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

2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 979-8-3503-7770-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/IROS58592.2024.10801637

Skill Transfer and Discovery for Sim-to-Real Learning: A Representation-Based Viewpoint

Haitong Ma, Zhaolin Ren, Bo Dai, Na Li

***Abstract*— We study sim-to-real skill transfer and discovery in the context of robotics control using representation learning. We draw inspiration from spectral decomposition of Markov decision processes. The spectral decomposition brings about representation that can linearly represent the state-action value function induced by any policies, thus can be regarded as skills. The skill representations are transferable across arbitrary tasks with the same transition dynamics. Moreover, to handle the sim-to-real gap in the dynamics, we propose a skill discovery algorithm that learns new skills caused by the sim-to-real gap from real-world data. We promote the discovery of new skills by enforcing orthogonal constraints between the skills to learn and the skills from simulators, and then synthesize the policy using the enlarged skill sets. We demonstrate our methodology by transferring quadrotor controllers from simulators to Crazyflie 2.1 quadrotors. We show that we can learn the skill representations from a single simulator task and transfer these to multiple different real-world tasks including hovering, taking off, landing and trajectory tracking. Our skill discovery approach helps narrow the sim-to-real gap and improve the real-world controller performance by up to 30.2%.**

I. Introduction

Reinforcement learning (RL) has demonstrated superior performance in many robotic simulators [1], [2]. However, transferring the controllers learned in simulators to real robots has long been a very challenging question in the RL community. One difficulty of sim-to-real transfer is that the learned policies are highly specific to the dynamics and tasks in the simulators [3]–[6], making them difficult to generalize to many real-world tasks, in which sim-to-real gaps exist.

Recent studies in the theoretical RL community on the spectral decomposition of Markov decision processes (MDPs) [7]–[10] reveal the idea of task-independent representations for RL. The results of spectral decomposition are the spectral functions of the transition dynamics in MDPs. The spectral functions can *linearly* represent the state-action value function, i.e., the Q-function, *induced by any policy*. Therefore, we say the spectral functions are task-independent representations of *skills*, because the spectral functions include information needed to accomplish any tasks. These task-independent skill representations are shared across arbitrary tasks, thus are reusable and transferable. Meanwhile, the representation can also be used to synthesize policies.

Haitong Ma, Zhaolin Ren, Na Li are with School of Engineering and Applied Sciences, Harvard University. Bo Dai is with School of Computational Science and Engineering, Georgia Institute of Technology. Email: {haitongma, zhaolinren}@g.harvard.edu, bodai@google.com, nali@seas.harvard.edu. The work is supported under NSF AI institute: 2112085, NSF CNS: 2003111, NSF ECCS: 2328241, 2401390, 2401391.

The Github link for codes, videos, and full report is available at https://congharvard.github.io/steady-sim-to-real/

Given the representation-based skill sets and a specific task, we can perform sample-efficient planning upon the skill sets to synthesize the optimal policy. When the representations are unknown, sample-efficient algorithms have been proposed to learn the representations from data [10], [11].

Nevertheless, these representation-based skill learning are still designed for specific transition dynamics. When it applies to sim-to-real transfer, the sim-to-real gap, which will induce new skills different from the simulator skill sets, has not been investigated yet. Learning the sim-to-real gap from real-world data, also called residual dynamics learning [4], [12]–[14], naturally aligns with our representation learning viewpoint. However, naively learning the representations of residual dynamics might lead us to relearn redundant skills that are linearly dependent with the existing simulator skill sets. Therefore, we need additional incentives to *discover* new skills that enable us to bridge the sim-to-real gap.

To further leverage the transferability of the representation-based skill sets and discover new skills induced by the sim-to-real gap, we proposed the **S**kill **T**ransf**E**r **A**nd **D**iscover**Y** **(STEADY)** for sim-to-real representation learning algorithm. We show that recent theoretical representation learning algorithms for spectral decomposition of MDPs, such as [10], [15], can apply to learning transferable representations of real-world robots. Moreover, we handle the sim-to-real gap by augmenting distinct representation-based skills learned from the sim-to-real gap, which we refer to as *skill discovery*. During the learning process, orthogonal constraints between the newly discovered skill sets and the simulator skill sets are enforced to fill the sim-to-real gap. In this way, we ensure that the skills necessary for the real robots are also included in our augmented skill sets, upon which the planning can be handled in a more complete space efficiently.

We demonstrate our proposed algorithm by transferring the learned quadrotor control policies to Crazyflie 2.1 quadrotors. First, we learn the representations and train a tracking controller on the simulator. Then, we collect task-specific data, including hovering, taking-off, landing, and trajectory tracking in the real world for skill transfer and discovery. The results show that our representation-based skills are easily transferred to the real world and generalizable to different tasks, and that our method has improved the real-world tracking performance by up to 30.2%.

II. Related works

A. Sim-to-Real Transfer

Sim-to-real transfer aims to transfer knowledge from simulator to real-world robots and overcome the sim-to-real gap.

We focus on sim-to-real gaps in terms of the dynamics. Representative sim-to-real transfer techniques include domain randomization and residual dynamics learning.

