boldsymbol{u}}_{\max}$ falls below a given threshold. As stated earlier, GRILC requires restarting repeatedly from the same initial state across iterations. In our work, we use a database to store experience and update the experience from arbitrary start states after each iteration of GRILC.

### D. Experience Database Query and Update

EHGO's experience database contains the information of trajectories we optimized offline for simulation dynamics as well as trajectories optimized online using GRILC. We *query* the database with an initial state $\boldsymbol{x}_0$ to obtain a guessed trajectory. If the discrepancy between the simulation and real system is minimal, then the simulation experience for starts similar to $\boldsymbol{x}_0$ will be highly relevant and their stored trajectories will work well for the query. However, if the real system differs significantly, we should prefer trajectories that have passed through multiple GRILC optimizations, possibly preferring a well-iterated trajectory with a far start state over a simulation-only trajectory with a close start state. Here we describe how we represent and query the database with a pseudo-distance metric that takes similarity and recentness into account.

The experience database is a set $\mathcal{S} = \left\{\left(\boldsymbol{x}_0^{(j)}, r^{(j)}, \boldsymbol{s}^{(j)}\right) \middle| j = 1, \ldots, |D|\right\}$. The nonnegative integer $r$ denotes the rank of the trajectory, indicating the number of iterations that the trajectory was updated by GRILC. The tuple $\boldsymbol{s} = (\boldsymbol{X}, \boldsymbol{U}, J)$ contains the state trajectory, the input trajectory, and the cost associated with the specific initial state $\boldsymbol{x}_0$. Experience with rank $r = 0$ represents raw trajectory information that has not been updated yet, which comes from trajectory optimization from sampled start states using the dynamics $\boldsymbol{f}_S$. Experience with $r > 0$ collects the trajectory information that was updated and implemented on the real system $\boldsymbol{f}_R$.

When processing a new query point, EHGO selects $k$-nearest neighbors from the experience set $\mathcal{S}$ and generates a nominal trajectory as a weighted average of the neighbors' trajectory. If a neighbor's initial state is close to the queried initial state $\boldsymbol{x}_{0,\text{query}}$, then its trajectory is more likely to exhibit a pattern similar to that of the trajectory starting from the query. Moreover, a neighbor with a higher rank implies its trajectory has undergone more updates, thereby rendering it more refined and suitable for adoption as a

**Algorithm 1** EHGO Online Learning

**Input:** Query point $\boldsymbol{x}_0$. Experience set $\mathcal{S}$.

1: Find k-nearest neighbors and their importance: $\{(\boldsymbol{s}^{(j)}, r^{(j)}, \rho^{(j)})\}_{j=1}^{k} = \text{KNN}(\boldsymbol{x}_0, \mathcal{S})$ {with distance function (5)}
2: Compute nominal trajectory: $\boldsymbol{s}_{\text{nom}} = \sum_j \rho^{(j)} \boldsymbol{s}^{(j)} / \sum_l \rho^{(l)}$
3: $\boldsymbol{s}_{\text{update}} = \text{GRILC}(\boldsymbol{x}_0, \boldsymbol{s}_{\text{nom}})$
4: $r_{\text{update}} = \max(\{r^{(j)}\}_{j=1}^{k}) + 1$
5: Augment: $\mathcal{S}$.insert $((\boldsymbol{x}_0, r_{\text{update}}, \boldsymbol{s}_{\text{update}}))$
6: **return** $\boldsymbol{s}_{\text{update}}$

nominal trajectory. To reflect this intuition, we design a distance function for the $k$-nearest neighbors as a weighted 1-norm combining the state-space distance and the rank distance:

$$\begin{aligned}&\text{Dist}(\Delta\boldsymbol{x}_0, \Delta r) := \\ &\quad \|\Delta\boldsymbol{x}_0\|_{2,\boldsymbol{w}_{\boldsymbol{x}}(\|\Delta r\|_{ps})} + w_r(\|\Delta r\|_{ps})\|\Delta r\|_{ps} + w_c,\end{aligned} \tag{5}$$

