Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

DexPBT: Scaling up Dexterous Manipulation for Hand-Arm Systems with Population Based Training

Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, Viktor Makoviychuk

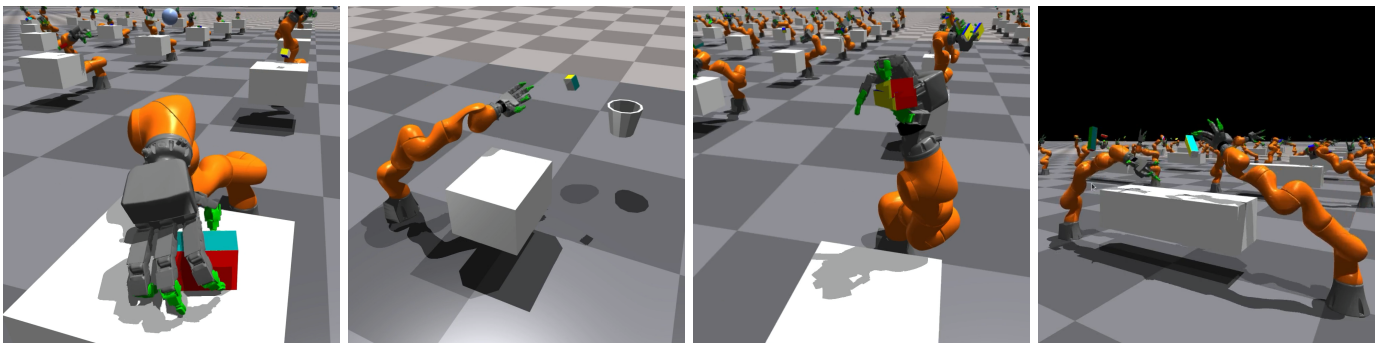

Fig. 1: Tasks trained with DexPBT. Left-to-right: *regrasping*, *throwing*, *single-handed reorientation*, *two-handed reorientation*.

***Abstract*—In this work, we propose algorithms and methods that enable learning dexterous object manipulation using simulated one- or two-armed robots equipped with multi-fingered hand end-effectors. Using a parallel GPU-accelerated physics simulator (Isaac Gym), we implement challenging tasks for these robots, including regrasping, grasp-and-throw, and object reorientation. To solve these problems we introduce a decentralized Population-Based Training (PBT) algorithm that allows us to massively amplify the exploration capabilities of deep reinforcement learning. We find that this method significantly outperforms regular end-to-end learning and is able to discover robust control policies in challenging tasks. Video demonstrations of learned behaviors and the code can be found at the supplementary website.**

I. Introduction

In recent years researchers have started to apply deep reinforcement learning methods in an increasing number of challenging continuous control domains. Some applications include impressive demonstrations of skill in virtual environments, such as playing football by directly controlling humanoid characters using joint torques [22]. In other domains such as agile drone flight [37, 28, 4] and quadruped locomotion [17, 27], control policies trained in simulation are deployed directly on the real hardware.

Learning-based approaches appear to be particularly promising in the domain of robotic manipulation. Traditional methods such as direct trajectory optimization, may struggle to model complex contact dynamics. Due to these difficulties, researchers working on manipulation problems typically focus their efforts on robotic arms with end-effectors that simplify contact handling, such as parallel jaw grippers [5, 29, 26]. Even though more capable human-like robotic hands have the potential to endow robots with far more advanced manipulation capabilities, this type of end-effector remains relatively unpopular due to the difficulty of controlling high degree-of-freedom (DoF) systems in contact-rich environments.

With the advent of modern deep RL methods that use a large amount of data and computation, it has become possible to learn control policies for multi-fingered robotic hands. Systems like "Dactyl" [2, 31] and "DeXtreme" [10] have demonstrated how large-scale reinforcement learning in simulation can be used to obtain robust policies for complex in-hand object manipulation.

In this work, we extend this approach and apply it to a fully actuated hand-arm system: a four finger 16-DoF Allegro Hand mounted on a 7-DoF Kuka arm. We target a variety of manipulation tasks in simulated environment Isaac Gym [24], such as regrasping, throwing, and reorientation. We then scale our learning method to train agents that control a pair of arms and hands with combined 46 degrees of freedom using a single neural network policy.

We observe that despite the relative success of straightforward end-to-end learning, our RL experiments are characterized by a high variance of results and dependency on initial conditions, especially in tasks that require exploration in the vast space of possible behaviors. To that end, we develop a Population-Based Training (PBT) algorithm [13], an outer optimization loop that can significantly amplify the exploration capabilities of end-to-end RL. In order to facilitate asynchronous learning in a volatile compute environment we implement a *decentralized* version of the PBT algorithm that we can run without a central orchestrator instance and is robust to the disconnect of one or a few learners. We find that the PBT approach demonstrates improved performance in all scenarios over standard end-to-end learning and becomes the enabling factor for the successful training of ambidextrous agents that control two hand-arm systems simultaneously.