1) Domain randomization: Domain randomization in RL refers to randomly perturbing the physical parameters of the simulators [3]–[5], [16]–[18]. Then the RL agents aim to learn a policy performing well under a distribution of transition dynamics. The trained policies are directly applied to the real world and no knowledge will be learned from real-world data. These methods are convenient since simulator data are usually cheap to sample. However, domain randomization cannot handle other sim-to-real gaps like aerodynamics, contact effects, response delays, etc.

2) Residual dynamics learning: Learning the sim-to-real gap, or the residual dynamics, appears in many recent sim-to-real transfer studies [4], [12]–[14], [19], [20]. Real-world data are collected to learn the sim-to-real gap and improve the policies. Existing practices have considered different models, including Gaussian mixture model [19], Gaussian process [12], [21], k-nearest neighbors [4], or deep neural networks [13], [14], [20]. Then the learned sim-to-real gap is integrated with the prior simulator knowledge as external disturbance [14], [20], [21] or additive controllers [13].

We also follow the residual dynamics learning idea to fill the sim-to-real gap, but through a novel representation view. We use the knowledge of the simulator skill sets in the residual dynamics learning process in the way that enforces orthogonal constraints between simulator skill sets and the new skill sets, expanding skill sets while preventing redundancy of learning skills that are already in the simulator skill sets.

B. Representation-Based Knowledge Transfer

Representation-based knowledge transfer is commonly seen for computer vision models and visual-input control tasks [22], [23]. For dynamics control tasks, [24] decomposed the policy networks into task-specific and robot-specific modules and show that the modules are transferable. [25] proposed transfer learning by matching the transfer functions, where transfer functions can be regarded as another type of representation but limited to single-input-single-output dynamical systems. Our approach applies to general nonlinear dynamical systems and the transferability is justified rigorously by theoretical analysis. For theoretical representation learning, [26] has proved that there are provable benefits on sampling complexity for transferring the representations. However, no experimental results on simulators or in the real world have been reported.

III. Preliminaries

A. Notations and Sim-to-Real Problem Setting

Markov Decision Processes (MDPs) are a standard sequential decision-making model for RL, and can be described as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, P, \rho, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition operator with $\Delta(\mathcal{S})$ as the family of distributions over $\mathcal{S}$, $\rho \in \Delta(\mathcal{S})$ is the initial distribution and $\gamma \in (0, 1)$ is the discount factor. The goal of RL is to find a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the infinite-horizon cumulative discounted reward

$$\mathbb{E}_{s_0 \sim \rho, \pi}\left[\sum_{i=0}^{\infty} \gamma^i r\left(s_i, a_i\right) \mid s_0\right]$$

by interacting with the MDP. The value function under transition dynamics P and policy π is defined as $V_P^\pi(s) = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i r\left(s_i, a_i\right) \mid s_0 = s\right]$, and the state-action value function under transition dynamics P is $Q_P^\pi(s, a) = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i r\left(s_i, a_i\right) \mid s_0 = s, a_0 = a\right]$. When doing off-policy learning we slightly abuse the notation of $\mathcal{B}$ to denote any data distribution sampled from the off-policy data set of replay buffer.

The sim-to-real problem indicates that we have a simulator of the real-world MDP $\mathcal{M}$, which is also an MDP $\mathcal{M}^\circ = (\mathcal{S}, \mathcal{A}, r^\circ, P^\circ, \rho^\circ, \gamma)$. Notations with superscript $^\circ$ means they are related to the simulator. The two MDPs might differ in the transition dynamics, P° and P, initial distributions ρ°, ρ and rewards r°, r. The simulator is cheap to query and the real world is expensive. Therefore, we split the learning procedure into the simulator stage and the real-world transfer stage. In the simulator stage, the agent interacts with $\mathcal{M}^\circ$ for sufficiently many transitions to obtain knowledge from the simulators. Then in the real-world stage, the agent transfers knowledge learned in the simulator to solve the optimal policy of $\mathcal{M}$, while only collecting a limited number of transitions by interacting with $\mathcal{M}$.

B. Spectral Decomposition and Skills in Markov Decision Processes

The formal definition of spectral decomposition of MDPs refers to the following structures on the transition dynamics and rewards,

Definition 1 (Spectral decomposition of MDPs, [7], [9])**.** The spectral decomposition of an MDP $\mathcal{M}^\circ$ with transition dynamics $P^\circ\left(s' \mid s, a\right)$ means there exists representations $\phi^\circ : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and $\mu^\circ : \mathcal{S} \to \mathbb{R}^d$ such that

$$P^\circ\left(s' \mid s, a\right) = \left\langle \phi^\circ(s, a), \mu^\circ\left(s'\right)\right\rangle, \quad r(s, a) = \left\langle \phi^\circ(s, a), \theta_r\right\rangle$$

where $\theta_r \in \mathbb{R}^d$ and $\langle \cdot, \cdot \rangle$ denotes the vector inner product.