where the state space distance is the weighted 2-norm of the difference between the initial states of the neighbor and the query, i.e., $\Delta\boldsymbol{x}_0 := \boldsymbol{x}_0 - \boldsymbol{x}_{0,\text{query}}$ and $\|\Delta\boldsymbol{x}_0\|_{2,\boldsymbol{a}} = \sqrt{\sum_i \boldsymbol{a}(i)|\Delta\boldsymbol{x}_0(i)|^2}$. The rank distance is the difference between the rank of the neighbor and the upper bound, i.e., $\Delta r := r_{\text{up}} - r$, and is equipped with a pseudo norm as below,

$$\|\Delta r\|_{ps} = \begin{cases} \Delta r, & \text{if } \Delta r > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The upper bound $r_{\text{up}}$ and the pseudo norm fulfill a critical role: during initial optimization iterations, the trajectory undergoes significant changes, leading to a rapid decrease in the updated cost. As iterations progress, the trajectory and the cost gradually converge. Therefore, choosing neighbors with ranks higher than $r_{\text{up}}$ holds no significance. The weighting parameters $\boldsymbol{W}_{\boldsymbol{x}} = \big[\boldsymbol{w}_{\boldsymbol{x}}(0), \boldsymbol{w}_{\boldsymbol{x}}(1), \ldots, \boldsymbol{w}_{\boldsymbol{x}}(r_{\text{up}})\big]$, $\boldsymbol{w}_r = \big[w_r(0), w_r(1), \ldots, w_r(r_{\text{up}})\big]$, and $w_c$ corresponds to the importance of the state-space distance, rank distance, and a bias term. Note that $\boldsymbol{W}_{\boldsymbol{x}}$ and $\boldsymbol{w}_r$ depend on the rank distance and modulate how the distance metric is affected by recentness of experience.

Once the distance metric is chosen, the online update is summarized in Alg. 1. For a query point $\boldsymbol{x}_{0,\text{query}}$, $k$-nearest neighbors of the query point are found in $\mathcal{S}$ using the learned distance function (5), with their corresponding distances $\{d^{(j)}\}_{j=1}^{k}$. Next, the importance of each neighbor is determined by $\rho^{(j)} = \exp(-d^{(j)}/c)$. This acts as a softmax function to give more importance to closer neighbors to the query point. The nominal trajectory is obtained as the weighted average of the neighbors with importance and fed into GRILC to perform a one-iteration update. The updated trajectory information $\boldsymbol{s}_{\text{update}}$ and its rank $r_{\text{update}}$ are then inserted into the experience $\mathcal{S}$ for future use.

The number of neighbors $k$, the upper bound rank $r_{\text{up}}$, and the scale factor $c$ are hyperparameters of the EHGO query process. Increasing $k$ selects more neighbors for the nominal trajectory, but if it's too large, distant and less relevant experiences may be included. A larger $c$ sharpens the distribution $\rho$, placing larger importance on closer neighbors over others. The value of $r_{\text{up}}$ depends on the properties of an ILC method and the complexity of dynamics. One can set $r_{\text{up}}$ to a sufficiently large value; otherwise, $r_{\text{up}}$ can be chosen for each problem based on trajectory convergence criteria, potentially minimizing required training data. In this work, $r_{\text{up}}$ is determined for each problem by observing that the cost improvement using GRILC is less than $5\%$ after $r$ iterations.

### *E. Distance Metric Learning*

The metric weights $\boldsymbol{W}_{\boldsymbol{x}}$, $\boldsymbol{w}_r$, and $w_c$ are estimated by a regression procedure described below. This approach is known as metric learning [23], which involves constructing the distance metrics specific to a task to improve classification or clustering models. The total number of tunable parameters in the distance function is $(r_{\text{up}}+1)n+(r_{\text{up}}+1)+1$.

We observe that the logarithm of updated costs is roughly proportional to distances measured by the distance function, i.e., $\log J \propto \text{Dist}$. We can estimate the weighting parameters $\boldsymbol{W}_{\boldsymbol{x}}$, $\boldsymbol{w}_r$, and $w_c$ by regression techniques with training data. The regression is implemented offline using data collected in simulation before applying EHGO to online learning on the real system. Since the real system is unknown, we apply the dynamic randomization technique [15] to mimic $\boldsymbol{f}_R$. Formally, perturbations of dynamic parameters $\Delta S$ are randomly sampled from distributions such that we believe that the real system is close to the distribution of $\boldsymbol{f}_{(S+\Delta S)}$. The general procedure of estimation is described below:

1) Sample $|D_{\text{train}}|$ training points with optimal trajectories using $\boldsymbol{f}_S$ to build the database $\mathcal{S}_S = \left\{\left(\boldsymbol{x}_0^{(i)}, 0, \boldsymbol{s}^{(i)}\right)\right\}_i^{|D_{\text{train}}|}$
2) Sample $|D_{\text{query}}|$ query points $\left\{\boldsymbol{x}_{0,\text{query}}^{(j)}\right\}_{j=1}^{|D_{\text{query}}|}$.
3) Generate $m$ seeds of randomized dynamics $\boldsymbol{f}_{(S+\Delta S)}^{(m)}$. For each seed,
   a) Take every points in $\mathcal{S}_S$ and run GRILC on $\boldsymbol{f}_{(S+\Delta S)}^{(m)}$ with $r_{\text{up}}$ iterations, to build the database $\mathcal{S}_{(S+\Delta S)}^{(m)} = \left\{\left(\boldsymbol{x}_0^{(m,i)}, r^{(m,i)}, \boldsymbol{s}^{(m,i)}\right)\right\}_i$ with ranks from 0 to $r_{\text{up}}$.
   b) For each query, take every point in $\mathcal{S}_{(S+\Delta S)}^{(m)}$ as a nominal trajectory to run GRILC for one iteration. Collect the updated cost $\log J^{(m,i,j)}$, the state-space distance $\Delta\boldsymbol{x}_0^{(m,i,j)} = \boldsymbol{x}_{0,\text{query}}^{(j)} - \boldsymbol{x}_0^{(m,i)}$ and the rank distance $\Delta r^{(m,i,j)} = r_{\text{up}} - r^{(m,i)}$ to form the data pair $\left\{\left((\Delta\boldsymbol{x}_0^{(m,i,j)}, \Delta r^{(m,i,j)}), \log J^{(m,i,j)}\right)\right\}$
4) Perform a regression on $m \cdot |D_{\text{train}}| \cdot |D_{\text{query}}| \cdot (1 + r_{\text{up}})$ data pairs to fit the parameters $\boldsymbol{W}_{\boldsymbol{x}}$, $\boldsymbol{w}_r$, and $w_c$.

## IV. Experiments and Results

We evaluate the proposed EHGO framework on three benchmark problems as described below.

### A. Benchmark Systems

Each problem considers standard quadratic costs penalizing the state trajectory to the target and the input energy $\Phi(\boldsymbol{x}, \boldsymbol{x}_{\text{tar}}) = (\boldsymbol{x} - \boldsymbol{x}_{\text{tar}})^T \boldsymbol{P}(\boldsymbol{x} - \boldsymbol{x}_{\text{tar}})$ and $\ell(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}_{\text{tar}}) = (\boldsymbol{x} - \boldsymbol{x}_{\text{tar}})^T \boldsymbol{Q}(\boldsymbol{x} - \boldsymbol{x}_{\text{tar}}) + \boldsymbol{u}^T \boldsymbol{R}\boldsymbol{u}$, where $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$ denote the weights of the cost on the terminal state, the state, and the control trajectories, respectively.

*1) Pendulum:* The goal is to stabilize a pendulum with the following dynamics in an upright state:

$$\begin{aligned} \theta(i+1) &= \theta(i) + t_s \omega(i) \\ \omega(i+1) &= \omega(i) + t_s \left( -\frac{3g}{2l} \sin\theta(i) - \frac{3d}{ml^2}\omega(i) + \frac{3}{ml^2}\tau(i) \right) \end{aligned} \tag{7}$$

where the state $\boldsymbol{x} = [\theta, \omega]$ are the angle and the angular velocity and $t_s$ is the time step. The control is the torque applying on the fixed end of the pendulum with a state feedback law $\tau(i) = k_\theta(\theta_{\text{wp}}(i) - \theta(i)) + k_\omega(\omega_{\text{wp}}(i) - \omega(i))$, with the gains $k_\theta$ and $k_\omega$. The control variables $\theta_{\text{wp}}$ and $\omega_{\text{wp}}$ are waypoints of the state. The dynamics parameters $m, l, d$, and $g$ denote the mass, length, damping, and acceleration of gravity of the pendulum, respectively. We set the maximum torque bound sufficient to lift the pendulum directly to the upright position. Queries are drawn from the domain $\mathbf{x}_0 \in [-\pi, \pi] \times [-1, 1]$ and $t_f = 4$. The cost terms are $\boldsymbol{Q} = \boldsymbol{P} = \text{diag}\{1, 0.1\}$ and $\boldsymbol{R} = 0.001$.

