Robotics: Science and Systems 2026
Sydney, Australia, July 13–17, 2026

Zero-Shot Sim-to-Real Robot Learning: A Dexterous Manipulation Study on Reactive Catching

Kejia Ren[1], Gaotian Wang[1], Andrew S. Morgan[2] and Kaiyu Hang[1]
[1]Department of Computer Science, Rice University
[2]Robotics and AI Institute

***Abstract*—Dexterous manipulation is physics-intensive and highly sensitive to modeling errors and perception noise, making sim-to-real transfer prohibitively challenging. Domain randomization (DR) is commonly used to improve the robustness of learned policies for such tasks, but conventional DR randomizes one instance per episode, offering very limited exposure to the variability of real-world dynamics. To this end, we propose Domain-Randomized Instance Set (DRIS), which represents and propagates a set of randomized instances simultaneously, providing richer approximation of uncertain dynamics and enabling policies to learn actions that account for multiple possible outcomes. Supported by theoretical analysis, we show that DRIS yields more robust policies and alleviates the need for real-world fine-tuning, even with a modest number of instances (e.g., 10). We demonstrate this on a challenging reactive catching task. Unlike traditional catching setups that use end-effectors designed to mechanically stabilize the object (e.g., curved or enclosing surfaces), our system uses a flat plate that offers no passive stabilization, making the task highly sensitive to noise and requiring rapid reactive motions. The learned policies exhibit strong robustness to uncertainties and achieve reliable zero-shot sim-to-real transfer.**

I. Introduction

Dexterous manipulation refers to the robot's ability to skillfully manipulate objects through coordinated and intentional contact interactions, e.g., in-hand manipulation [5, 11], tool use [26], pushing [14], catching [25], etc. Such tasks often require well-planned contact, timing, and motion to achieve stable and adaptive control over object behavior. Moreover, they often introduce a substantial sim-to-real gap, as the success of such tasks can be highly sensitive to inaccuracies in estimating the underlying dynamics and physical properties such as contact geometry, frictions, object inertia, and compliance. Even small discrepancies between simulation and the real world can lead to large deviations in manipulation behavior and ultimately cause task failure.

Traditional model-based frameworks depend on known system parameters to achieve accurate prediction and control [9]. However, in practical manipulation scenarios, properties such as contact stiffness, friction coefficients, and object geometries can greatly vary between environments and are difficult to precisely identify. Learning-based methods address these limitations by training policies with extensive domain randomization or data augmentation over uncertain physical parameters in simulation [33], improving the robustness to these uncertainties when deployed in the real world. Yet, tasks involving very dynamic, contact-rich interactions remain difficult to learn, as their success hinges on precise coordination of contact timing and force modulation under uncertainty.

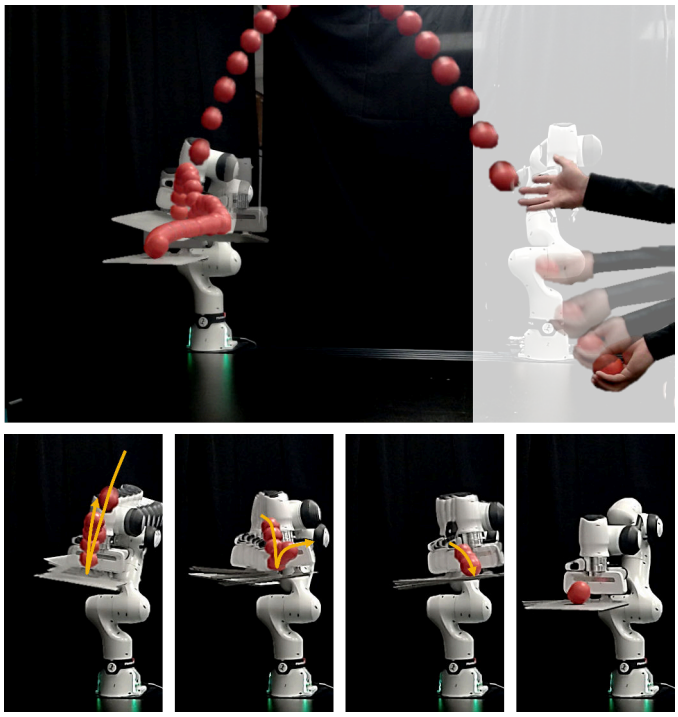


Fig. 1: Robot reactive catching of a rubber ball using a flat plate. The second row shows sequential impacts during the catching motion, where the final frame depicts the ball successfully stabilized on the plate; the yellow arrows indicate the ball's trajectory at the moments of impact.

Domain randomization is traditionally performed by perturbing relevant physical parameters during data collection, e.g., sampling one object instance with randomized properties in each rollout and training policy on the collected rollouts. While this exposes the policy to diverse experiences, it lacks a structured mechanism for the policy to reason about how uncertainties affect possible manipulation outcomes. In contrast, if such uncertainties are explicitly incorporated into the task representation itself, the policy can learn to generate actions that compensate for them and plan with respect to a distribution of possible outcomes, thereby mitigating the sim-to-real gap introduced by these uncertainties.

To this end, rather than training a policy on individual

manipulation instances with randomly sampled physical parameters (as is done in conventional domain randomization), we propose *Domain-Randomized Instance Set (DRIS)* that explicitly models a distribution of possible physical variations encountered during manipulation. Our proposed DRIS contains multiple parallel instances (e.g., objects with different physical properties), all of which are propagated simultaneously. Their state evolutions under a shared action jointly inform the policy update. As a result, the learned policy becomes inherently tolerant to the uncertainties encoded in DRIS, leading to improved robustness against the sim-to-real gap (e.g., enabling reliable zero-shot transfer).

To demonstrate the effectiveness of the proposed learning strategy, we consider a highly challenging reactive catching task illustrated in Fig. 1. Unlike traditional catching setups that employ curved or concave end-effectors such as cups [15], nets [17], or articulated hands [7], to mechanically stabilize the object after impact, our setup uses a flat and low-friction plate rigidly attached to the robot's end-effector. As a result, the system becomes highly sensitive to physical uncertainties or perception errors, where even small errors in contact timing, plate orientation, or unwanted lateral forces can cause the ball to bounce off or roll away from the plate.

In this work, we propose a robust learning strategy for dynamic manipulation, demonstrated through a challenging robot reactive catching case study. Our key technical contributions are summarized as follows:

1) We propose Domain-Randomized Instance Set (DRIS) as a representation to capture the uncertainty in system dynamics induced by physical parameter variations;
2) We propose a DRIS-based data collection and policy learning paradigm that enables simultaneous handling of multiple task instances, yielding more stable and robust policy learning and improved zero-shot sim-to-real transfer than the traditional paradigm, supported by theoretical analysis;
3) We instantiate the framework on a challenging reactive catching problem with a tailored task representation and policy architecture, and validate our framework through both simulation and real-world experiments.

II. Related Work

Reinforcement Learning for Dexterous Manipulation. Unlike traditional analysis-based approaches sensitive to modeling errors, Reinforcement Learning (RL) learns directly from data, making it advantageous for the complex contact dynamics of dexterous manipulation. RL has successfully solved tasks including in-hand manipulation [5, 11], catching [1], juggling [36], deformable object manipulation [12], and regrasping [35]. However, these data-intensive methods typically require large-scale high-fidelity simulation or expert demonstrations to facilitate real-world transfer.

Sim-to-Real Transfer in Robotics. Bridging the sim-to-real gap caused by sensing and contact discrepancies is a central challenge. Beyond Domain Randomization (DR), researchers utilize high-fidelity simulators [51, 41, 27] or explicitly bridge the gap via adaptive system identification [54, 23, 45] and hybrid dynamics learning [3, 24]. While these techniques enable zero-shot transfer [49], reliable deployment for dynamic manipulation remains difficult due to the discontinuous, stochastic nature of high-speed contact dynamics.

Domain Randomization in Robot Learning. Since real-world data collection is costly, DR is essential for reducing the simulation gap in contact-rich tasks [30, 21]. Randomization strategies have evolved from visual and geometric properties [49, 50] to kinematic [16] and dynamic parameters [33]. While active DR optimizes sampling distributions to improve transfer [39, 10, 28, 29, 47], it typically requires real-world rollouts. Although some established works strategically employ DR—either through curriculum-based scaling [4] or entropy maximization [48]—to enable zero-shot ransfer, our approach differs by training over the joint evolution of multiple randomized instances simultaneously to achieve zero-shot transfer for challenging tasks such as catching.