Our contributions can be summarized as follows:

- We develop a framework that combines on-policy RL and decentralized Population Based Training with realistic GPU-accelerated robotic simulation and use this framework to train policies for dexterous object manipulation

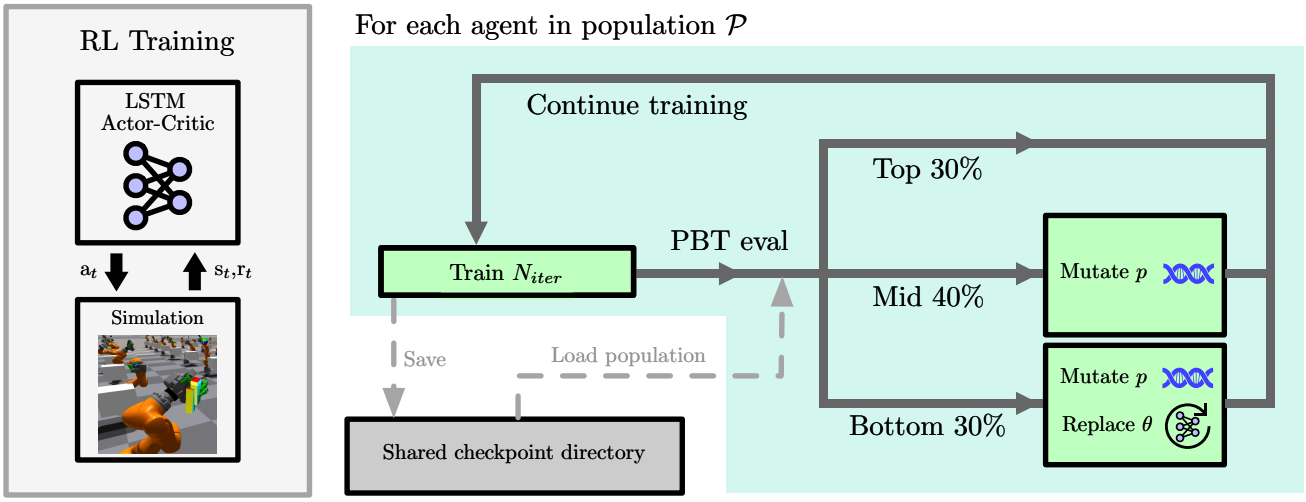

Fig. 2: An illustration of the system used to solve our complex manipulation tasks using a combination of RL, highly parallelized robotic simulation, and Population Based Training (PBT).

with high-DoF single and dual hand-arm systems.

- We introduce a staged reward function formulation and a PBT meta-objective to simplify and automate reward tuning and hyperparameter search.
- We release our environments, RL, and PBT code to facilitate further research in dexterous robotic manipulation (see the supplementary website).

II. Related Work

Producing control policies to perform complex, contact-rich tasks has been a long-standing challenge in robotics. Classical methods for this have focused on directly leveraging robot kinematics [34, 25, 20]. While useful in free-space or pick-and-place style tasks, these methods struggle as the number of contacts grows.

Recently, various systems have leveraged learning-based methods to perform contact-rich robotic manipulation, achieving impressive results both in simulation and reality [18, 15]. In particular, RL-based methods in simulation have shown the ability to learn complex and robust behaviors in realistic scenarios which transfer to the real world [17, 2, 10]. The advent of high-throughput simulation on GPUs has improved the speed at which such tasks can be learned using RL [24, 33, 1, 12, 8].

Multiple prior projects have explored the problem of dexterous object manipulation. Kumar et al. [16] achieved in-hand object rotation with a Shadow Hand-like robotic hand using a model-based approach. Andrychowicz et al. [2] and Handa et al. [10] showed that it was possible to train the policy to do in-hand cube manipulation and even Rubik's cube [31] solving entirely in simulation and deploy the learned policy on the real robot. Other work has shown multiple tasks with free-floating hands [6] or with object grasping [41]. Matl et al. [26] demonstrated methods for learning grasping policies with two robotic arms and different types of end-effectors such as suction cups and parallel jaw grippers. Gupta et al. [9] were able to learn object manipulation skills on a hand-arm system using off-policy reset-free reinforcement learning directly in the real world. Our approach is most similar to Handa et al. [10]: we also take advantage of the massively parallel GPU-accelerated physics engine, and we add a Population-Based Training outer loop for improved exploration, automated reward function tuning, and hyperparameter optimization.

Jaderberg et al. [13] popularized Population-Based Training in a variety of domains such as RL, adversarial learning, and machine translation, however PBT methods turned out to be particularly promising in deep RL where exploration is often the bottleneck. PBT provides a way to combine the exploration power of multiple learners and directs resources toward more promising behaviors. Since then, PBT algorithms have been exceptionally successful in RL for video game applications and helped to produce state-of-the-art agents for games such as Quake [14], Doom [32], and Starcraft II [38].

Recently Wan et al. [39] augmented PBT-style methods with trust-region based Bayesian Optimization and were able to optimize both hyperparameters and model architectures simultaneously. Flajolet et al. [7] proposed highly efficient JAX implementation of PBT that enables evaluation of multiple agents on one accelerator. Both Wan et al. [39] and Flajolet et al. [7] demonstrated great results on standard continuous control benchmarks such as Half-Cheetah and Humanoid, in contrast, we apply our method in complex dexterous manipulation domains more similar to real robot settings. Additionally, our *decentralized* PBT implementation (Section III-D) makes it easy to use the algorithm in distributed compute environments such as Slurm clusters.

III. Method

A. Problem Statement

A lot of problems in practical robotics require the robot to perform some notion of *rearrangement* [3], *i.e.* bringing a given environment into a specified state. In household, factory, or warehouse environments many interesting tasks will involve

a type of rearrangement that requires dexterous object handling and manipulation [30]. In this work, we target a high-DoF anthropomorphic hand+arm system in the attempt to build towards the desired level of dexterity.

We focus on a problem that can be seen as a special case of rearrangement: *single-object reposing*. This task requires changing the state of a single rigid body such that it matches the target position $x \in \mathbb{R}^3$ and, optionally, orientation $R \in SO(3)$. Dexterous single-object manipulation can be seen as an essential primitive required to perform general-purpose rearrangement. This task requires mastery of contact-rich grasping and in-hand manipulation and presents an exciting challenge for robotics research.