*2) Car:* We consider a second order planar car model containing the following dynamics:

$$\begin{aligned} x(i+1) &= x(i) + t_s v(i) \sin(\theta(i)) \\ y(i+1) &= y(i) + t_s v(i) \cos(\theta(i)) \\ \theta(i+1) &= \theta(i) + t_s u_\theta(i) v(i) s_\theta \\ v(i+1) &= v(i) + t_s u_v(i) s_v \end{aligned} \tag{8}$$

where the state $\boldsymbol{x} = [x, y, \theta, v]$ includes planar coordinates, orientation, and speed. The control $\boldsymbol{u} = [u_\theta, u_v]$ contains actions changing the orientation and speed. The coefficients $s_\theta$ and $s_v$ represent turning rate and acceleration modulation, set to 1 in the nominal system. The goal is to control the car from a given query to the origin with zero orientation angle and velocity. Queries are drawn from the domain $\mathbf{x}_0 \in [-7.5, 0] \times [-7.5, 0] \times [0, \pi/2] \times [0, 3.1]$, and $t_f = 3$. Cost terms are designed to encourage the car to reach the target at the final time, with $\boldsymbol{Q} = \mathbf{0}, \boldsymbol{R} = 0.1\boldsymbol{I}_2$ and $\boldsymbol{P} = 1 \times 10^4 \boldsymbol{I}_4$, where $\boldsymbol{I}_n$ is the $n \times n$ identity matrix

*3) Quadcopter:* We consider a planar quadcopter model with the following dynamics:

$$\begin{aligned} x(i+1) &= x(i) + t_s \dot{x}(i) \\ z(i+1) &= y(i) + t_s \dot{z}(i) \\ \theta(i+1) &= \theta(i) + t_s \omega(i) s_\omega \\ \dot{x}(i+1) &= \dot{x}(i) + t_s u(i) \sin(\theta(i)) s_u \\ \dot{z}(i+1) &= \dot{z}(i) + t_s (u(i) \cos(\theta(i)) s_u - g) \end{aligned} \tag{9}$$

TABLE I

MISMATCHES OF DYNAMICS PARAMETERS TESTED IN EXPERIMENTS.

| | Simulated | Real |
|---|---|---|
| Pendulum | $l = 1, d = 0$ | $\Delta l = 50\%$, $d = 5$ |
| Car (small $\Delta$) | $s_\theta = s_v = 1$ | $\Delta s_\theta = 25\%$ $\Delta s_v = -20\%$, |
| Car (large $\Delta$) | $s_\theta = s_v = 1$ | $\Delta s_\theta = -50\%$, $\Delta s_v = 50\%$ |
| Quadcopter | $s_u = s_\omega = 1$ | $\Delta s_u = -25\%$, $\Delta s_\omega = 25\%$ |

where the state $\boldsymbol{x} = [x, z, \theta, \dot{x}, \dot{z}]$ comprises planar coordinates, orientation, and planar velocities. The control variables $\boldsymbol{u} = [u, \omega]$ are the thrust and the angular velocity of the quadcopter. Coefficients $s_u$ and $s_\omega$ scale the thrust and angular velocity controls. The goal is to control the quadcopter from a given query in the hover state to the origin with zero orientation angle and velocity. Queries are drawn from the domain $\mathbf{x}_0 \in [-10, 0] \times [-10, 10] \times [0, 0]^3$ and $t_f = 2.6$. We encourage the drone to reach the target at the final time with costs $\boldsymbol{Q} = \text{diag}\{1, 1, 1, 0.1, 0.1\}, \boldsymbol{R} = 0.1\boldsymbol{I}_2$ and $\boldsymbol{P} = 1 \times 10^5 \boldsymbol{I}_5$.

*4) Dynamics Mismatch and Parameter Selection:* We introduce errors in the simulated vs real dynamics as described in Tab. I. We study two magnitudes of disturbance for the Car problem.