The spectral decomposition enables that the representation ϕ can linearly represent the state-action value function $Q_{P^\circ}^\pi$ *for any policy* π,

$$Q_{P^\circ}^\pi(s, a) = \left\langle \phi^\circ(s, a), w^\pi\right\rangle \tag{1}$$

where $w^\pi = \theta_r + \gamma \int_{\mathcal{S}} V_{P^\circ}^\pi\left(s'\right) \mu\left(s'\right) ds'$. The linear structure can be obtained by the recursive relationship between the $Q_{P^\circ}^\pi$ and $V_{P^\circ}^\pi$,

$$\begin{aligned} Q_{P^\circ}^\pi(s, a) =& r(s, a) + \gamma \int_{\mathcal{S}} P^\circ(s'|s, a) V_{P^\circ}^\pi(s') ds' \\ =& \left\langle \phi^\circ(s, a), \underbrace{\theta_r + \gamma \int_{\mathcal{S}} V_{P^\circ}^\pi\left(s'\right) \mu^\circ\left(s'\right) ds'}_{w^\pi} \right\rangle. \end{aligned}$$

Representation-based skills. Note that the linear structure of Q-function holds for any policies under dynamics P°. We argue that ϕ° can be regarded as the skill sets under dynamics P°, since it includes the information of constructing arbitrary

policy. Therefore, we interpret the representation ϕ° as the task-independent *skill sets*, which includes the information of all the skills needed for the model P°. Formally, given a Q-function, it induces the max-entropy policy as

$$\pi_Q(a \mid s) := \frac{\exp\left(Q(s,a)/\tau\right)}{\sum_{a\in\mathcal{A}} \exp\left(Q(s,a)/\tau\right)} \tag{2}$$

$$= \underset{\pi(\cdot|s)\in\Delta(\mathcal{A})}{\arg\max}\ \mathbb{E}_\pi[Q(s,a)] + \tau H(\pi) \tag{3}$$

where π_Q is the greedy max-entropy policy given a Q-function, $H(\pi) := \sum_{a\in\mathcal{A}} \pi(a \mid s) \log \pi(a \mid s)$. Therefore, if we know the skill sets ϕ°, we can construct the max-entropy policies $\pi(a \mid s) \propto \exp(w^\top \phi^\circ(s,a))$ from skills $\phi^\circ(s,a)$.

Practical Implementations. For practical implementations, the policy evaluation can be conducted by minimizing the temporal-difference error w.r.t. the parameter w, i.e.,

$$\min_w \mathbb{E}_{(s,a,r,s')\sim\mathcal{B},a'\sim\pi(\cdot|s')}[(r + \gamma\overline{Q}(s',a') - w^\top\phi^\circ(s,a))^2] \tag{4}$$

where $\mathcal{B}$ is the data distribution in the replay buffer, and $\overline{Q}$ is the target Q-function commonly used in the target network trick. We emphasize that compared to the deep Q-learning [27], the policy evaluation optimization is in the linear space spanned by the learned skill sets, therefore, is more stable. The policy improvement is the same as in other off-policy algorithms that optimize (3) by policy gradient. Practical implementations like soft actor-critic uses the reparameterization trick to simplify the calculations of (3) [2]. We will discuss our policy parameterization in the experimental setup in Section V-A.

C. Skill Learning by Spectral Conditional Density Estimation

If the representation ϕ°, μ° is unknown, we need to learn these representations from data. The objective is to make $\langle\phi^\circ(s,a), \mu^\circ(s')\rangle$ as close as $P^\circ(s' \mid s,a)$ as possible. There are multiple statistical learning methods to solve the problem [10], [11], [28]. We follow the spectral conditional density estimation method, which optimizes the following loss function,

$$\min_{\phi^\circ,\mu^\circ} \mathbb{E}_{(s,a)\sim\mathcal{B},s'\sim P^\circ(\cdot|s,a)} \left[\|P^\circ(s' \mid s,a) - \langle\phi^\circ(s,a), \mu^\circ(s')\rangle\|_2^2\right] \tag{5}$$

where $\mathcal{B}$ denotes the offline dataset distribution. However, (5) is usually intractable since we usually do not know the exact value of $P^\circ(s' \mid s,a)$. This necessitates the use of sampling-based algorithms. For example, [10] uses a surrogate loss function that is equivalent with (5):

$$L_{\text{feat}}(\phi^\circ,\mu^\circ) := C - 2\mathbb{E}_{(s,a)\sim\mathcal{B},s'\sim P(s'|s,a)}\left[\phi(s,a)^\top\mu(s')\right] + \mathbb{E}_{(s,a)\sim\mathcal{B}}\left[\int_{\mathcal{S}} \left(\phi(s,a)^\top\mu(s')\right)^2 \,\mathrm{d}s'\right], \tag{6}$$

where C is a constant independent of ϕ°, μ°. For practical implementations, we can parameterize the ϕ°, μ° both as neural networks (with matching output layer dimensions) and doing gradient descent on L_{feat}.

IV. STEADY: Skill Transfer and Discovery for Sim-to-Real Learning

In the following two sections, we introduce our methodology of skill transfer and discovery **(STEADY)** for sim-to-real learning. The whole process includes the following steps:

(i) Learning skill sets and policy in the simulator;
(ii) Discovery of new skills from real-world data;
(iii) Policy synthesis from skill sets.

An overview of the whole framework is shown in Figure 1.

A. Learning Skill Sets in the Simulator

We can leverage the previously mentioned representation learning techniques to learn the simulator skill sets ϕ° and μ°, as well as policy upon the skill sets. In this paper, we follow a similar procedure to the spectral decomposition representation learning (SPEDER) algorithm in [10], which is summarized in Algorithm 1.