We approach dexterous manipulation as a discrete-time sequential decision-making process. At each step the controller observes environment state $s_t \in \mathbb{R}^{N_{obs}}$ (which includes the target object pose) and yields an action $a_t \in \mathbb{R}^{N_{dof}}$ specifying the desired angles of arm and finger joints. Whenever the object state matches the target within a specified tolerance, the attempt is considered successful, and the target state is reset. If the object is dropped during the attempt, or if the target state is not achieved within a time period τ we consider this attempt failed. The simulation proceeds until N_{max} consecutive successes are reached or until the first failure. The performance on the task can thus be measured as a number of consecutive successes within the episode $N_{succ} \leq N_{max}$, as done in prior work [2].

Similar to Allshire et al. [1] we use N_{kp} keypoints to represent the observed object pose $x_{kp} \in \mathbb{R}^{3N_{kp}}$. In tasks that require orientation matching, keypoints are also used to represent the desired object pose $x_{targ} \in \mathbb{R}^{3N_{kp}}$ (Figure 3). We consider such task successfully executed when all of N_{kp} keypoints are within the tolerance threshold of their corresponding target locations $\max_{i \in 1..N_{kp}} ||x^{(i)}_{targ} - x^{(i)}_{kp}|| \leq \varepsilon^*$. For tasks that require only position matching we just require the object's center to be within ε^* of the desired location.

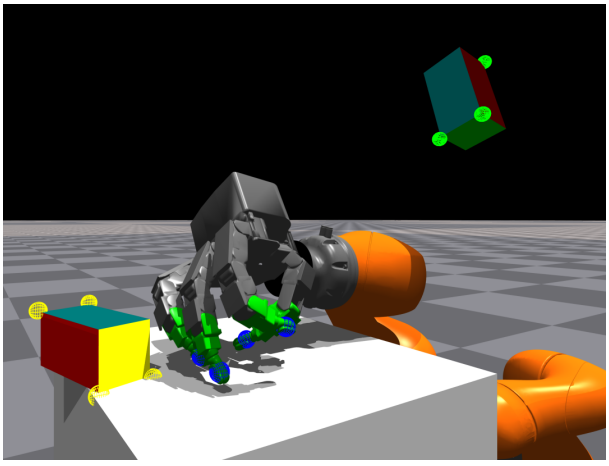

Fig. 3: We use object keypoints (yellow) to represent both observed object position $x_{kp} \in \mathbb{R}^{3N_{kp}}$ and object shape. In reorientation tasks keypoints are also used to represent the desired object pose $x_{targ} \in \mathbb{R}^{3N_{kp}}$ (green). Observed fingertip locations are additionally rendered in blue.

The keypoint representation provides multiple benefits. It eliminates the need to tune tolerance thresholds and RL reward weights separately for rotation and translation. In addition to that, keypoints placed in a predefined pattern on the convex hull of the object allow the policy to observe the shape of the rigid body. In this work we use parallelepipeds with dimensions ranging from $3\,\mathrm{cm}$ to $30\,\mathrm{cm}$ as our manipulation targets, therefore a representation with $N_{kp} = 4$ keypoints is sufficient to convey the shape information.

B. Scenarios

We developed three variants of our object manipulation task that highlight different challenges in dexterous manipulation. These scenarios are *regrasping*, *grasp-and-throw*, and *reorientation* (Figure 1). At the beginning of each episode, the object appears in a random position on the table and the hand-arm system is reset to a random state $s_{robot} \in \mathbb{R}^{2N_{dof}}$ consisting of random joint angular velocities and initial angles within the DoF limits. The episode then proceeds according to one of the following scenarios.

The *regrasping* task demands that the agent grasp the object, pick it up from the table, and hold it in a specified location for a duration of time, after which both object and target positions are reset. To succeed in this scenario the control policy must develop stable grasps that minimize the probability of dropping the object during the attempt.

In *grasp-and-throw* the task is to pick up the object and displace it into a container that can be outside of manipulator's reach. This scenario requires aiming for the container and throwing the object a significant distance. Here we test the ability of the control policy to understand the dynamic aspects of object manipulation: successful execution of this task requires releasing the grip at the right point of the trajectory, giving the object just the right amount of momentum to direct it towards the goal. After each attempt we reset the positions of the object and the container to make sure that the policy is able to complete the task repeatedly for a diverse set of initial conditions.

The *reorientation* task provides perhaps the most difficult object manipulation challenge among the chosen scenarios. The goal is to grasp the object and consecutively move it to different target positions and orientations. This scenario requires maintaining a stable grip for minutes of simulated time, fine control of the joints of the robotic arm, and occasional in-hand rotation when the reorientation cannot be performed by merely using the affordances of the Kuka arm (video demonstration). Thus formulated, the reorientation task includes elements seen in previous work such as dexterous in-hand manipulation [31, 10], but extends the capability of the manipulator to perform reposing in much larger volume. While regrasping and grasp-and-throw tasks only require matching the target position, reorientation scenario additionally demands that object rotation matches the target.

In both regrasping and reorientation the final required tolerance is $\varepsilon^* = 1\,\mathrm{cm}$, thus demanding very precise control.

TABLE I: Actor/critic observations and their dimensionality. Here $N_{arm} = 1$ for single-arm and $N_{arm} = 2$ for dual-arm tasks, and $N_{kp} = 4$.