Trajectories are discretized with grid size $N = 200$ for Pendulum, $N = 480$ for Car, and $N = 495$ for Quadcopter. Empirically nonlinear systems with larger dimensions require smaller time steps for GRILC to capture more accurate trajectories. We set $r_{\text{up}} = 1$ for Pendulum, $r_{\text{up}} = 5$ for Car, and $r_{\text{up}} = 6$ for Quadcopter. For all three systems, we set $k = 3$ and $c = 0.025$.

### B. Distance Metric Learning

The metric learning method described in Sec. III-E is applied to the Car and Quadcopter problems. For Car, we sample 5 seeds of dynamic parameters from the distributions $s_\theta \sim \text{Unif}[0.5, 1.5]$ and $s_v \sim \text{Unif}[0.5, 1.5]$ in simulation. We sample $|D_{\text{train}}| = 400$ training points and $|D_{\text{query}}| = 3$ query points. For Quadcopter, 5 seeds of dynamics parameters are sampled from $s_u \sim \text{Unif}[0.5, 1.5]$ and $s_\omega \sim \text{Unif}[0.5, 1.5]$. We sample $|D_{\text{train}}| = 1000$ training points and $|D_{\text{query}}| = 3$ query points.

Distance metric learning results on Quadcopter are shown in Fig. 2. Results for Car are similar. The left heatmap shows cost logarithms and the right heatmap shows predictions from the learned distance function, showing good alignment. The $x$-axis represents the state-space distance calculated using the estimated weights, i.e., $\|\Delta \boldsymbol{x}_0\|_{2, \boldsymbol{w}_{\boldsymbol{x}}(\|\Delta r\|_{ps})}$, while the $y$-axis presents the rank distance $\|\Delta r\|_{ps}$. In general, we observe that smaller state-space distances and rank distances have lower costs. Moreover, when rank distance is large (i.e., the experience is not recent), the state-space distance contributes less to the cost. In contrast, when the rank distance is small (i.e., the experience is recent), the state-space distance is highly correlated to cost.

For Pendulum, due to the system's simplicity, we bypass the metric learning stage and set the parameters of

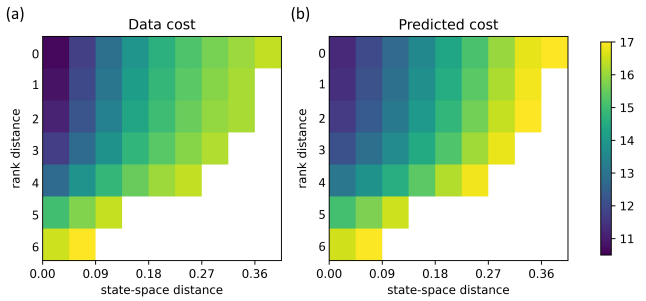


Fig. 2. Distance metric learning results for the Quadcopter example, showing (a) the log-costs and (b) the costs predicted by the distance function of the training data.

TABLE II

TRAINING AND TESTING SAMPLE COUNTS (TRAJECTORIES)

| | Pendulum | Car | Quadcopter |
|---|---|---|---|
| Pretraining | 500 | 36000 | 105000 |
| Online learning | 500 | 5000 | 5000 |
| Testing | 500 | 1600 | 1600 |

the distance function to some nominal values: $\boldsymbol{W_x} = [[1, 0.1], [1, 0.1]]$, $\boldsymbol{w}_r = [1, 1]$, and $w_c = 0$.

### C. Baseline Methods

We evaluate the learning performance of EHGO and compare it to variants of Proximal Policy Optimization (PPO) [3], a widely used model-free RL algorithm, as well as the idealized case of trajectory optimization on the real system dynamics.

*1) PPO Setup:* We use the Stable Baselines3 [24] implementation of PPO, and the dynamical systems are wrapped as gym environments [25]. Since our problems can be formulated as finite-time episodic RL problems, the remaining time is included as part of the agent's observation [26]. The length of an episode matches that of an iteration in EHGO.

Hyperparameter tuning is performed on each task via parameter sweeps. The learning rates are set to $1e^{-3}$, $1e^{-4}$, and $1e^{-4}$ for the Pendulum, Car, and Quadcopter examples, respectively. The hidden layers of MLP policy networks chosen for three examples are (64, 64) units with Tanh activation, (256, 256) units with ReLU activation, and (256, 256) units with ReLU activation, respectively. PPO performs an update every 4 episodes, and since our problems include large penalties in the terminal step, to enhance the stability we take the full batch to update the policy. Referring to the work [27], which demonstrates more robust performance across various environments, we incorporate state-dependent exploration during training. We experimented with training the models using Gaussian noise exploration but observed no significant improvement in the results. Other hyperparameters not mentioned are set to default values in Stable Baselines3.