Algorithm 1 Spectral Decomposition Representation **(SPEDER)** for reinforcement learning [10]

Input: simulator MDP $\mathcal{M}^\circ$
1: **for** episode $n = 1, 2, \ldots, N$ **do**
2: Collect transitions (s,a,r,s') and updates buffer $\mathcal{B}$
3: Learn representation ϕ° by minimizing (6)
4: Update Q function by equation (4)
5: Update policy by (3)
6: **end for**
Output: simulator policy π°, representations ϕ°, μ°, parameters w^{π°

B. Skill Discovery from Real-World Data

After the simulator stage, we transfer learned simulator representations/skill sets, namely ϕ°, to real robots. However, simply applying the policies π° learned from the simulator to the real world, like in zero-shot sim-to-real transfer, might lead to problems due to the sim-to-real gap. Therefore, the major motivation for skill discovery is learning new skills induced by the sim-to-real gap.

Following a similar procedure of assuming the decomposition structure on P°, we can assume that the sim-to-real gap also admits a spectral decomposition.

Assumption 1 (Spectral decomposition of sim-to-real gap)**.** There exists $\phi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}^s$ and $\mu : \mathcal{S} \to \mathbb{R}^s$ such that

$$P(s'|s,a) - P^\circ(s'|s,a) = \langle\phi(s,a), \mu(s')\rangle\,.$$

The spectral decomposition allows us to efficiently learn the gap from expensive real-world data, and then a good policy (and potentially more realistic simulators) given current knowledge of the existing simulator. We then learn the residual dynamics by formulating and solving the following least-square style optimization problem.

$$\min_{\phi,\mu} \mathbb{E}_{(s,a)\sim\rho_0} \|P(\cdot \mid s,a) - P^\circ(s' \mid s,a) - \langle\phi(s,a), \mu(s')\rangle\|_2^2\,. \tag{7}$$

Enforce Skill Discovery by Constraints. Before we proceed to solve the (7), we need to make sure that we are *discovering*

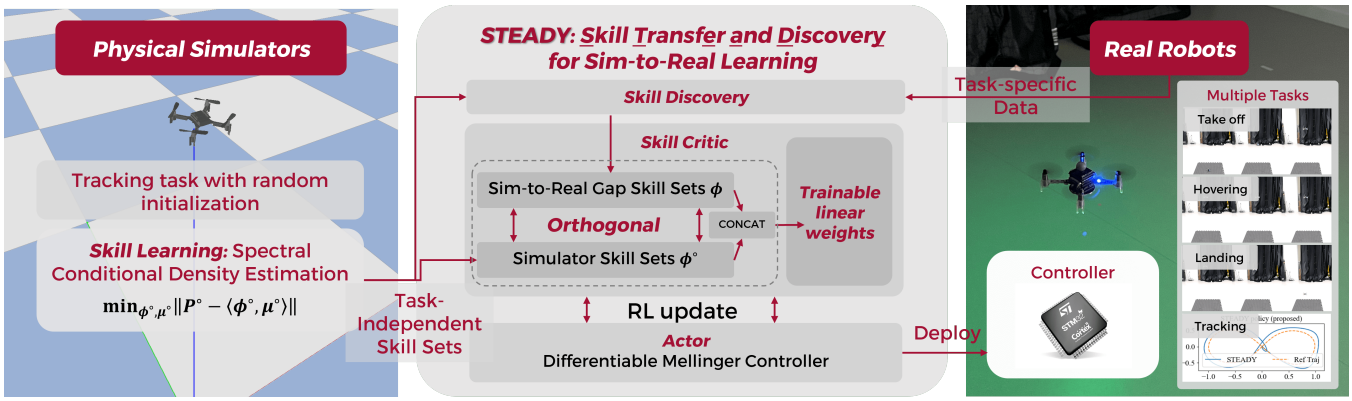


Fig. 1: Overview of the **STEADY** framework for sim-to-real learning. More information can be found on our project website.

new skills, which means that the skills learned from the real-world data are different from the previous simulator skills. In mathematical terms, having different skills means that the newly learned representations must be *linearly independent* with all the previous representations. Otherwise, we could simply represent the new skills by a linear combination of the previous skills. In such an undesirable case, the new skills will be redundant when we use the representations to linearly represent the Q-function in (1). To prevent this and enforce linear independence between the new and simulator skill sets, we add the orthogonal constraints to (7)

$$\begin{aligned} &\min_{\phi,\mu} \mathbb{E}_{(s,a)\sim\rho_0} \left\| P(\cdot \mid s,a) - P^\circ(\cdot \mid s,a) - \phi(s,a)^\top \mu(\cdot) \right\|_2^2 \\ &\text{s.t. } \langle \phi_i^\circ, \phi_j \rangle = 0, \forall i \in \{1,2,\ldots,d\},\ \forall j \in \{1,2,\ldots,s\}. \end{aligned} \tag{8}$$

where the inner product is defined as $\langle \phi_i^\circ, \phi_j \rangle = \mathbb{E}_{(s,a)\sim\mathcal{B}} [\phi_i^\circ(s,a)\phi_j(s,a)]$. The orthogonal constraints enforce the linear independence between simulator skill sets ϕ° and newly learned skill sets ϕ.