Representing Uncertainty in States and Dynamics. Robotics has extensively studied uncertainty representation. Classical methods employ belief-space planning via POMDPs [37, 8, 22]. Learning-based approaches incorporate uncertainty through probabilistic latent dynamics [18, 19], history-based parameter inference [31, 44], or ensembles for model-based RL [13]. While some history-based approaches require real-world data in the loop [44], our approach is purely zero-shot. Furthermore, while others can perform adaptation online [31], they are not suitable enough for highly dynamic tasks such as catching, where the robot must react within milliseconds. Complementary to these, recent work learns belief embeddings to explicitly encode uncertainty for policy conditioning in partially observable settings [52]. While such belief-representation methods learn a latent state space to implicitly represent uncertainty, our approach more explicitly and intuitively captures the uncertainty through a multi-instance representation during high-speed contacts.

Dexterous Robot Catching. Catching typically requires fast prediction and reactive motion. Prior work evolved from optimization-based planning [7] and probabilistic models for uneven objects [25] to deep learning-based policies on mobile manipulators [55]. However, most rely on mechanically assisting end-effectors like cups [32, 15, 1], nets [17], or articulated hands [40, 20]. In contrast, we use a flat plate without mechanical stabilization. Unlike similar flat-surface work [6] limited to specific objects and a number of rebounds, our approach handles diverse conditions without such restrictions.

III. Learning Manipulation Policy with Domain-Randomized Instance Set

A. Dexterous Manipulation as Policy Learning

We consider the dexterous manipulation problem in which a robotic manipulator interacts with an object through contact-rich dynamics. We define the domain space $\mathcal{C}$ as the set of all relevant physical parameters that influence the system dynamics, such as robot characteristics (e.g., stiffness), object geometry and physical properties (e.g., friction). Each $\boldsymbol{c} \in \mathcal{C}$

represents a specific domain instance that induces a particular realization of the system dynamics. These parameters are often difficult to measure accurately, but they can substantially affect the object's motion during manipulation.

The problem is formulated as a discrete-time Markov Decision Process (MDP). At time step t, the system observes a state $\boldsymbol{s}_t \in \mathcal{S}$, executes an action $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t) \in \mathcal{A}$ according to a robot policy $\pi(\cdot)$, and transitions to the next state $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{c})$, where the dynamics f depend on the domain parameter $\boldsymbol{c}$. Different values of $\boldsymbol{c}$ (e.g., object properties) lead to distinct motion behaviors even under the same robot action. At each step, the system receives a scalar reward $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) \in \mathbb{R}$.

The objective is to find an optimal policy $\pi_{\boldsymbol{\theta}} : \mathcal{S} \mapsto \mathcal{A}$, that maximizes the expected return (i.e., cumulative reward), where $\gamma \in [0, 1)$ is a discount factor:

$$\begin{aligned} \max_{\pi_{\boldsymbol{\theta}}} \quad & \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T-1} \gamma^t\, r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right] \\ \text{s.t.} \quad & \boldsymbol{a}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t) \in \mathcal{A}, \quad \forall t = 0, \cdots, T-1, \\ & \boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{c}) \in \mathcal{S}, \quad \boldsymbol{c} \in \mathcal{C} \end{aligned} \tag{1}$$

In deep reinforcement learning (DRL), the policy $\pi_{\boldsymbol{\theta}}$ is modeled by a neural network parameterized by $\boldsymbol{\theta}$, and optimized via gradient-based update using data collected from robot-object interactions.

B. Domain-Randomized Instance Set (DRIS)

In practice, physical parameters $\boldsymbol{c}$ are difficult to identify accurately, introducing uncertainty into manipulation dynamics. To improve robustness, most approaches model the dynamics stochastically as a probability $P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and apply domain randomization to approximately capture this stochasticity, typically by sampling a single parameter instance per episode in standard model-free RL [33]. More recent methods further adapt the sampling distribution using real-world data [39, 47]. However, these strategies still predict only individual next states, and do not explicitly model how the distribution of possible states evolves over time.

Inspired by prior work on set-based representations [53], rather than propagating a single state $\boldsymbol{s}_t$, we instead propagate a set of possible states $\mathcal{S}_t \subset \mathcal{S}$ simultaneously under a shared robot action, reflecting the effects of physical variations across different domain parameters. Specifically, we construct a discrete set $\hat{\mathcal{C}} \subset \mathcal{C}$ of N manipulation instances (indexed by i) by uniformly sampling from the domain space:

$$\hat{\mathcal{C}} = \left\{\boldsymbol{c}^{(i)}\right\}_{i=1}^{N}, \ \boldsymbol{c}^{(i)} \sim \mathcal{U}(\mathcal{C}) \tag{2}$$

We then define the *Domain-Randomized Instance Set (DRIS)* as the collection of state-parameter pairs associated with these sampled instances:

$$\mathcal{D}_t = \left\{\left(\boldsymbol{s}_t^{(i)}, \boldsymbol{c}^{(i)}\right) \,\middle|\, \boldsymbol{c}^{(i)} \in \hat{\mathcal{C}}\right\}_{i=1}^{N} \subset \mathcal{S} \times \mathcal{C} \tag{3}$$

where $\boldsymbol{s}_t^{(i)}$ in each element denotes the state of the i-th manipulation instance, as it evolves at time t under its corresponding domain parameter $\boldsymbol{c}^{(i)}$. For clarity, we also extract the state of all instances in the DRIS to form another set (i.e., the projection of $\mathcal{D}_t$ onto the state space $\mathcal{S}$):

$$\mathcal{S}_t = \text{proj}_{\mathcal{S}}(\mathcal{D}_t) = \left\{\boldsymbol{s}_t^{(i)} \,\middle|\, \left(\boldsymbol{s}_t^{(i)}, \boldsymbol{c}^{(i)}\right) \in \mathcal{D}_t\right\} \subset \mathcal{S} \tag{4}$$

We extend the system dynamics by introducing a new function $\mathcal{F}$ to evolve the entire instance set:

$$\begin{aligned} \mathcal{D}_{t+1} &= \left\{\left(\boldsymbol{s}_{t+1}^{(i)}, \boldsymbol{c}^{(i)}\right)\right\}_{i=1}^{N} = \mathcal{F}(\mathcal{D}_t, \boldsymbol{a}_t) \\ &= \left\{\left(f\left(\boldsymbol{s}_t^{(i)}, \boldsymbol{a}_t, \boldsymbol{c}^{(i)}\right), \boldsymbol{c}^{(i)}\right) \,\middle|\, \left(\boldsymbol{s}_t^{(i)}, \boldsymbol{c}^{(i)}\right) \in \mathcal{D}_t\right\} \end{aligned} \tag{5}$$

C. Zero-Shot Sim-to-Real Learning

Since DRIS defined in Sec. III-B inherently embeds uncertainties into its representation, integrating DRIS into policy learning in simulation enables more robust sim-to-real transfer without fine-tuning. Because the DRIS size N may vary and because the robot will manipulate only a single instance (although trained with multiple instances in DRIS) in real deployment, the policy must take an input whose dimensionality is independent of N. Thereafter, we require a compact and size-agnostic representation of DRIS. To this end, we employ an encoder $\psi(\cdot)$ that maps the DRIS state (i.e., $\mathcal{S}_t$) to a fixed-dimensional latent vector $\boldsymbol{z}_t = \psi(\mathcal{S}_t) \in \mathbb{R}^{d_z}$, where d_z is the latent dimension and remains constant regardless of the size N of $\mathcal{D}_t$. A policy that directly operates in this latent space is then learned by solving:

$$\begin{aligned} \max_{\pi_{\boldsymbol{\theta}}} \quad & \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T-1} \gamma^t \cdot \frac{1}{N} \sum_{\boldsymbol{s}_t^{(i)} \in \mathcal{S}_t} r(\boldsymbol{s}_t^{(i)}, \boldsymbol{a}_t)\right] \\ \text{s.t.} \quad & \boldsymbol{a}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{z}_t) \in \mathcal{A}, \quad \forall t = 0, \cdots, T-1, \\ & \boldsymbol{z}_t = \psi(\mathcal{S}_t) \in \mathbb{R}^{d_z}, \quad \forall t = 0, \cdots, T-1, \\ & \mathcal{S}_t = \text{proj}_{\mathcal{S}}(\mathcal{D}_t), \quad \forall t = 0, \cdots, T-1, \\ & \mathcal{D}_{t+1} = \mathcal{F}(\mathcal{D}_t, \boldsymbol{a}_t) \subset \mathcal{S} \times \mathcal{C} \end{aligned} \tag{6}$$

Compared to conventional DR, we summarize the benefits of DRIS in Theorem III.1:

Theorem III.1 (DRIS improves stability and robustness)**.** *Relative to conventional domain randomization, DRIS is an exact particle approximation for the system belief propagation. Consequently, DRIS yields a more stable optimization, more robust policies, and improved sim-to-real generalization. (See Appendix B for detailed analysis and proofs.)*

As illustrated in Fig. 2 (left) with a catching task, DRIS can typically be implemented by a set of multiple object instances, each initialized with different physical properties (domain parameters) such as shape, weight, and friction. The instances evolve independently, i.e., they do not interact with one another, and their respective state transitions under the same robot action are evaluated in parallel for policy update.