INPUT	DIMENSIONALITY
JOINT ANGLES	$23D * N_{arm}$
JOINT VELOCITIES	$23D * N_{arm}$
HAND POSITION	$3D * N_{arm}$
HAND ROTATION	$3D * N_{arm}$
HAND VELOCITY	$3D * N_{arm}$
HAND ANGULAR VELOCITY	$3D * N_{arm}$
FINGERTIP POSITIONS	$12D * N_{arm}$
OBJECT KEYPOINTS REL. TO HAND	$3D * N_{kp} * N_{arm}$
OBJECT KEYPOINTS REL. TO GOAL	$3D * N_{kp}$
OBJECT ROTATION	$4D$ (QUATERNION)
OBJECT VELOCITY	$3D$
OBJECT ANGULAR VELOCITY	$3D$
OBJECT DIMENSIONS	$3D$
$\mathbb{1}_{picked}$	$1D$
$d_{closest}$	$1D$
$\hat{d}_{closest}$	$1D$
TOTAL, ONE ARM TASKS	$110D$
TOTAL, DUAL-ARM TASKS	$192D$

For grasp-and-throw we use $\varepsilon^* = 7.5\,\mathrm{cm}$ since the main focus is on landing the object into the container, not the precise positioning within it.

Dual-Arm Scenarios. In the attempt to find the limits of end-to-end learning for continuous control, we introduce versions of regrasping and reorientation scenarios for two hand-arm systems. It is likely not a coincidence that humans, sculpted into a form optimised for object manipulation by evolution, wield not one but a pair of arms and hands. Thus solving object reposing with two high-DoF manipulators in simulation can be seen as a milestone on the path towards future robotic systems that can match or exceed human dexterity.

To produce dual-arm scenarios we double the number of simulated robots per task for the total of 46 degrees of freedom (Figure 1, on the right). We change the sampling of initial and target object positions in a way that guarantees that the task cannot be solved by any one robot. Thus, a complete solution requires grasping the object, passing the object from one hand to another, as well as in-hand manipulation, combining the most challenging elements from all scenarios. We extend both observation (Table I) and action space ($a_t \in \mathbb{R}^{N_{dof} * N_{arm}}, N_{arm} = 2$) to allow a single policy to control both manipulators.

C. Reinforcement Learning

We formalize the problem as a Markov Decision Process (MDP) where the agent interacts with the environment to maximize the expected episodic discounted sum of rewards $\mathbb{E}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$. We use Proximal Policy Optimization algorithm [36] to simultaneously learn the policy π_θ and the value function $V_\theta^\pi(s)$, both parameterized by a single parameter vector θ. The model architecture is an LSTM [11] followed by a 3-layer MLP. Even though agents can observe all relevant parts of the environment state, we decided to follow prior work [2, 31] and train recurrent models because any future real-world deployment will necessarily involve partial observability and require memory for test-time system identification.

The policy is trained using experience simulated in Isaac Gym [24], a highly parallelized GPU-accelerated physics engine. To process high volume of data generated by this simulator we use an efficient PPO implementation [23] which keeps the computation graph entirely on the GPU. Combined with the minibatch size of 2^{15} transitions, this allows us to maximize the hardware utilization and learning throughput.

We utilize normalization of observations, advantages, and TD-returns [35] to make the algorithm invariant to absolute scale of observations and rewards. We also use an adaptive learning rate algorithm that maintains a constant KL-divergence $D_{KL}(\pi|\pi_{old})$ between the current policy π_θ and the behavior policy $\pi_{\theta_{old}}$ that collected the rollouts. The learning rate is reduced by a factor of 1.5 whenever D_{KL} exceeds the threshold by the end of the training iteration, and is multiplied by 1.5 when D_{KL} falls below the threshold.

Both regrasping and reorientation demand very precise control ($\varepsilon^* = 1\,\mathrm{cm}$). Because of that, agents almost never encounter successul task execution early in the training. In order to create a smooth learning curriculum we adaptively anneal the tolerance from a larger initial value $\varepsilon_0 = 7.5\,\mathrm{cm}$. We periodically check if the policy crossed the performance threshold $N_{succ} > 3$, and in this case we decrease the current success tolerance until it reaches the final value ε^*: $\varepsilon \leftarrow \max(0.9\varepsilon, \varepsilon^*)$.

Reward function. For the successful application of any reinforcement learning method, the reward should be dense enough to facilitate exploration yet should not distract the agent from the sparse final objective (which in our case is to maximize the number of consecutive successful manipulations).

We propose a reward function that naturally guides the agent through a sequence of motions required to complete the task, from reaching for the object to picking it up and moving it to the final location:

$$r(s, a) = r_{reach}(s) + r_{pick}(s) + r_{targ}(s) - r_{vel}(a). \quad (1)$$

Here r_{reach} rewards the agent for moving the hand closer to the object at the start of the attempt:

$$r_{reach} = \alpha_{reach} * \max(d_{closest} - d, 0), \quad (2)$$

where both d and $d_{closest}$ are distances between the end-effector and the object, d is the current distance, and $d_{closest}$ is the closest distance achieved during the attempt so far. In dual-arm scenarios we calculate distance d for the end-effector that's closer to the object.