*2) Training and Testing Procedures:* We establish three training protocols for PPO, designated as Robust PPO, Direct PPO, and Ideal Direct PPO. The training procedures are summarized as follows:

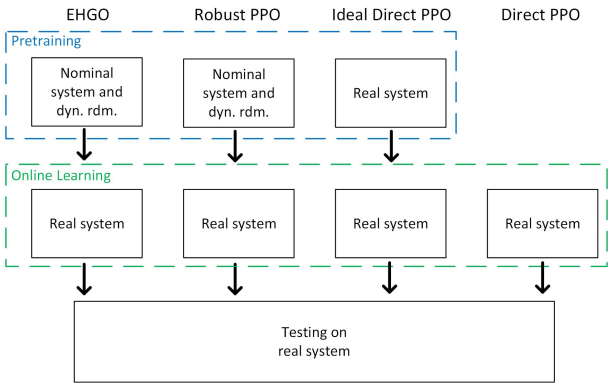


Fig. 3. Flowchart of training and testing. An equivalent amount of data is utilized for EHGO and PPO methods in each dashed color box.

- Robust PPO: pretrain a policy with the nominal system and dynamics randomization. Then, adapt online to real experience.
- Direct PPO: starting from randomly initialized weights, adapt to the real experience.
- Ideal Direct PPO: pretrain on the real system, then continue to gather and adapt online.

Specifically, Robust PPO is structured to emulate the offline and online learning phases of EHGO. Initially, the policy undergoes training within a simulation environment, utilizing system dynamics drawn from distributions. For the pendulum example, we sample the length $l \sim \text{Unif}[0.5, 1.5]$, and for the other two examples, parameter ranges are identical to those employed in EHGO metric learning. Subsequently, for online learning the policy undergoes fine-tuning. The Direct PPO setting models the case of no pretraining at all. Ideal Direct PPO represents an "idealized" training setting for RL in which the policy pretrained on the real system.

*3) Pretraining and online training:* To ensure fair comparison, an equivalent volume of data is utilized across both pretraining and online training for the EHGO and PPO approaches. The total number of sample points (i.e., trajectories) we used for three examples in training and testing are shown in Tab. II. By *pretraining* in EHGO we mean the size of the nominal trajectory database which is used to perform the metric learning, whereas in Robust PPO this is the number of training epochs with dynamics randomization, and in Ideal Direct PPO this indicates training epochs on the real system. The flowchart of the training procedure is shown in Fig. 3.

*4) Independent GRILC:* We also consider a fourth "idealized" baseline, *Independent* GRILC, which is defined as the resulting performance of the original GRILC algorithm after 10 optimization iterations starting from the same start state. Essentially, this acts as a trajectory optimizer that is given access to the true system dynamics.

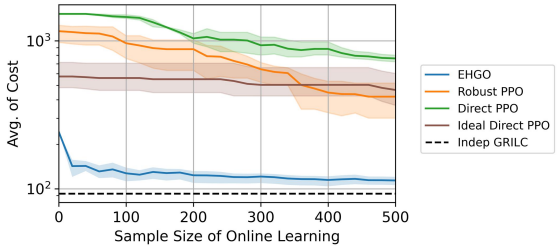


Fig. 4. Pendulum: trajectory cost versus sample size of online learning. Costs are plotted on a log scale.

### D. Results and Discussion

We examine the evolution of performance as each method is trained on progressive amounts of online data. Figs. 4, 5, and 6 display results for the Pendulum, Car, and Quadcopter problems, respectively. Note that the small mismatch and large mismatch Car problems are shown in Figs. 5(a) and 5(b), respectively. In each problem, three experiments are conducted with different randomization orders of the online training set, and we plot means and 25-75 percentile ranges of performance (cost for Pendulum and terminal distance for Car and Quadcopter) on the testing dataset.

The database query's running time for Pendulum, Car, and Quadcopter is 9, 45, and 62ms, respectively. The GRILC computation speed for the three examples is 0.4, 0.38, and 1.34s, respectively. Computation time for a GRILC iteration involves linearizing and vectorizing a trajectory, conducting optimization, and executing on the real system. We note that our framework is implemented in Python, and a careful C++ implementation can reduce the running time.