We summarize the DRIS-based policy learning pipeline (using an on-policy example) and its zero-shot inference in Alg. 1. At each training iteration in simulation, we randomly

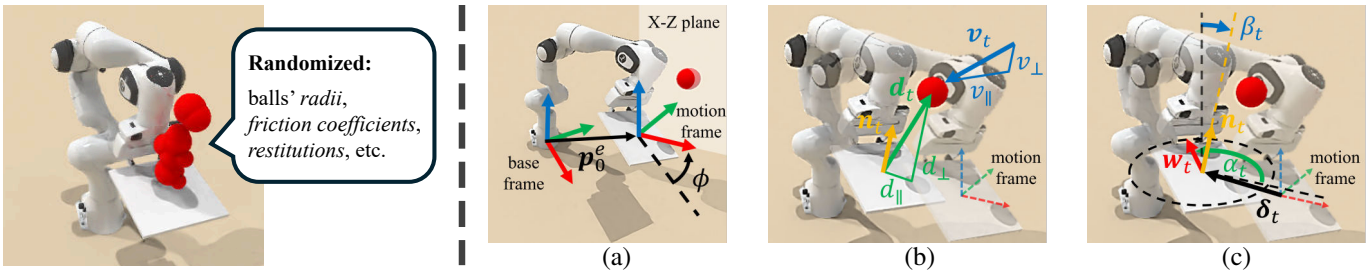


Fig. 2: The reactive catching problem instantiation. *Left:* Randomized parameters in DRIS. *Right:* Components used to represent the problem: (a) The motion frame is defined at the plate's initial position $\boldsymbol{p}_0^e$ and rotated about the Z-axis by angle ϕ so that its X-Z plane passes through the incoming ball. (b) The robot has moved from the initial ghosted to the current opaque configuration. The system state consists of the ball's relative displacement $\boldsymbol{d}_t$ (green arrow) to the plate and its linear velocity $\boldsymbol{v}_t$ (blue arrow), both expressed in the motion frame; these two quantities are decomposed with respect to the plate's normal vector $\boldsymbol{n}_t$ (yellow arrow) to define the reward. (c) The action comprises a translation of the plate $\boldsymbol{\delta}_t$ (black arrow) and a tilting configuration: angle α_t (green) specifies a horizontal axis $\boldsymbol{w}_t$ (red arrow), about which the plate is rotated by angle β_t (blue).

Algorithm 1 DRIS-based Policy Learning and Inference

Input: Domain parameter space $\mathcal{C}$, initial policy $\pi_{\boldsymbol{\theta}}$, DRIS encoder ψ, dynamics $\mathcal{F}$ modeled by a simulator

Part (a): Policy Learning in Simulation

while training **do**
  $\boldsymbol{s}_0 \leftarrow \textsc{InitState}()$ ▷ Random Initialization
  $\mathcal{D}_0 \leftarrow \{\}$ ▷ Initialize DRIS
  for $i = 1, \cdots, N$ **do**
    $\boldsymbol{c}^{(i)} \sim \mathcal{U}(\mathcal{C})$, $\boldsymbol{s}_0^{(i)} \leftarrow \boldsymbol{s}_0$
    $\mathcal{D}_0 \leftarrow \mathcal{D}_0 \cup \left\{\left(\boldsymbol{s}_0^{(i)}, \boldsymbol{c}^{(i)}\right)\right\}$
  end for
  $\mathcal{T} \leftarrow \{\}$ ▷ Simulation Rollout
  for $t = 0, \cdots, T-1$ **do**
    $\mathcal{S}_t \leftarrow \text{proj}_{\mathcal{S}}(\mathcal{D}_t)$ ▷ DRIS State
    $\boldsymbol{z}_t \leftarrow \psi(\mathcal{S}_t)$, $\boldsymbol{a}_t \leftarrow \pi_{\boldsymbol{\theta}}(\boldsymbol{z}_t)$
    $r_t \leftarrow \frac{1}{N}\sum_{\boldsymbol{s}\in\mathcal{S}_t} r(\boldsymbol{s}, \boldsymbol{a}_t)$ ▷ Average Reward
    $\mathcal{D}_{t+1} \leftarrow \mathcal{F}(\mathcal{D}_t, \boldsymbol{a}_t)$ ▷ Propagate by Eq. (5)
    $\mathcal{T} \leftarrow \mathcal{T} \cup (\boldsymbol{z}_t, \boldsymbol{a}_t, r_t)$
  end for
  $\pi_{\boldsymbol{\theta}} \leftarrow \textsc{PolicyUpdate}(\pi_{\boldsymbol{\theta}}, \mathcal{T})$ ▷ Policy Update
end while

Part (b): Zero-Shot Inference in Real World

while not finished **do**
  $\boldsymbol{s} \leftarrow \textsc{ObserveState}()$ ▷ Real-World State
  $\boldsymbol{z} \leftarrow \psi(\{\boldsymbol{s}\})$, $\boldsymbol{a} \leftarrow \pi_{\boldsymbol{\theta}}(\boldsymbol{z})$ ▷ Generate Action
  finished $\leftarrow \textsc{Execute}(\boldsymbol{a})$
end while

initialize the system state $\boldsymbol{s}_0$ and sample N different domain parameters from $\mathcal{C}$ to construct the DRIS, with all instances initialized at $\boldsymbol{s}_0$. We then collect on-policy rollouts to update the policy, where the rollout stores the encoded latent state $\boldsymbol{z}_t$ rather than the full set of instance states $\left\{\boldsymbol{s}_t^{(i)}\right\}_{i=1}^{N}$. After training, the policy can be deployed in the real world without fine-tuning: At each time step, the observed single real-world state $\boldsymbol{s}$ (equivalent to a DRIS of size $N = 1$) is encoded and provided as input to the trained policy to infer the next action, repeatedly until the task is finished.

IV. Reactive Catching Case Study

We instantiate the proposed DRIS-based learning strategy on the problem of reactive ball catching with an M-DoF robot manipulator. As illustrated in Fig. 1, the robot is equipped with a flat plate rigidly attached to its end-effector and is required to catch a ball thrown toward it. To successfully perform this task, the robot must approach and make contact with the incoming ball, absorb its kinetic energy through controlled motions, and subsequently stabilize and balance the ball once it comes to rest on the plate.

A. Problem Representation

To enable a more compact problem representation and ensure the catching policy is invariant to the ball's incoming direction, we define a fixed motion frame relative to the ball at $t = 0$ (the beginning of manipulation). All task-related quantities (e.g., state and action) are expressed in this motion frame. As shown in Fig. 2 (right-a), the motion frame is constructed with its origin at the plate's center, its Z-axis aligned vertically upward, and its X-Z plane passing through the ball's center. The transformation from the robot's base frame to the motion frame is denoted by ${}^b\boldsymbol{T}_m = (\boldsymbol{R}_z(\phi), \boldsymbol{p}_0^e) \in SE(3)$, where $\boldsymbol{R}_z(\phi) \in SO(3)$ denotes a rotation about the Z-axis by angle ϕ and $\boldsymbol{p}_0^e \in \mathbb{R}^3$ represents the plate's center position at $t = 0$, expressed in the robot's base frame.