Component r_{pick} rewards the agent for picking up the object and lifting it off the table:

$$r_{pick} = (1 - \mathbb{1}_{picked}) * \alpha_{pick} * h_t + r_{picked}. \quad (3)$$

In this equation $\mathbb{1}_{picked}$ is an indicator function which becomes 1 once the height of the object relative to the table h_t exceeds a predefined threshold of $15\,\mathrm{cm}$. At this moment the agent

receives an additional sparse reward r_{picked}. Once the object is picked up, r_{targ} rewards the agent for moving the object closer to the target state:

$$r_{targ} = \mathbb{1}_{picked} * \alpha_{targ} * \max(\hat{d}_{closest} - \hat{d}, 0) + r_{success}. \quad (4)$$

In the reorientation task $\hat{d} = \max_{i \in 1..N_{kp}} ||x^{(i)}_{targ} - x^{(i)}_{kp}||$ is the maximum distance between corresponding pairs of object and target keypoints, while in tasks that do not require orientation matching $\hat{d}$ is simply the distance between the object center and the target location; in both cases $\hat{d}_{closest}$ is the smallest $\hat{d}$ achieved during the attempt so far. A large sparse reward $r_{success}$ is added when the desired position and/or orientation is reached, $\hat{d} = \hat{d}_{closest} \leq \varepsilon^*$.

Finally, r_{vel} in Eq. (1) is a simple joint velocity penalty that can be tuned to promote smoother movement, and in Equations (2) to (4) $\alpha_{reach}, \alpha_{pick}, \alpha_{targ}$ are relative reward weights. Note that we apply virtually the same reward function in all scenarios, sans the minor differences in $\hat{d}$ calculation.

Overall, our reward formulation follows a sequential pattern: the reward components r_{reach}, r_{pick} and r_{targ} are mutually exclusive and do not interfere with each other. For example: by the time the hand approaches the object, the component r_{reach} is exhausted since $d = d_{closest} = 0$, therefore r_{reach} does not contribute to the reward for the remainder of the trajectory. Likewise, $r_{pick} \neq 0$ if and only if $r_{targ} = 0$ and vice versa due to the indicator function $\mathbb{1}_{picked}$. The fact that only one major reward component guides the motion at each stage of the trajectory makes it easier to tune the rewards and avoid interference between reward components. This allows us to avoid many possible local minima: for example, if r_{pick} and r_{targ} are applied together, depending on the relative reward magnitudes the agent might choose to slide the object to the edge of the table closer to the target location to maximize r_{targ} and cease further attempts to pick it up and solve the problem for the fear of dropping the object.

In addition to that, rewards in Equations (2) and (4) have a predefined maximum total value depending on the initial distance between the hand and the object, and the object and the target respectively. This eliminates an entire class of reward hacking behaviors where the agent would remain close but not quite at the goal to keep collecting the proximity reward. In our formulation, only movement towards the goal is rewarded while mere proximity to the goal is not.

Observations and actions. In our experiments both actor π_θ and critic $V^\pi_\theta(s)$ observe environment state directly, including joint angles and velocities, positions of fingertips, object rotation, velocity, and angular velocity. Additionally, keypoint positions and object dimensions provide information about the object shape. Table I lists all observations available to the agent and their corresponding dimensionalities.

The policy π_θ outputs two vectors $\mu, \sigma \in \mathbb{R}^{N_{dof} * N_{arm}}$ which are used as parameters of $N_{dof} * N_{arm}$ independent Gaussian probability distributions. Actions are sampled from these distributions $a \sim \mathcal{N}(\mu, \sigma)$, clipped to corresponding joint limits and interpreted as target joint angles. Then a PD controller yields joint torques in order to get joints to the target angles specified by the policy.

TABLE II: RL hyperparameters and reward function coefficients. Rightmost column provides an example of the highest scoring agent's final parameter values for a single dual-arm reorientation PBT experiment (parameters not optimized by PBT are omitted).

PARAMETER	INITIAL VALUE	PBT-OPTIMIZED VALUE
LSTM SIZE	768	-
MLP LAYERS	[768,512,256]	-
NONLINEARITY	ELU	-
DISCOUNT FACTOR γ	0.99	0.9888
GAE DISCOUNT λ [35]	0.95	-
LEARNING RATE	ADAPTIVE (SEC III-C)	-
ADAPT. LR $D_{KL}(\pi \vert \pi_{old})$	0.016	0.01432
GRADIENT NORM	1.0	1.028
PPO-CLIP ϵ	0.1	0.2564
CRITIC LOSS COEFF.	4.0	5.188
ENTROPY COEFF.	0	-
NUM. AGENTS	8192	-
MINIBATCH SIZE	32768	-
ROLLOUT LENGTH	16	-
NUM. PPO EPOCHS	2	1
α_{reach}	50	74.8
α_{pick}	20	22.1
r_{picked}	300	414.4
α_{targ}	200	263.5
$r_{success}$	1000	1322.7

It is unrealistic to expect that all of the mentioned observations are available on the real robot. In this case, we propose using all aforementioned observations for the critic during training, while the policy only receives a subset of observations that can be obtained from *i.e.* a vision system, also known as asymmetric actor-critic approach (see [10, 31]).

D. Population-Based Training

A contact-rich continuous control problem with up to 192 observation dimensions (Table I) and up to 46 action dimensions can be exceptionally challenging even for modern RL algorithms [23]. The main challenge is exploration: from the large number of possible behaviors that maximize rewards early in the training only relatively few lead to high-performance solutions at convergence.

Another dimension of complexity when using learning methods is hyperparameter tuning. Any reward shaping scheme contains a number of coefficients that need to be carefully balanced in order to maximize the objective. In addition to that, modern RL algorithms have a substantial number of settings, such as learning rate, number of training epochs on each set of collected trajectories, relative magnitudes of actor and critic losses, and so on. Choosing these parameters can be quite challenging and heavily relies on the expertise of engineers and researchers.

In order to mitigate these problems we employ a Population-Based Training approach [13]. The core idea is akin to an evolutionary algorithm: we train a population of agents $\mathcal{P}$, perform mutation to generate promising hyperparameter combinations, and use selection to prioritize agents with

TABLE III: Parameters of the PBT algorithm.