In general, EHGO is able to adapt quickly to the dynamics mismatch in most examples. It performs strictly better than Direct PPO, and surpasses Robust PPO within a few hundreds of iterations. Moreover, it even outperforms Ideal Direct PPO relatively quickly, surpassing its performance for both the small-mismatch Car and Quadcopter at around 900 and 100 samples, respectively. In the worst-case scenario, EHGO's performance still surpasses that of PPO after training on 2400 samples in the large-mismatch Car. In Pendulum, EHGO has approximately reached optimality, while for Car and Quadcopter, there is still room for improvement after 5,000 iterations. This is likely due to the larger state space in these examples, which requires more samples with high rank to cover the space with sufficient density.

Fig. 7 shows 50 sampled trajectories from the testing dataset on Car example after 5000 iterations of training. The arrows show the direction the car is heading. The red star denotes the target point on the plane, and the blue dots indicate the final states of the testing trajectories. Observe that EHGO trajectories reach a tighter cluster around the target point. Interestingly, the trajectories generated by PPO have distinct patterns from those of GRILC and EHGO, in which the car tends to steer abruptly first and drives more directly to the target. Fig. 8 shows 50 samples on Quadcopter, displaying similar trends.

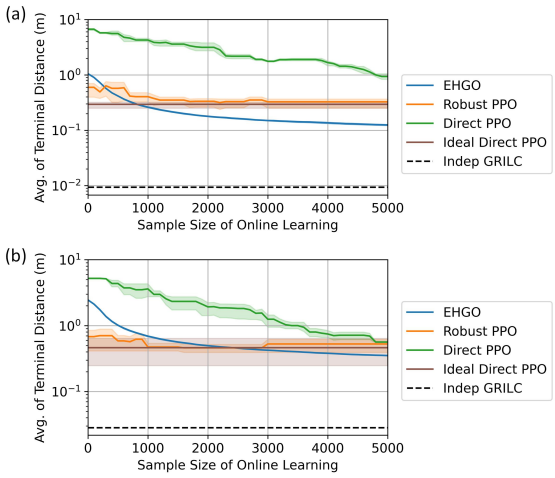


Fig. 5. Car: terminal distances versus sample size of online learning for the (a) small mismatch and (b) large mismatch cases.

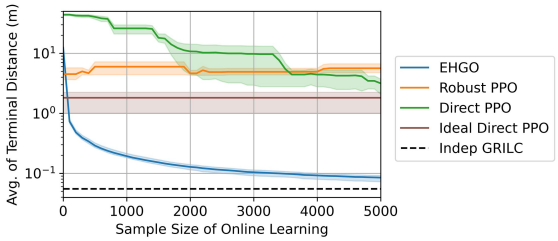


Fig. 6. Quadcopter: terminal distances versus sample size of online learning.

## V. Conclusions

We demonstrated that the presented EHGO method can adapt a trajectory database to systems with significant mismatch between the real and reference dynamics. Moreover, the adaptation is rapid and on the examples presented outperforms reinforcement learning methods within a few hundreds of samples of experience. Using a hybrid approach, EHGO does not require an explicit model of the real system and instead makes use of gradients from the reference.

In future work we would like to address the limitation that our current approach requires full trajectory executions before adapting the database. To include this approach in an MPC setting, EHGO would need to adapt trajectories after partial execution of the trajectory. Moreover, although the use of a weighted $k$-nearest neighbor scheme works well in problems with relatively low-dimensional state spaces (up to 5 in our experiments), its performance is likely to degrade as dimensionality grows. We are interested in exploring other approaches, such as deep neural networks or other function approximators, to predict trajectories more quickly in more complex domains. Additionally, we plan to conduct experimental evaluations on real platforms.