State. The state of the system consists of the ball's displacement relative to the plate's center and its linear velocity, both expressed in the motion frame. Specifically, as shown in Fig. 2 (right-b), the state is denoted as $\boldsymbol{s}_t = (\boldsymbol{d}_t, \boldsymbol{v}_t) \in \mathbb{R}^6$, where $\boldsymbol{d}_t = \boldsymbol{R}_z(\phi)^\top(\boldsymbol{p}_t^o - \boldsymbol{p}_t^e) \in \mathbb{R}^3$ is the displacement of the ball (with position $\boldsymbol{p}_t^o$) relative to the plate's center

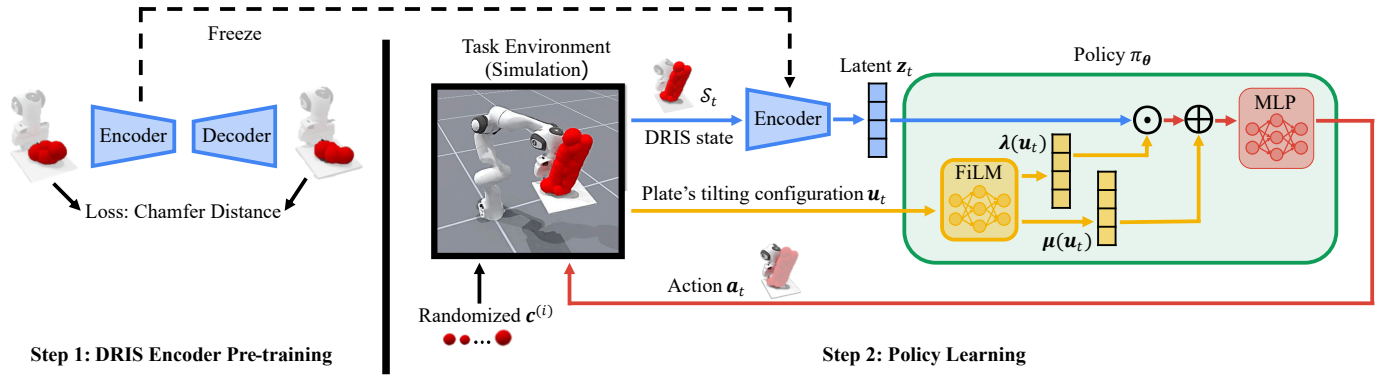


Fig. 3: Schematic overview of the DRIS-based learning pipeline. *Step 1 (Left):* The DRIS encoder is pre-trained to reconstruct the input DRIS state. *Step 2 (Right):* The policy network is trained via RL using the pre-trained (and frozen) DRIS encoder. At the beginning of each simulation episode, the physical property $\boldsymbol{c}^{(i)}$ of each ball instance in the DRIS is randomized; at each time step, the DRIS state $\mathcal{S}_t$ is encoded and fed to the policy $\pi_{\boldsymbol{\theta}}$ to generate an action $\boldsymbol{a}_t$.

(with position $\boldsymbol{p}_t^e$) at time t, expressed in the motion frame; $\boldsymbol{v}_t = \boldsymbol{R}_z(\phi)^\top \boldsymbol{v}_t^o \in \mathbb{R}^3$ is the ball's linear velocity $\boldsymbol{v}_t^o$ transformed into the motion frame.

Action. The action is defined as $\boldsymbol{a}_t = (\boldsymbol{\delta}_t, \boldsymbol{u}_t) \in \mathbb{R}^5$, where $\boldsymbol{\delta}_t \in \mathbb{R}^3$ denotes the displacement command of the plate's center expressed in the motion frame, i.e., the corresponding target position of the plate's center in the robot's base frame is computed as $\tilde{\boldsymbol{p}}_t^e = \boldsymbol{R}_z(\phi)\,\boldsymbol{\delta}_t + \boldsymbol{p}_t^e$. The remaining action component $\boldsymbol{u}_t = (\alpha_t, \beta_t) \in \mathbb{R}^2$ represents the target tilting configuration of the plate in 3D. Specifically, the plate is rotated about a horizontal axis $\boldsymbol{w}_t = (\cos\alpha_t, \sin\alpha_t, 0)^\top$ by an angle β_t, where $\alpha_t \in [0, 2\pi)$ and $\beta_t \in [0, \pi/4)$. This rotation tilts the plate's normal vector $\boldsymbol{n}_t \in \mathbb{R}^3$ as illustrated in Fig. 2 (right-c). During execution, the complete action $\boldsymbol{a}_t$ is converted into a desired plate pose, which is then used to command the robot via joint torque control, as will be detailed in Sec. IV-D.

Reward. Given the observed state $\boldsymbol{s}_t = (\boldsymbol{d}_t, \boldsymbol{v}_t)$ and the plate normal $\boldsymbol{n}_t \in \mathbb{R}^3$ with $\|\boldsymbol{n}_t\| = 1$ (all in the motion frame), we decompose the ball's displacement and velocity into components perpendicular and parallel to the plate surface $d_\perp, d_\parallel, v_\perp, v_\parallel \in \mathbb{R}$, as in Fig. 2 (right-b):

$$\begin{aligned} d_\perp &= \boldsymbol{d}_t \cdot \boldsymbol{n}_t, \quad d_\parallel = \|\boldsymbol{d}_t - d_\perp \cdot \boldsymbol{n}_t\| \\ v_\perp &= \boldsymbol{v}_t \cdot \boldsymbol{n}_t, \quad v_\parallel = \|\boldsymbol{v}_t - v_\perp \cdot \boldsymbol{n}_t\| \end{aligned} \tag{7}$$

The reward is the sum of a velocity term $r_v \in (0, 1]$ (which yields a higher value when the ball's velocity is kept lower) and a constant penalty term $r_p = -1$ applied when the ball moves beneath or outside the plate:

$$\begin{aligned} r_t &= r(\boldsymbol{s}_t, \boldsymbol{a}_t) = r_v + r_p, \\ r_v &= \frac{1}{2}\exp\left(-\frac{v_\parallel^2}{\eta^2}\right) + \frac{1}{2}\exp\left(-\frac{\max\{v_\perp, -0.1\}^2}{\eta^2}\right), \\ r_p &= -\mathbb{1}\{d_\perp < 0 \vee d_\parallel > l_e\} \end{aligned} \tag{8}$$

where $\eta \in \mathbb{R}$ in the r_v term is a decay coefficient controlling the sensitivity to ball velocity, and $l_e \in \mathbb{R}$ in r_p term denotes the plate's size (e.g., half length).

B. Training with DRIS

For the reactive catching problem, we consider four physical properties as the domain parameters: the ball's radius, static friction, dynamic friction, and restitution coefficient, i.e., $\mathcal{C} \subset \mathbb{R}^4$. As shown in Fig. 3, in each episode, we spawn N balls whose physical properties are independently sampled from $\mathcal{C}$ to construct the DRIS. In a vectorized representation, the DRIS state is formed by concatenating the states (i.e., relative displacements and velocities) of these ball instances, i.e., $\mathcal{S}_t = \left(\boldsymbol{s}_t^{(1)}, \cdots, \boldsymbol{s}_t^{(N)}\right) \in \mathbb{R}^{N\times 6}$.

As described in Sec. III-C, an encoder $\psi(\cdot)$ is required to transform the DRIS state into a latent vector in a compact feature space. The encoder must be size-agnostic to the DRIS size N, mapping any set to a fixed-dimensional latent feature $\boldsymbol{z}_t \in \mathbb{R}^{d_z}$ where d_z denotes the dimension of the latent feature space. We adopt a point cloud-based Autoencoder (AE) [2] and extend its input dimensionality from 3D to 6D by incorporating both the positions and velocities of balls. The point cloud AE consists of an encoder built from 1D convolutional layers followed by a feature-wise max-pooling layer to extract features of the input set [38], and an MLP-based decoder that reconstructs the original set from the latent feature. We collect a dataset of DRIS samples by executing random robot actions, and pre-train this AE to reconstruct DRIS state using the Chamfer distance as reconstruction loss:

$$\mathcal{L}_\psi(\mathcal{S}_t, \widetilde{\mathcal{S}}) = \frac{1}{N}\sum_{\boldsymbol{s}\in\mathcal{S}_t}\min_{\boldsymbol{s}'\in\widetilde{\mathcal{S}}}\|\boldsymbol{s}-\boldsymbol{s}'\|^2 + \frac{1}{|\widetilde{\mathcal{S}}|}\sum_{\boldsymbol{s}'\in\widetilde{\mathcal{S}}}\min_{\boldsymbol{s}\in\mathcal{S}_t}\|\boldsymbol{s}-\boldsymbol{s}'\|^2 \tag{9}$$

where $\mathcal{S}_t$ and $\widetilde{\mathcal{S}}$ denote the input and reconstructed DRIS state, respectively. The encoder from the pre-trained AE is then used as $\psi(\cdot)$ for downstream policy learning.