PARAMETER	VALUE
POPULATION SIZE $\lvert\mathcal{P}\rvert$	8, 16, OR 32 AGENTS
POPULATION SPLIT $\lvert\mathcal{P}_{top}\rvert, \lvert\mathcal{P}_{mid}\rvert, \lvert\mathcal{P}_{bottom}\rvert$	30%, 40%, 30% OF $\lvert\mathcal{P}\rvert$
INITIAL DELAY N_{start}	200,000,000 ENV. STEPS
BURN-IN AFTER MUTATION N_{adapt}	50,000,000 ENV. STEPS
NORMAL PBT PERIODICITY N_{iter}	20,000,000 ENV. STEPS
MUTATION PROBABILITY β_{mut}	0.2
MIN. PERTURBATION μ_{min}	1.1
MAX. PERTURBATION μ_{min}	1.5

superior performance. Each agent $(\theta_i, p_i) \in \mathcal{P}$ is characterized by a parameter vector θ_i and a set of hyperparameters p_i, which includes settings of the RL algorithm as well as reward coefficients $\alpha_{reach}, \alpha_{pick}, \alpha_{targ}, r_{picked}, r_{success}$ (see Table II). Periodically (after training on N_{iter} environment transitions), each agent is evaluated to obtain the target performance metric r_{meta} and the population is sorted according to this metric into three subsets $\mathcal{P}_{top}, \mathcal{P}_{mid}, \mathcal{P}_{bottom} \subset \mathcal{P}$ with cardinality equal to 30%, 40%, and 30% of $\mathcal{P}$ respectively (see Figure 2 and Algorithm 1). Note that for performance reasons we use r_{meta} measured at the end of the training iteration to approximate a separate evaluation procedure $\text{EVAL}(\theta)$.

Agents that belong to the highest-performing subset $\mathcal{P}_{top} \subset \mathcal{P}$ continue training without interruption. Agents in the middle 40% of the population $\mathcal{P}_{mid}$ undergo hyperapameter mutation and continue training. Underperforming agents $(\theta, p) \in \mathcal{P}_{bottom}$ are discarded and get replaced with a randomly sampled high-performing agent $(\theta^*, p^*) \sim \mathcal{P}_{top}$ with mutated copy of hyperparameters p^*. This algorithm directs computational resources towards more promising agents in $\mathcal{P}_{top}$ and explores numerous hyperparameter combinations, maximizing exploration (as 70% of the population constantly undergoes mutation).

The hyperparameter mutation scheme is kept simple: at each iteration of mutation each regular floating-point hyperparameter has a β_{mut} probability to be multiplied or divided by random number sampled from the uniform distribution $\mu \sim \mathcal{U}(\mu_{min}, \mu_{max})$ (see Algorithm 2 and Table III for details). Note that discrete hyperparameters (such as the number of PPO epochs) and parameters with limited scope (such as discount factor $0 < \gamma < 1$) require slightly different mutation rules.

Some hyperparameter mutations may temporarily lead to decreased performance, *e.g.* even relatively small discount factor mutations lead to significant changes in the distribution of TD-returns, thus decreasing the accuracy of the critic. Yet despite this temporary disadvantage these changes may lead to long-term benefits. To give agents a chance to adapt to altered parameters we temporarily pause PBT updates for this agent for $N_{adapt} = 5 \times 10^7$ steps. Similar to Jaderberg et al. [14] and Petrenko et al. [32] we also enable PBT only $N_{start} = 2 \times 10^8$ steps after the beginning of training in order to promote population diversity. This helps us prevent the situation where the entire population is filled by copies of one particularly lucky seed.

Meta-objective. One advantage of PBT is that it introduces an outer optimization loop that can meta-optimize for a final sparse scalar objective as opposed to inner RL loop which balances various dense reward components. Although meta-optimizing for N_{succ} is an obvious choice, the adaptive tolerance annealing described in Section III-C creates a complication: different agents in the population can have different current values of success tolerance ε and therefore cannot be compared directly as it is much easier to achieve high N_{succ} at looser tolerance.

To address this issue, we define our meta-optimization objective that takes both N_{succ} and ε into account:

$$\begin{cases} r_{meta} = \frac{\varepsilon_0 - \varepsilon}{\varepsilon_0 - \varepsilon^*} + 0.01 * N_{succ} & \text{if } \varepsilon > \varepsilon^* \\ r_{meta} = 1 + N_{succ} & \text{if } \varepsilon = \varepsilon^* \end{cases} \tag{5}$$

Until the target tolerance ε^* is reached, this objective is dominated by the term $0 \leq \frac{\varepsilon_0 - \varepsilon}{\varepsilon_0 - \varepsilon^*} \leq 1$ which is maximized when ε approaches ε^*. After the desired tolerance is reached the maximization of N_{succ} is prioritized.