Practical Implementations of Skill Discovery. For the practical implementation of the skill discovery in (8), the minimization problem (ignoring the constraints) is similar to the problem in (5). We don't know the exact value of P°, so we use the learned representation $\langle \phi^\circ, \mu^\circ \rangle$ to replace P°. Then we follow a similar idea in (6) to minimize

$$\begin{aligned} L_{\text{disc}}(\phi,\mu) = &-2\mathbb{E}_{(s,a)\sim\mathcal{B},s'\sim P(s'|s,a)} \left[\begin{bmatrix} \phi^\circ(s,a) \\ \phi(s,a) \end{bmatrix}^\top \begin{bmatrix} \mu^\circ(\cdot) \\ \mu(\cdot) \end{bmatrix} \right] \\ &+\mathbb{E}_{(s,a)\sim\mathcal{B}} \left[\int_{\mathcal{S}} \left(\begin{bmatrix} \phi^\circ(s,a) \\ \phi(s,a) \end{bmatrix}^\top \begin{bmatrix} \mu^\circ(\cdot) \\ \mu(\cdot) \end{bmatrix} \right)^2 \mathrm{d}s' \right], \end{aligned} \tag{9}$$

where the simulator skill sets ϕ°, μ° are fixed in the skill discovery stage. We transfer the constraints in (9) to soft constraints by penalty methods for training stability, which leads to the following empirical loss function,

$$\min_{\phi,\mu} L_{\text{disc}}(\phi,\mu) + \lambda \sum_{i,j} |\langle \phi_i^\circ, \phi_j \rangle|, \tag{10}$$

where λ is a hyperparameter penalizing the constraint violations.

C. Policy Synthesis for Real-World Tasks

Real-world Policy Evaluation. After we learn the representations ϕ, μ, we can leverage the augmented skill sets, $[\phi^\circ, \phi]$, to synthesize the policies for specific real-world tasks by the MDP planning algorithm such as policy iteration. Here, we also follow a similar policy iteration algorithm to the one used in the simulator stage, while changing the skill sets/representations to be $[\phi^\circ, \phi]$. Similarly, in the policy evaluation stage, we parameterize the Q-function by a linear combination of the enlarged skill sets $[\phi^\circ, \phi]$, $Q(s,a;w) = w_1^\top \phi^\circ(s,a) + w_2^\top \phi(s,a)$. We can minimize the TD error,

$$\begin{aligned} \min_{w_1,w_2} \mathbb{E}_{(s,a,r,s')\sim\mathcal{B},a'\sim\pi(\cdot|s')} [(r + \gamma \overline{Q}(s',a') \\ - w_1^\top \phi^\circ(s,a) - w_2^\top \phi(s,a))^2] \end{aligned} \tag{11}$$

Then we can optimize the policy similar to (3) with the new linear parameterization of Q-function.

Policy Synthesis. When initializing the real-world stage, we initialize the policy by simulator policy π° and weights w_1 by the simulator learned weights w^{π° so that we do not need to learn the Q-function from scratch. The skill discovery and policy synthesis are conducted simultaneously, and the algorithm is listed in Algorithm 2. To avoid the instability caused by changing from the simulator to the real world, we penalize the KL-divergence between the simulator policy and the updated policy to (3) when solving the real-world policy improvement,

$$\max_{\pi} \ \mathbb{E}_{s\sim\mathcal{B},a\sim\pi(\cdot|s)}[Q(s,a)] - \tau_\pi \bar{D}_{KL}(\pi \| \pi^\circ) \tag{12}$$

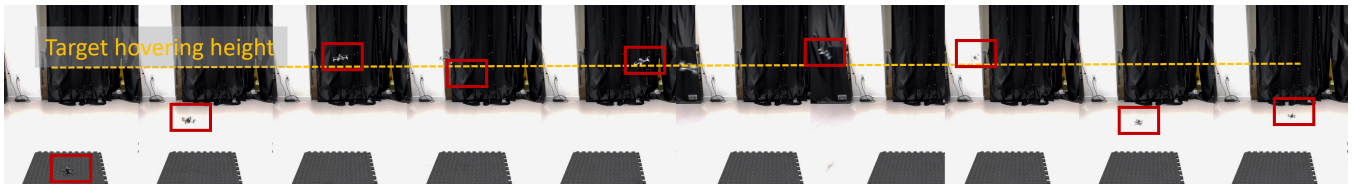
where $\bar{D}_{KL}(\pi\|\pi^\circ) = \mathbb{E}_s [D_{KL}(\pi(\cdot \mid s) \| \pi^\circ(\cdot \mid s))]$, τ_π is a hyperparameter penalizing policy update.

Practical Implementations of Policy Synthesis. In practical implementations, we add several modifications to the soft actor-critic (SAC) [2] algorithm. We change the Q-function parameterization to the linear representation $w_1^\top \phi^\circ(s,a) + w_2^\top \phi(s,a)$. The representations $\phi, \mu, \phi^\circ, \mu^\circ$ are all parameterized by fully connected neural networks. The proposed framework is compatible to any policy parametrizations.