With RL, at each training step, the pre-trained encoder ψ transforms the observed DRIS state $\mathcal{S}_t$ into a latent vector $\boldsymbol{z}_t$; the policy, operating in the latent feature space (as detailed in Sec. IV-C), is trained using a policy gradient algorithm such as Proximal Policy Optimization (PPO) [43].

C. FiLM-Conditioned Policy Network

The policy $\pi_{\boldsymbol{\theta}}$ consists of two components: a Feature-wise Linear Modulation (FiLM) [34] module and an MLP-based action generation network, as illustrated in Fig. 3.

At each step t, given the encoded state feature $\boldsymbol{z}_t$ and the observed plate tilting orientation $\boldsymbol{u}_t$ (which serves as the conditional signal), the FiLM module applies a feature-wise affine transformation:

$$\tilde{\boldsymbol{z}}_t = \text{FiLM}\left(\boldsymbol{z}_t,\, \boldsymbol{u}_t\right) = \boldsymbol{\lambda}\left(\boldsymbol{u}_t\right) \odot \boldsymbol{z}_t + \boldsymbol{\mu}\left(\boldsymbol{u}_t\right) \tag{10}$$

where $\boldsymbol{\lambda}\left(\boldsymbol{u}_t\right)$, $\boldsymbol{\mu}\left(\boldsymbol{u}_t\right) \in \mathbb{R}^{d_z}$ are outputs of two small neural networks that generate scaling and bias coefficients, and $\odot$ denotes element-wise multiplication. The FiLM-modulated feature $\tilde{\boldsymbol{z}}_t \in \mathbb{R}^{d_z}$ is then passed through an MLP-based action-generation network to produce the action $\boldsymbol{a}_t$:

$$\boldsymbol{a}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{z}_t) = \text{MLP}\left(\tilde{\boldsymbol{z}}_t\right) = \text{MLP}\left(\text{FiLM}\left(\boldsymbol{z}_t,\, \boldsymbol{u}_t\right)\right) \tag{11}$$

The policy parameters $\boldsymbol{\theta}$ include the learnable weights of the FiLM modulation networks $\boldsymbol{\lambda}(\cdot)$ and $\boldsymbol{\mu}(\cdot)$ as well as those of the action-generation MLP. During training, all parameters are jointly optimized through gradient-based updates.

The use of FiLM conditioning is motivated by the contact physics of the catching task. When the ball rests on the plate, the plate's orientation (tilting angle) directly alters the tangential component of gravity along the plate surface and the contact direction, thereby affecting the ball's acceleration. By modulating the latent feature with the plate's orientation, the policy can dynamically adapt its action generation to account for these gravity-induced variations, enabling more physically consistent control across different tilt configurations.

D. Control Implementation on High-DoF Manipulator

As described in Sec. IV-A, the action $\boldsymbol{a}_t$ consists of a desired plate displacement $\boldsymbol{\delta}_t$ and a tilting configuration $\boldsymbol{u}_t = (\alpha_t,\, \beta_t)$ which corresponds to a rotation of the plate about a horizontal axis $\boldsymbol{w}_t = (\cos\alpha_t,\, \sin\alpha_t,\, 0)^\top$ by an angle β_t (all quantities expressed in the motion frame). The desired plate pose (expressed in the robot's base frame) is given by: $\widetilde{\boldsymbol{T}}_t = \left(\widetilde{\boldsymbol{R}}_t^e,\, \tilde{\boldsymbol{p}}_t^e\right) \in SE(3)$, where the orientation and position components are computed using Rodrigues' formula ($\hat{\boldsymbol{w}}_t \in \mathbb{R}^{3\times3}$ is the skew-symmetric matrix of $\boldsymbol{w}_t$):

$$\begin{aligned}
\widetilde{\boldsymbol{R}}_t^e &= \boldsymbol{R}_z(\phi)\left(\mathbb{I} + \hat{\boldsymbol{w}}_t \sin\beta_t + \hat{\boldsymbol{w}}_t{}^2\left(1 - \cos\beta_t\right)\right) \in SO(3) \\
\tilde{\boldsymbol{p}}_t^e &= \boldsymbol{R}_z(\phi)\,\boldsymbol{\delta}_t + \boldsymbol{p}_t^e \in \mathbb{R}^3
\end{aligned} \tag{12}$$

The desired joint configuration of the robot $\tilde{\boldsymbol{q}}_t$ is obtained via inverse kinematics (M being the robot's DoF): $\tilde{\boldsymbol{q}}_t = \text{IK}\left(\widetilde{\boldsymbol{T}}_t\right) \in \mathbb{R}^M$. The desired joint configuration is tracked using a joint-space torque controller that generates real-time joint torques (no inertial feedforward term is included, as the desired joint acceleration is zero):

$$\boldsymbol{\tau} = \boldsymbol{K}\left(\tilde{\boldsymbol{q}}_t - \boldsymbol{q}\right) - \boldsymbol{D}\dot{\boldsymbol{q}} + \boldsymbol{C}\left(\boldsymbol{q},\, \dot{\boldsymbol{q}}\right) + \boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^M \tag{13}$$

where $\boldsymbol{q},\, \dot{\boldsymbol{q}} \in \mathbb{R}^M$ are the observed joint angles and velocities from sensor readings; $\boldsymbol{K},\, \boldsymbol{D} \in \mathbb{R}^{M\times M}$ are the stiffness and damping gain matrices; $\boldsymbol{C}\left(\boldsymbol{q},\, \dot{\boldsymbol{q}}\right),\, \boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^M$ are the Coriolis and gravity terms, respectively.

V. Training and Validation in Simulation

In this section, we implement the reactive catching task, train the policies purely in simulation using our proposed strategy, and evaluate its performance under various sources of uncertainty.

A. Training in Simulation

The task simulation was implemented in ManiSkill [46]. We instantiated 128 parallel simulation environments for training, each containing a robot manipulating a set of multiple different balls (i.e., DRIS) for up to 20 steps (corresponding to one second of simulated time). When launching each environment instance, the balls are randomly positioned at a horizontal distance between $[1.0,\, 2.0]\,m$ away from the plate and assigned an initial velocity such that they reach a predefined catching region after $[1.0,\, 1.5]\,s$, approaching from a random incoming direction. The catching region is defined as a zone at the same height as the plate and within a horizontal radius of $0.2m$ from its center. Each episode begins when the ball is about to enter this region, specifically within a random lead time of $[0.08, 0.12]\,s$ before arrival, at which point the robot starts executing generated actions. The initial joint angles of the robot are sampled from a Gaussian distribution centered at the home configuration with a standard deviation of 0.02 radians per joint. We disable collision detection in simulation between balls within the same DRIS and between balls and all robot links except the plate (i.e., only ball-plate collisions are enabled) to eliminate unmodeled contacts for stable training.

We first collected training data for the DRIS encoder by allowing the robot to interact with the DRIS containing 200 balls using random actions for 50 episodes. Across all 128 parallel environment instances, this resulted in a total of $128,000$ recorded DRIS state samples (i.e., the states of all balls at each step). The encoder was then trained on this dataset for 100 epochs, which took approximately 10 minutes. We then froze the pre-trained DRIS encoder and used it to train the FiLM-conditioned policy for 1000 epochs with PPO, which took approximately 2 hours.

B. Evaluation against Uncertainties

To evaluate the effect of DRIS size on policy performance, we trained separate policies with different numbers of balls in the DRIS, specifically $N = 1, 10, 50, 200$. Moreover, to compare training with the proposed DRIS-based strategy against a standard RL setting, we trained an additional baseline policy using the same neural network architecture but learned end-to-end directly from the observed single-ball state to the desired action. We refer to this baseline as *E2E*. During evaluation, the DRIS-trained policies, although trained with multiple balls in each DRIS, were always tested on catching a single ball to reflect the real task setting.