Decentralized PBT. A trait that characterizes our implementation is a complete lack of any central orchestrator, typically found in other algorithms [32, 21]. Instead, we propose a completely *decentralized* PBT architecture (see Figure 2). In such architecture, each agent is responsible for execution of its own part of Algorithm 1, including finding its ranking in

Algorithm 1 Population-Based Training

Require: $\mathcal{P}$ (initial population, θ, p sampled randomly)
for $(\theta, p) \in \mathcal{P}$ **do** (async. and decentralized)
  while not end of training **do**
    $\theta \leftarrow \text{TRAIN}(\theta, p)$ ▷ Do RL for N_{iter} steps
    $r_{meta} \leftarrow \text{EVAL}(\theta)$
    $\mathcal{P}_{top}, \mathcal{P}_{mid}, \mathcal{P}_{bottom} \subset \mathcal{P}$ ▷ Sort $\mathcal{P}$ according to r_{meta}
    $(\theta^*, p^*) \sim \mathcal{P}_{top}$ ▷ Get agent from top 30%
    if $(\theta, p) \in \mathcal{P}_{bottom}$ **then**
      $p \leftarrow \text{MUTATE}(p^*)$
      $\theta \leftarrow \theta^*$ ▷ Replace weights
    else if $(\theta, p) \in \mathcal{P}_{mid}$ **then**
      $p \leftarrow \text{MUTATE}(p)$
    end if
  end while
end for
return $\theta_{best} \in \mathcal{P}$ ▷ Agent with the highest r_{meta}

Algorithm 2 Hyperparameter mutation subroutine.

function MUTATE(p) ▷ Hyperparameter vector p
  for $j \leftarrow 1$ to $len(p)$ **do**
    $z' \sim \mathcal{U}(0, 1)$
    if $z_{mut} < \beta_{mut}$ **then** ▷ Mutation probability
      $\mu \sim \mathcal{U}(\mu_{min}, \mu_{max})$ ▷ Mutation amount
      $z_{dir} \sim \mathcal{U}(0, 1)$ ▷ Mutation direction
      if $z_{dir} < 0.5$ **then**
        $p_j \leftarrow p_j * \mu$
      else
        $p_j \leftarrow p_j / \mu$
      end if
    end if
  end for
  return p
end function

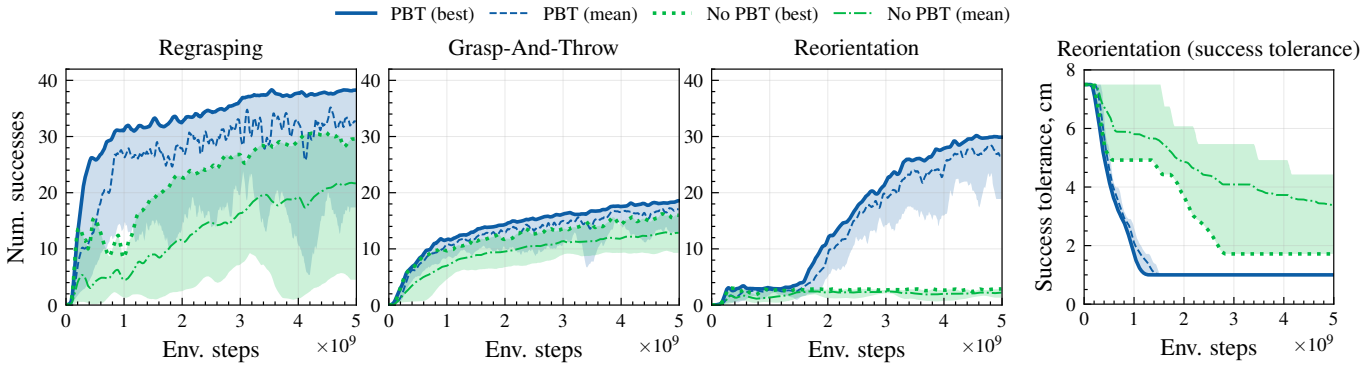

Fig. 4: Training curves with and without PBT for **single-arm + hand tasks**. Shaded area is between the best and the worst policy among 8 agents in $\mathcal{P}$ or 8 seeds in non-PBT experiments.

$\mathcal{P}$, mutating its own hyperparameters p, or replacing self with a copy of another agent.

In our implementation, agents in $\mathcal{P}$ interact exclusively through low-bandwidth access to a shared network directory containing histories of agent checkpoints and performance metrics for each agent. This eliminates the need for any other form of network communication or message passing between instances and makes it exceptionally easy to deploy the system, as most popular clusters provide some kind of shared filesystem. Our approach allows us to run exactly the same code with NFSv4 on Slurm, sshfs on NGC, S3 on AWS, or local filesystem on a multi-GPU server, provided that a shared workspace folder is mounted and accessible via a predefined path.

The lack of any central controller not only removes a point of failure, but also allows training in a volatile compute environment such as a contested cluster where some jobs can remain in queue for a long time. In this case, agents that start training later will be at a disadvantage when compared to other members of $\mathcal{P}$ that started earlier. To allow such agents to contribute to the population, we compare their performance only to historical checkpoints (of more advanced agents) that correspond to the same amount of collected experience.

It is almost inevitable that some learners may be lost during training, *e.g.* due to random hardware issues. We require no special treatment for such agents and simply use the evaluation result from the latest available checkpoint. Although reduced number of active agents will almost certainly hinder the exploration capabilities of the algorithm, empirically we observe that decentralized PBT is quite robust to loss of one or a few agents, especially with larger population sizes. Alternatively, is is also possible to detect agent disconnects in a decentralized way (*e.g.* by the lack of recent updates) and exclude such agents from the selection process, effectively reducing $|\mathcal{P}|$.

IV. Experiments

We conduct our experiments using instances with 8 CPU cores and a single Nvidia V100 GPU with 16 Gb of VRAM. Using the Isaac Gym engine [24] we are able to simulate 8192 parallel environments on each GPU. Combined with a GPU-based vectorized RL implementation rl_games [23], this allows us to reach training throughput of 5×10^4 samples per second for single-arm tasks and 3.5×10^4 samples per second for dual-arm scenarios. At this rate, to train successful policies on 5×10^9 environment transitions, it takes 30 hours for single- and 40 hours for dual-arm tasks (PBT and non-PBT training having almost equivalent per-node throughput).