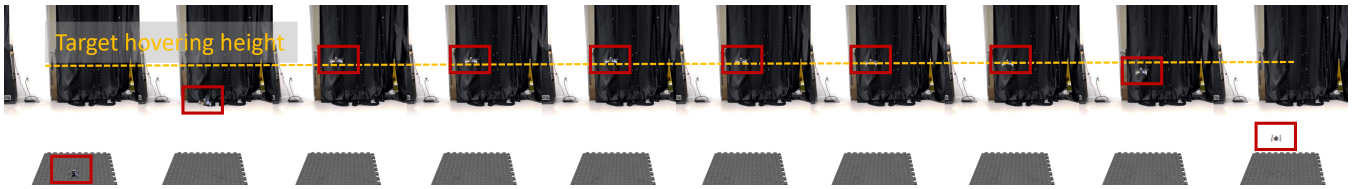
V. Experiments

A. Experimental Setup

We conduct experiments on the quadrotors. Quadrotor is a representative agile and safety-critical dynamical system,



(a) Simulator policy.



(b) Policy improved by **STEADY** after training with 20 taking-off and landing trajectories.

Fig. 2: Snapshots of taking-off, hovering 7 seconds then landing with the simulator policy and policy improved by . Yellow dash lines indicate the target hovering height (1m). The Crazyflies are highlighted with red boxes. The snapshots are taken every 0.8 seconds. Figure 2(a) shows the simulator policy and Figure 2(b) shows the policy learned by the proposed **STEADY** algorithm.

Algorithm 2 STEADY: Skill Transfer and Discovery for Sim-to-Real Learning

Input: Simulator MDP $\mathcal{M}^\circ$, real world MDP $\mathcal{M}$

```
# Simulator stage.
```
$\phi^\circ, \mu^\circ, \pi^\circ, w^{\pi^\circ}$ = **SPEDER**($\mathcal{M}^\circ$) (See Algorithm 1)
```
# Real-world stage.
```
Initialization: $\pi^0 = \pi^\circ, w_1^0 = w^{\pi^\circ}$

for episode $k = 1, 2, \ldots, N$ **do**

  Collect trajectory of (s, a, s') in real world, following current policy π^{k-1}.

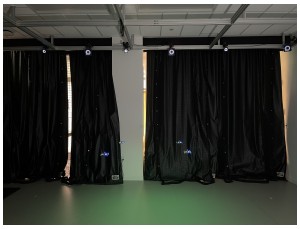
  Skill discovery to learn ϕ, μ by optimizing (10), given simulator skills ϕ°, μ°.

  Policy evaluation by solving Eq. (11).

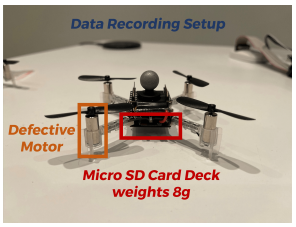
  Policy improvement by doing policy gradient with respect to (12).

end for

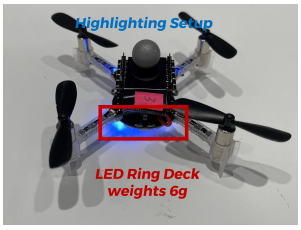
Output: Final policy π^N



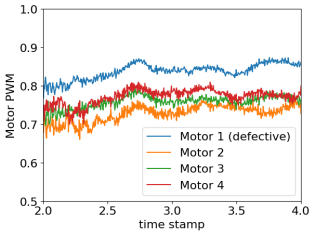
(a) Motion capture system.



(b) Data recording setup.



(c) LED highlight setup.



(d) Motor PWM inputs during hovering.

Fig. 3: The experiment setup.

which is sensitive to controller malfunctions. Controller failure will immediately cause noticeable oscillations or even falling from the air. Moreover, there are multiple sim-to-real gaps in the quadrotor dynamics that are difficult to model, like motor response [5], [29] and aerodynamics including ground effect and downwash [14], [20].

Policy Parameterization. Our transfer learning approaches focus on representing the value function, which supports arbitrary policy parameterization. Although neural-network-based controllers have been implemented on the Crazyflies [5], [6], [29], we found that they are usually unstable and not robust to external disturbance such as observation noise, response delays, motor dynamics, etc. To improve controller stability and make the experiments more reproducible, we use a differentiable version of Mellinger controller [30] as our policy parameterization. THe differentiable version means that the controller parameters are updated through policy gradient.

Simulator and real-word tasks. We first train the policies in the simulator for general trajectory tracking tasks. We fix a goal position and randomly initialize the drone states (position, rotation, velocity, angular and positional velocity, propeller rpm) to track the reference. Then for the downstream real-world transfer stage, we consider three different tasks, (1) taking off, hovering, and landing; (2) following a trajectory in the shape of "8". We will show that the skill sets learned from the general trajectory tracking tasks are transferable to these specific tasks, and the skill discovery will improve the real-world controller performance. For detailed simulator setup please refer to Section C-A in our online report [31].

Baselines and ablation studies. To demonstrate the effectiveness of the proposed controller, we show the comparison between the controller learned by the Algorithm 2 and baselines and ablations. The baseline is the built-in controller in the Crazyflie firmware (labeled as "Built-in"). Ablation studies including simulator policy (labeled as "Simulator") and skill transfer policy (labeled as "Skill

Transfer"). Simulator policy means that we directly use the controller learned in the simulator by Algorithm 1. For the skill transfer policy, we only synthesize policies from the simulator skill sets. The skill discovery in line 7 of Algorithm 2 is skipped, and the Q-function parameterization is the same as the simulator stage.