We evaluate robustness against the following three sources of uncertainty:

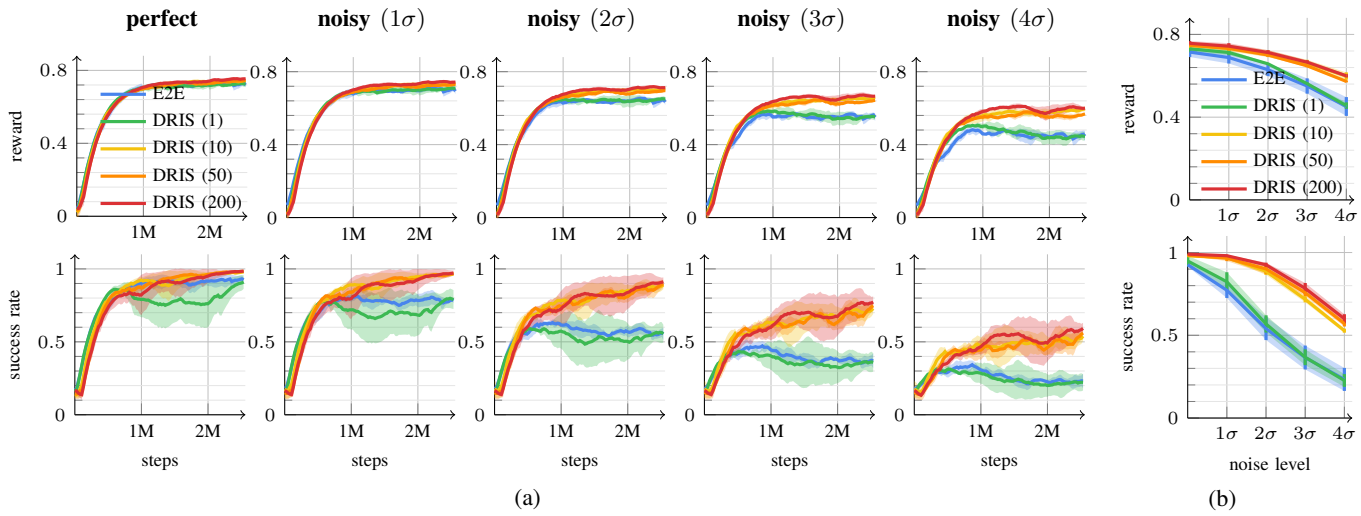


Fig. 4: Reward (top) and success rate (bottom) under varying observation noise. (a) Training curves across simulation steps, evaluated under progressively larger noise levels (left to right); (b) Evaluation of each policy at each noise level, averaged over the last 100 epochs.

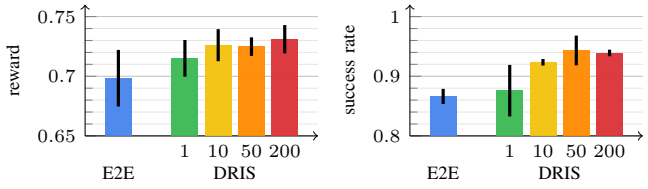


Fig. 5: Reward (left) and success rate (right) of each policy under execution errors, averaged over the last 100 epochs.

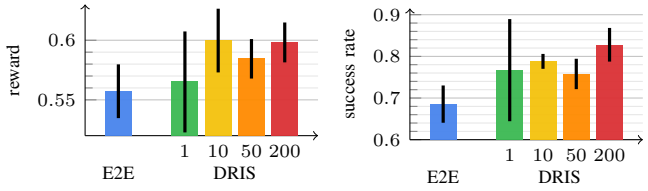


Fig. 6: Reward (left) and success rate (right) of each policy when manipulating balls with unseen restitution values (higher than those in training), averaged over the last 100 epochs.

Observation Noise. This setting mimics scenarios where the ball's state is not accurately estimated (e.g., due to camera calibration errors or tracking inaccuracies). As shown in Fig. 4, we inject Gaussian noise into the observed ball states at different noise levels, obtained by scaling a base standard deviation σ ($1\,cm$ for position and $5\,cm/s$ for velocity) by factors ranging from 1 to 4. We report both the final episode reward and success rate, where success is defined as the ball remaining on the plate with a velocity below $0.1\,m/s$ at the end of the episode. The E2E baseline and the DRIS policy with $N = 1$, both trained using a single ball (where the DRIS policy reduces to a standard RL setting but with a pre-trained encoder), perform comparably under perfect observations. However, as the observation noise increases, their performance degrades significantly. Moreover, they are more prone to overfitting to the perfect observations, as can be seen from their gradual performance degradation when trained for a longer period in Fig. 4. In contrast, DRIS policies trained with even a slightly larger number of balls (e.g., 10) exhibit substantially improved robustness, demonstrating greater tolerance to observation noise.

Execution Error. Then, we added noise to the desired joint positions by uniformly sampling perturbations from $[-0.05, 0.05]$ rad to simulate robot execution errors (e.g., controller's tracking inaccuracies). We report the final reward and success rate for each policy in Fig. 5. Similar to the observation noise evaluation, the E2E baseline is notably less robust than policies trained with DRIS. While increasing the DRIS size generally improved robustness in this evaluation, even a modest number of balls (e.g., 10) provided an obvious boost in performance under execution noise.

Out-of-Distribution Physics. To evaluate how the policies perform under unseen physical parameters, we tested them on balls with higher and more challenging restitution coefficients in $[0.7, 0.8]$, despite being trained only within $[0.4, 0.7]$. The reward and success rate are reported in Fig. 6. Compared with both the E2E baseline and the DRIS policy trained with a single ball, policies trained with larger DRIS generalize better to manipulate balls with unseen physical properties (restitution, in this case).

C. Scalability and Computational Efficiency

To evaluate the efficacy of DRIS from a computational fairness perspective, we conducted a scaling study to verify that the observed performance gains are not merely a product of increased ball interactions. Specifically, we expanded the E2E baseline to 128×50 single-ball environments and compared it against our DRIS configuration, which utilizes

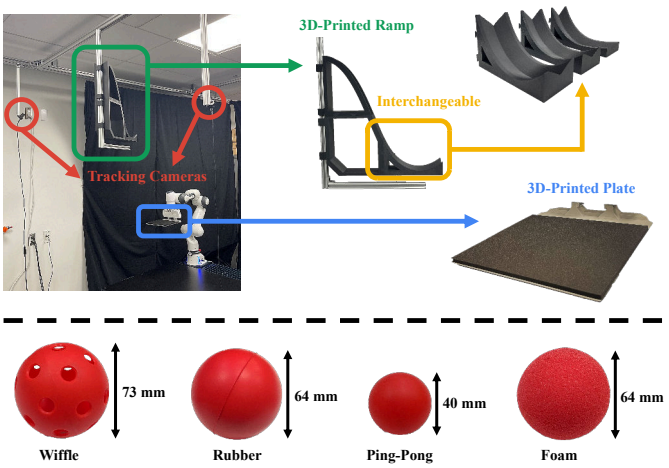


Fig. 7: The system setup (top) and the four different balls (bottom) used for real-world experiments.

128 environments with a DRIS size of 50 (i.e., 50 balls per environment), thereby matching the total number of ball interactions across both methods.

Evaluated under 2σ observation noise, the results indicate that while massive parallelization improves the E2E baseline success rate from 0.55 to 0.73, it does so at a substantial computational cost, increasing simulation VRAM usage from 0.65 GB to 2 GB. In contrast, the DRIS configuration achieves a significantly higher success rate of 0.89 while increasing VRAM only modestly to 0.71 GB. These results demonstrate that DRIS-based propagation provides superior robustness while remaining considerably more resource-efficient than standard environment parallelization.

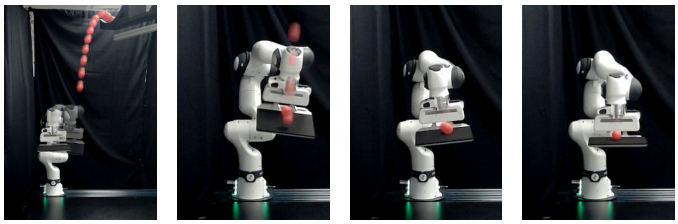
VI. Real-Robot Deployment

To more realistically challenge our learned policies under real-world uncertainties and evaluate their sim-to-real transfer performance, we directly deployed the policies trained purely in simulation in a zero-shot manner on a real 7-DoF Franka Research 3 (FR3) robot manipulator, without any fine-tuning using real-world data.

A. System Setup

Fig. 7 shows the setup we used for real-world evaluation (top), and the four test balls (wiffle, rubber, ping-pong, and foam) which differ in size, weight, and physical behavior (bottom). The plate is 3D-printed and covered with a neoprene foam padding to protect the robot from damage during repeated trials. Although the padding slightly reduces the impact force, its surface is smoother than the underlying plastic and causes the ball to roll and accelerate more easily, thus preserving the difficulty of the task.