## References

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

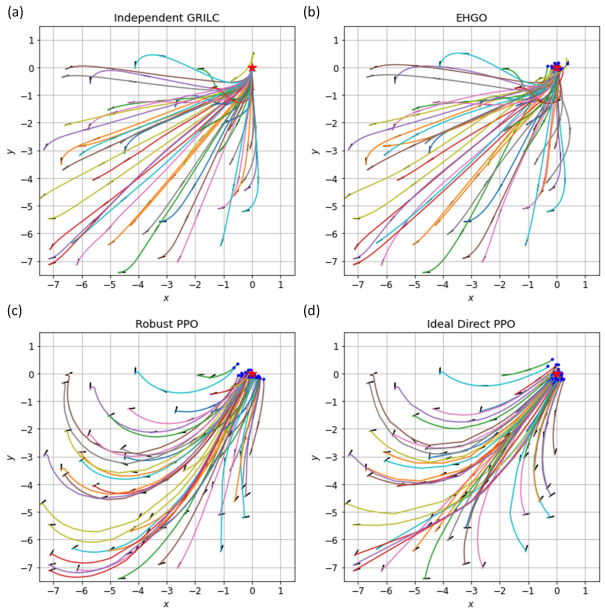


Fig. 7. Sampled learned trajectories for the Car example: (a) Independent GRILC, (b) EHGO, (c) Robust PPO, and (d) Ideal Direct PPO.

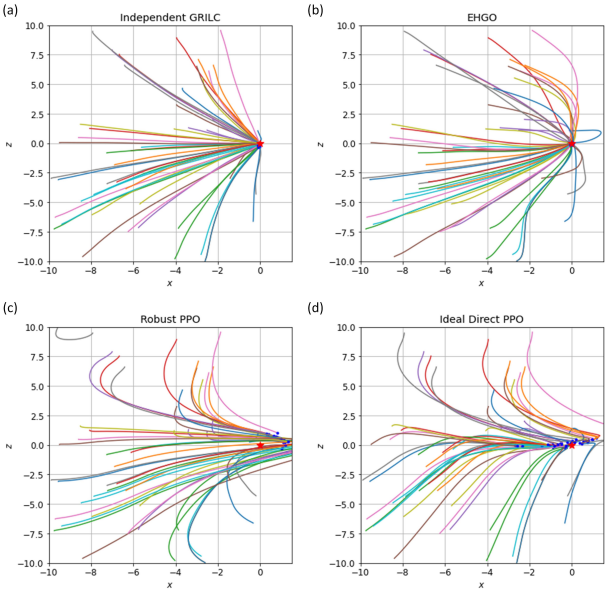


Fig. 8. Sampled learned trajectories for the Quadcopter example: (a) Independent GRILC, (b) EHGO, (c) Robust PPO, and (d) Ideal Direct PPO.


[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

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

[4] C. J. Ostafew, A. P. Schoellig, T. D. Barfoot, and J. Collier, "Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking," *Journal of Field Robotics*, vol. 33, no. 1, pp. 133–152, 2016.

[5] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[6] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE control systems magazine*, vol. 26, no. 3, pp. 96–114, 2006.

[7] K.-Y. Tseng, J. S. Shamma, and G. E. Dullerud, "Low-fidelity gradient updates for high-fidelity reprogrammable iterative learning control," in *2022 American Control Conference (ACC)*, 2022, pp. 4772–4777.

[8] G. Tang and K. Hauser, "A data-driven indirect method for nonlinear optimal control," *Astrodynamics*, vol. 3, no. 4, pp. 345–359, Dec. 2019.

[9] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[10] U. Rosolia and F. Borrelli, "Learning how to autonomously race a car: a predictive control approach," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2713–2719, 2019.

[11] A. P. Schoellig, F. L. Mueller, and R. D'andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33, pp. 103–127, 2012.

[12] Y. Chen and D. J. Braun, "Iterative online optimal feedback control," *IEEE Transactions on Automatic Control*, vol. 66, no. 2, pp. 566–580, 2020.

[13] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[14] E. Chisari, A. Liniger, A. Rupenyan, L. Van Gool, and J. Lygeros, "Learning from simulation, racing in reality," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8046–8052.

[15] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[16] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[18] I. Exarchos, Y. Jiang, W. Yu, and C. K. Liu, "Policy transfer via kinematic domain randomization and adaptation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 45–51.

[19] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on robot learning*. PMLR, 2017, pp. 262–270.

[20] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," *arXiv preprint arXiv:1702.02453*, 2017.

[21] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8973–8979.

[22] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[23] B. Kulis *et al.*, "Metric learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2013.

[24] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[25] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[26] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev, "Time limits in reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4045–4054.

[27] A. Raffin, J. Kober, and F. Stulp, "Smooth exploration for robotic reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 1634–1644.