B. Real-World Robots Setup

We show the effectiveness of our sim-to-real learning on the Crazyflie 2.1 Quadrotor with an STM32F405 microcontroller clocked at 168 MHz. We transfer the learned controller parameters to Crazyflie and handle the communication using Crazyswarm [32]. We use the Optitrack motion capture system with single-marker configuration to provide location information for the Crazyflie, shown in Figure 3(a). The Crazyflie carries an additional micro SD card deck to record data or a LED ring deck to highlight itself shown in Figure 3(b) and 3(c), respectively. Additional explanation of the sim-to-real gap can be found in Appendix C-B in our online report [31].

C. Experimental Results

1) Simulation Stage: The features $\phi^{\circ}, \phi, \mu^{\circ}, \mu$ are parameterized by fully connected neural networks with two hidden layers with 256 neurons. The dimensions of all the representations are 256. For the simulation stage, we train the algorithm with 1.6×10^6 transitions and the expected return during the training process are shown in Figure 4.

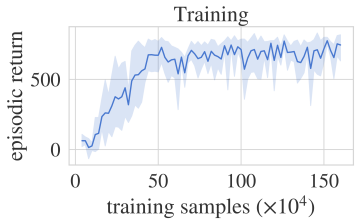


Fig. 4: Episodic return to training samples during the training process with 5 random seeds. The shaded region implies 95% confidential interval over 10 evaluation episodes for every 2000 samples of the five random seeds.

2) Real-world Stage: **Taking off, hovering and landing.** First we show the experimental results on the real-world task of taking off, hovering for 7 seconds at 1m and landing. We compare the 3D trajectories in Figure 5 with skill transfer and discovery policy (labeled as **"STEADY"**), skill transfer policy (labeled as **"Skill Tranfer"**) and simulator policy (labeled as **"Simulator Zero-Shot"**, Zero-Shot means that no learning from real-world data is implemented.). Two sequences of snapshots comparing Simulator policy and the policy improved by STEADY are also shown in Figure 2, and the full video can be found at our project page. Results show that after learning from the real-world data using the STEADY framework, the controller can maintain at the target height very stably without oscillations even with a defective motor and extra weight carried. For the controllers from ablation studies, the simulator policy oscillates heavily and cannot maintain the height. The skill transfer policy stay in the air longer than the simulator policy but still fails to maintain stably at the target position.

Robustness under External Disturbance. We tried learning to hover with external wind to show the robustness and the adaptive capability of the proposed framework, and the video can be found on our project page.

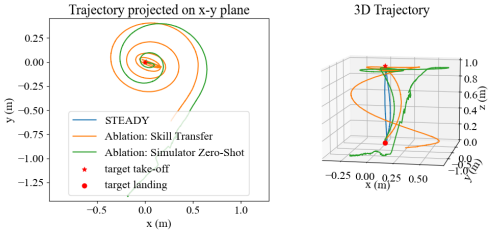


Fig. 5: Trajectories of taking-off, hovering and landing trajectories.

Trajectory tracking. For the trajectory tracking task, the drone needs to follow a trajectory in the shape of the number "8", which is commonly seen in related studies [29], [33]. The trajectories is $(x(t), y(t), z(t)) = (\sin(t), \frac{1}{2}\sin(2t), 1.0)$ and the shape is shown as orange dash lines in Figure 6. Figure 6 shows the trajectory tracking results with different controllers compared to reference trajectories. The performance is compared in Table I. We calculate two metrics for the tracking error, the average position tracking error and the cumulative rewards. The average position tracking error averages $\|\mathbf{p} - \mathbf{p}_{\text{goal}}\|$ over the trajectory. The cumulative rewards sums up rewards defined in (13) over the trajectory (removing the initial constant term of 2). Table I shows that the STEADY controller achieves the smallest trajectory tracking error and highest accumulative rewards. The tracking error is comparable to the built-in controllers and improved by 11.9% and 30.2% compared to ablation skill transfer controller and ablation simulator controller, respectively. For the cumulative reward, the STEADY controller outperforms all three baseline controllers, improving on the ablation skill transfer controller by 9.1%, the Built-in PID controller by 10.9%, and the ablation simulator controller by 22.7%.

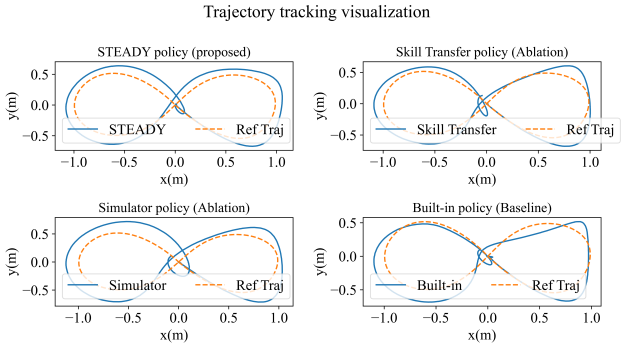


Fig. 6: Trajectory tracking visualization.

TABLE I: Tracking performance comparison.