To ensure relatively consistent initial ball states for fair comparative evaluation across different policies, we designed and 3D-printed a ramp with interchangeable lower sections and mounted it above the workspace. The ball is released by rolling from the top of the ramp, and the three interchangeable



Ball	Ramp	VelTrack	E2E	DRIS (Ours)
Wiffle	$R = 0.13\,m$	0 / 5	0 / 5	**5 / 5**
	$R = 0.20\,m$	0 / 5	2 / 5	**4 / 5**
	$R = 0.32\,m$	0 / 5	0 / 5	**3 / 5**
Rubber	$R = 0.13\,m$	0 / 5	0 / 5	**2 / 5**
	$R = 0.20\,m$	0 / 5	0 / 5	**5 / 5**
	$R = 0.32\,m$	0 / 5	1 / 5	**2 / 5**
Ping-Pong	$R = 0.13\,m$	0 / 5	1 / 5	**4 / 5**
	$R = 0.20\,m$	2 / 5	1 / 5	**5 / 5**
	$R = 0.32\,m$	0 / 5	0 / 5	**4 / 5**
Foam	$R = 0.13\,m$	0 / 5	0 / 5	**2 / 5**
	$R = 0.20\,m$	1 / 5	1 / 5	**3 / 5**
	$R = 0.32\,m$	0 / 5	**2 / 5**	**2 / 5**
Total		3 / 60	8 / 60	**41 / 60**
Success Rate		0.05	0.13	**0.68**

Fig. 8: *Top:* Snapshots of our policy catching a ping-pong ball released from the ramp. *Bottom:* Success rates for each policy across all combinations of ball types and ramp settings.

sections (with radii $R = 0.13$, 0.20, and $0.32\,m$) induce different release speeds, allowing evaluation under varying initial ball states.

B. Quantitative Evaluation

We deployed a single DRIS-trained policy (with $N = 200$) on the real robot across all evaluation trials and compared its performance against two baselines: 1) *VelTrack*: a hand-crafted policy that moves the plate horizontally to follow the ball based on its estimated velocity after the first impact, with the plate oriented to face the ball's incoming direction at the moment of first impact; 2) *E2E*: similar to the simulation evaluation in Sec. V, this is a policy trained end-to-end with a single ball in simulation and directly deployed on the real robot. The controller gains, i.e., $\boldsymbol{K}$ and $\boldsymbol{D}$ in Eq. (13), were identified on the real robot by matching the controller's behavior to that in simulation. Aside from this gain identification, no additional adaptation or fine-tuning was performed.

For each policy, and for every combination of ball type and ramp, we conducted 5 trials and evaluated performance based on the number of successful catches, as summarized in Fig. 8. A trial is considered a successful catch if the robot keeps the ball on the plate for at least 10 seconds after it begins moving. The VelTrack baseline rarely succeeded, as its response was not sufficiently reactive, particularly under inaccurate ball velocity estimates. Its only 3 successful trials occurred when the ball was accidentally trapped in the gap between the plate and the finger. The E2E baseline, although producing reasonable motions to contact the ball, was highly sensitive to observation noise, leading to jerky robot movements that frequently knocked the ball off the

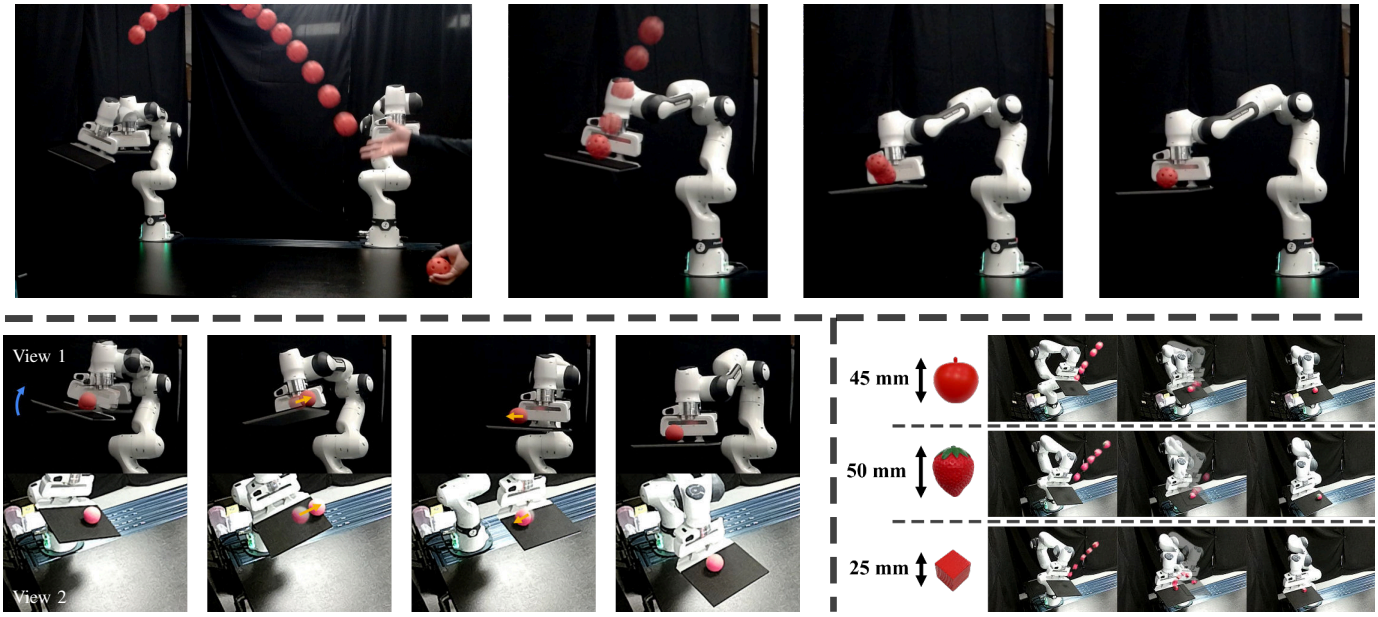


Fig. 9: Our policy successfully catches a wiffle ball thrown by a human (top); balances a foam ball (bottom left), where a hard-coded motion rotates the plate by 30° (blue arrow) to initiate the ball's rolling in the first frame; and catches irregularly shaped objects (bottom right).

plate or hit the robot's power limit. In contrast, our DRIS-trained policy achieved a 68% success rate when deployed zero-shot, and successfully caught all tested balls. While the system maintains high performance across diverse conditions, the few remaining failure cases typically occur when the initial contact cannot sufficiently dissipate the ball's momentum or redirect the ball unfavorably due to tracking latency or noise. In these instances, the robot may occasionally over-stretch while attempting to recover the high-speed trajectory.

C. Qualitative Evaluation

We conducted several qualitative evaluations to further assess the learned policy. First, as showcased in Fig. 9 (top), the policy successfully catches all test balls thrown by a human, despite the increased uncertainty and inconsistency of human throws. Second, as illustrated in Fig. 9 (bottom left), although trained for reactive catching, the policy generalizes directly to a pure balancing task. Third, as shown in Fig. 9 (bottom right), the policy was able to catch irregularly shaped objects, including a lightweight toy apple, a heavier toy strawberry, and a wooden cube. The wooden cube is caught only occasionally due to its highly unpredictable impact dynamics, which makes reliable catching difficult even for humans.

VII. Conclusion and Limitations

In this work, we propose a DRIS-based learning strategy that enables robust policy learning in simulation and improved zero-shot sim-to-real transfer for dexterous manipulation under severe physical uncertainty. By simultaneously propagating multiple randomized instances instead of one, DRIS better approximates state evolution under diverse physical conditions. Combined with a task-specific policy design, this enables stable reactive catching without passive mechanical aids. Extensive experiments show substantially improved robustness and sim-to-real performance over conventional baselines.

Limitations. While the proposed DRIS-based strategy is promising, we acknowledge several limitations to be addressed in future work. First, DRIS requires an encoder to map states of varying instances to a fixed-dimensional latent representation; while this is efficient for low-dimensional states, tasks involving high-dimensional visual or geometric inputs may require larger models and substantial pre-training. Second, effective DRIS training relies on efficient parallel propagation of multiple instances, which may be computationally expensive or impractical without suitable simulator support. This is particularly relevant when scaling to complex multi-body contact settings involving multiple mutual interactions, where the computational overhead of resolving simultaneous collision manifolds across the entire DRIS can become a significant bottleneck. Finally, since all instances share a single action, excessive divergence among instance states can destabilize learning, placing a practical limit on the amount of variation introduced. As noted in [42], preventing such divergence is easier in dissipative systems. While our method thrives in "energy-dissipating" tasks like catching, it may struggle in "energy-injecting" tasks like throwing. In such expansionary dynamics, physical variations are rapidly amplified, making it difficult for a shared action to reconcile the diverging states. Future work using adaptive or curriculum-based instance randomization during training might mitigate this challenge.

Acknowledgments

This work was supported by the U.S. National Science Foundation (NSF) under grant FRR-2240040.

References

[1] Saminda Abeyruwan, Alex Bewley, Nicholas Matthew Boffi, Krzysztof Marcin Choromanski, David B D'Ambrosio, Deepali Jain, Pannag R Sanketi, Anish Shankar, Vikas Sindhwani, Sumeet Singh, et al. Agile catching with whole-body mpc and blackbox policy learning. In *Learning for Dynamics and Control Conference*, pages 851–863. PMLR, 2023.

[2] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International Conference on Machine Learning (ICML)*, pages 40–49. PMLR, 2018.

[3] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 3066–3073. IEEE, 2018.

[4] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[5] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39 (1):3–20, 2020.

[6] Georg Bätz, Arhan Yaqub, Haiyan Wu, Kolja Kühnlenz, Dirk Wollherr, and Martin Buss. Dynamic manipulation: Nonprehensile ball catching. In *18th Mediterranean Conference on Control and Automation, MED'10*, pages 365–370. IEEE, 2010.

[7] Berthold Bäuml, Thomas Wimböck, and Gerd Hirzinger. Kinematically optimal catching a flying ball with a hand-arm-system. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 2592–2599. IEEE, 2010.

[8] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 723–730. IEEE, 2011.

[9] Nikhil Chavan-Dafle and Alberto Rodriguez. Prehensile pushing: In-hand manipulation with push-primitives. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 6215–6222. IEEE, 2015.

[10] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[11] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pages 297–307. PMLR, 2022.

[12] Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Iterative residual policy: for goal-conditioned dynamic manipulation of deformable objects. *The International Journal of Robotics Research*, 43(4):389–404, 2024.

[13] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[14] Nils Dengler, David Großklaus, and Maren Bennewitz. Learning goal-oriented non-prehensile pushing in cluttered scenes. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 1116–1122. IEEE, 2022.

[15] Ke Dong, Karime Pereida, Florian Shkurti, and Angela P Schoellig. Catch the ball: Accurate high-speed motions for mobile manipulators via inverse dynamics learning. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 6718–6725. IEEE, 2020.

[16] Ioannis Exarchos, Yifeng Jiang, Wenhao Yu, and C Karen Liu. Policy transfer via kinematic domain randomization and adaptation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 45–51. IEEE, 2021.

[17] Tobias Gold, Ralf Römer, Andreas Völz, and Knut Graichen. Catching objects with a robot arm using model predictive control. In *American Control Conference (ACC)*, pages 1915–1920. IEEE, 2022.

[18] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[19] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*, pages 2555–2565. PMLR, 2019.

[20] Binghao Huang, Yuanpei Chen, Tianyu Wang, Yuzhe Qin, Yaodong Yang, Nikolay Atanasov, and Xiaolong Wang. Dynamic handover: Throw and catch with bimanual hands. In *Conference on Robot Learning*, 2023.

[21] Johann Huber, François Hélénon, Hippolyte Watrelot, Faïz Ben Amar, and Stéphane Doncieux. Domain randomization for sim2real transfer of automatically generated grasping datasets. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4112–4118. IEEE, 2024.

[22] Ajinkya Jain and Scott Niekum. Efficient hierarchical robot motion planning under uncertainty and hybrid dynamics. In *Conference on Robot Learning*, pages 757–766. PMLR, 2018.

[23] Krishna Murthy Jatavallabhula, Miles Macklin, Florian

Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, et al. gradsim: Differentiable simulation for system identification and visuomotor control. *arXiv preprint arXiv:2104.02646*, 2021.

[24] Yifeng Jiang, Tingnan Zhang, Daniel Ho, Yunfei Bai, C Karen Liu, Sergey Levine, and Jie Tan. Simgan: Hybrid simulator identification for domain adaptation via adversarial reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2884–2890. IEEE, 2021.

[25] Seungsu Kim, Ashwini Shukla, and Aude Billard. Catching objects in flight. *IEEE Transactions on Robotics*, 30 (5):1049–1065, 2014.

[26] Ziang Liu, Stephen Tian, Michelle Guo, Karen Liu, and Jiajun Wu. Learning to design and use tools for robotic manipulation. In *Conference on Robot Learning*, pages 887–905. PMLR, 2023.

[27] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[28] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020.

[29] Fabio Muratore, Theo Gruner, Florian Wiese, Boris Belousov, Michael Gienger, and Jan Peters. Neural posterior domain randomization. In *Conference on Robot Learning*, pages 1532–1542. PMLR, 2022.

[30] Fabio Muratore, Fabio Ramos, Greg Turk, Wenhao Yu, Michael Gienger, and Jan Peters. Robot learning from randomized simulations: A review. *Frontiers in Robotics and AI*, 9:799893, 2022.

[31] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[32] Akio Namiki and Naoki Itoi. Ball catching in kendama game by estimating grasp conditions based on a high-speed vision system and tactile sensors. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 634–639. IEEE, 2014.

[33] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[34] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[35] Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, and Viktor Makoviychuk. Dexpbt: Scaling up dexterous manipulation for hand-arm systems with population based training. In *Robotics: Science and Systems*, 2023.

[36] Kai Ploeger, Michael Lutter, and Jan Peters. High acceleration reinforcement learning for real-world juggling with binary rewards. In *Conference on Robot Learning*, pages 642–653. PMLR, 2021.

[37] Josep M. Porta, Nikos Vlassis, Matthijs T.J. Spaan, and Pascal Poupart. Point-based value iteration for continuous pomdps. *Journal of Machine Learning Research*, 7 (83):2329–2367, 2006.

[38] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017.

[39] Fabio Ramos, Rafael Possas, and Dieter Fox. Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators. In *Robotics: Science and Systems*, 2019.

[40] Seyed Sina Mirrazavi Salehian, Mahdi Khoramshahi, and Aude Billard. A dynamical system approach for softly catching a flying object: Theory and experiment. *IEEE Transactions on Robotics*, 32(2):462–471, 2016.

[41] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning (ICML)*, pages 8459–8468. PMLR, 2020.

[42] Stefan Schaal and Christopher G Atkeson. Open loop stable control strategies for robot juggling. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 913–918. IEEE, 1993.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] Matthew Sheckells, Gowtham Garimella, Subhransu Mishra, and Marin Kobilarov. Using data-driven domain randomization to transfer robust control policies to mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3224–3230. IEEE, 2019.

[45] Nikhil Sobanbabu, Guanqi He, Tairan He, Yuxiang Yang, and Guanya Shi. Sampling-based system identification with active exploration for legged sim2real learning. In *Conference on Robot Learning*, 2025.

[46] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. In *Robotics:*

Science and Systems, 2025.

[47] Gabriele Tiboni, Karol Arndt, and Ville Kyrki. Dropo: Sim-to-real transfer with offline domain randomization. *Robotics and Autonomous Systems*, 166:104432, 2023.

[48] Gabriele Tiboni, Pascal Klink, Jan Peters, Tatiana Tommasi, Carlo D'Eramo, and Georgia Chalvatzaki. Domain randomization via entropy maximization. In *International Conference on Learning Representations (ICLR)*, volume 2024, pages 19841–19863, 2024.

[49] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[50] Josh Tobin, Lukas Biewald, Rocky Duan, Marcin Andrychowicz, Ankur Handa, Vikash Kumar, Bob McGrew, Alex Ray, Jonas Schneider, Peter Welinder, et al. Domain randomization and generative models for robotic grasping. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 3482–3489. IEEE, 2018.

[51] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE, 2012.

[52] Andrew Wang, Andrew C Li, Toryn Q Klassen, Rodrigo Toro Icarte, and Sheila A McIlraith. Learning belief representations for partially observable deep rl. In *International Conference on Machine Learning (ICML)*, pages 35970–35988. PMLR, 2023.

[53] Gaotian Wang, Kejia Ren, Andrew S Morgan, and Kaiyu Hang. Caging in time: A framework for robust object manipulation under uncertainties and limited robot perception. *The International Journal of Robotics Research*, page 02783649251343926, 2025.

[54] Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

[55] Yuanhang Zhang, Tianhai Liang, Zhenyang Chen, Yanjie Ze, and Huazhe Xu. Catch it! learning to catch in flight with mobile dexterous hands. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025.