Controller	Average Tracking Error ($\times10^{-1}$m)	Cumulative Rewards
STEADY	**0.982**	**-1241**
Skill Transfer	1.115	-1365
Simulator	1.406	-1584
Built-in PID	0.983	-1392

VI. Concluding Remarks

In this paper, we proposed the STEADY framework, which utilizes skill transfer and discovery for sim-to-real learning.

Inspired by the concept of representations as skill sets when considering the spectral decompositions of MDPs, we show that we can learn the skill sets from simulators and then transfer them to the real world. The representation-based skill sets can also help sim-to-real transfer. By enforcing orthogonal constraints between the simulator skill sets and the skill sets induced by the sim-to-real gap, we promote the discovery of useful and distinct new skills. Building on the enlarged skill sets comprising these new skill sets and the existing simulator skill sets, STEADY facilitates more efficient and effective sim-to-real transfer smoothly.

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.

[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 *Proceedings of the 35th International Conference on Machine Learning*. PMLR, Jul. 2018, pp. 1861–1870.

[3] O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, Jan. 2020, publisher: SAGE Publications Ltd STM.

[4] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, Aug. 2023, number: 7976 Publisher: Nature Publishing Group.

[5] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 59–66.

[6] S. Gronauer, D. Stümke, and K. Diepold, "Comparing Quadrotor Control Policies for Zero-Shot Reinforcement Learning under Uncertainty and Partial Observability," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Detroit, MI, USA: IEEE, Oct. 2023, pp. 7508–7514.

[7] C. Jin, Z. Yang, Z. Wang, and M. I. Jordan, "Provably efficient reinforcement learning with linear function approximation," in *Conference on learning theory*. PMLR, 2020, pp. 2137–2143.

[8] L. Yang and M. Wang, "Reinforcement learning in feature space: Matrix bandit, kernels, and regret bound," in *International Conference on Machine Learning*. PMLR, 2020, pp. 10 746–10 756.

[9] A. Agarwal, S. Kakade, A. Krishnamurthy, and W. Sun, "Flambe: Structural complexity and representation learning of low rank mdps," *Advances in neural information processing systems*, vol. 33, pp. 20 095–20 107, 2020.

[10] T. Ren, T. Zhang, L. Lee, J. E. Gonzalez, D. Schuurmans, and B. Dai, "Spectral decomposition representation for reinforcement learning," in *The Eleventh International Conference on Learning Representations*, 2023.

[11] T. Zhang, T. Ren, M. Yang, J. Gonzalez, D. Schuurmans, and B. Dai, "Making linear mdps practical via contrastive representation learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 26 447–26 466.

[12] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Data-efficient control policy search using residual dynamics learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 4709–4715.

[13] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual Reinforcement Learning for Robot Control," in *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada: IEEE, May 2019, pp. 6023–6029.

[14] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural Lander: Stable Drone Landing Control Using Learned Dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 9784–9790.

[15] T. Ren, Z. Ren, N. Li, and B. Dai, "Stochastic nonlinear control via finite-dimensional spectral dynamic embedding," in *2023 62nd IEEE Conference on Decision and Control (CDC)*. IEEE, 2023, pp. 795–800.

[16] 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)*, Sep. 2017, pp. 23–30.

[17] 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)*, May 2019, pp. 8973–8979.

[18] 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)*, May 2018, pp. 3803–3810.

[19] S. Levine and P. Abbeel, "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics," in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.

[20] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-Swarm: Decentralized Close-Proximity Multirotor Control Using Learned Interactions," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 3241–3247, iSSN: 2577-087X.

[21] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, Jul. 2019, conference Name: IEEE Transactions on Automatic Control.

[22] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, "Auto-Tuned Sim-to-Real Transfer," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, May 2021, pp. 1290–1296.

[23] A. Tanwani, "DIRL: Domain-Invariant Representation Learning for Sim-to-Real Transfer," in *Proceedings of the 2020 Conference on Robot Learning*. PMLR, Oct. 2021, pp. 1558–1571.

[24] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 2169–2176.

[25] M. K. Helwa and A. P. Schoellig, "Multi-robot transfer learning: A dynamical system perspective," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 4702–4708.

[26] A. Agarwal, Y. Song, W. Sun, K. Wang, M. Wang, and X. Zhang, "Provable Benefits of Representational Transfer in Reinforcement Learning," in *Proceedings of Thirty Sixth Conference on Learning Theory*. PMLR, Jul. 2023, pp. 2114–2187.

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

[28] T. Ren, C. Xiao, T. Zhang, N. Li, Z. Wang, sujay sanghavi, D. Schuurmans, and B. Dai, "Latent variable representation for reinforcement learning," in *The Eleventh International Conference on Learning Representations*, 2023.

[29] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," 2023.

[30] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, May 2011, pp. 2520–2525.

[31] H. Ma, Z. Ren, B. Dai, and N. Li, "Skill transfer and discovery for sim-to-real learning: A representation-based viewpoint." [Online]. Available: https://scholar.harvard.edu/haitongma/files/transfer_learning_online_report.pdf

[32] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3299–3304.

[33] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural-fly enables rapid learning for agile flight in strong winds," *Science Robotics*, vol. 7, no. 66, p. eabm6597, 2022.