For non-PBT experiments we simply train policies starting from multiple random seeds, each trained on a single instance. In our PBT experiments we use $|\mathcal{P}|$ separate 1-GPU instances that exchange information using low-bandwidth access to a shared directory (in our case, an NFS/sshfs shared folder on a Slurm/NGC cluster). Table II lists RL parameters and reward shaping coefficients used in our experiments.

Figures 4 and 5 demonstrate performance of PBT with $|\mathcal{P}| = 8$ compared to regular PPO. We use equivalent amount of compute in both cases and compare the best agent in the population with the best agent among 8 independent PPO runs. We find that PBT improves training substantially in all scenarios, and in three of them PBT becomes the enabling factor allowing the algorithm to reach non-trivial performance. For example, in single-arm reorientation task, without PBT none of the eight single-GPU training sessions reached the

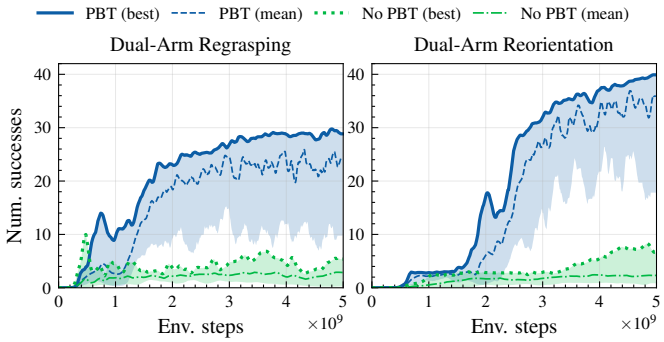

Fig. 5: Training curves with and without PBT for **dual-arm + hand tasks**. Shaded area is between the best and the worst policy among 8 agents in $\mathcal{P}$ or 8 seeds in non-PBT experiments.

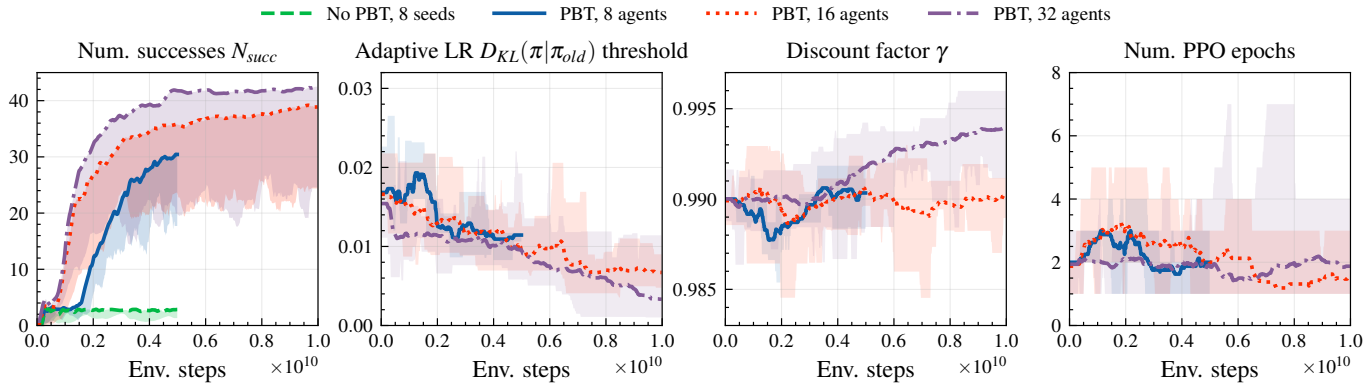

Fig. 6: Extended PBT experiments on Single-Arm Reorientation task with $|\mathcal{P}| = 16$ and $|\mathcal{P}| = 32$ agents, trained on 10 billion environment steps. 8-agent PBT run and non-PBT PPO (see Figure 4) trained to 5 billion steps superimposed for comparison. Leftmost plot shows the performance of the best agent in $\mathcal{P}$ and the shaded area is between the best and the worst policy according to N_{succ}. Additional plots show hyperparameter schedules discovered by PBT: adaptive learning rate KL-divergence threshold (Section III-C), RL discount factor γ, and the number of PPO epochs per training iteration (see Table II). We highlight the mean population value and shade the area between the minimum and the maximum value of the hyperparameter in $\mathcal{P}$.

target tolerance ε^* (rightmost plot in Figure 4). Of the three types of scenarios, reorientation relies on exploration the most: finding the correct approach to in-hand manipulation is essential, and there are many local optima to get stuck in. PBT excels at overcoming these challenges. It greatly amplifies the exploration capabilities of RL by directing computational resources towards promising agents and trying multiple combinations of RL hyperparameters and reward shaping coefficients.

Our learning approach scales well even to dual-arm tasks, despite the significantly increased overall complexity and exploration challenges. Moreover, PBT dual-arm agent (Figure 5) performed better in the reorientation task, reaching almost 40 out of $N_{max} = 50$ successes. The dual-arm task requires the agent to constantly pass the object from one hand to another, as a result the policy became more confident at juggling and tossing the object in-hand, while single-handed agents tend to rely on more conservative in-hand rotations.

To test the scaling properties of the algorithm we train populations of 16 and 32 agents on the single-arm reorientation task, allowing each agent to observe 10^{10} environment transitions (the total amount of collected experience in the $|\mathcal{P}| = 32$ experiment is thus 0.32 trillion environment steps). We observe that each increase in population size leads to significant improvement of both convergence speed and final agent performance, reaching $N_{succ} > 42$ for the best agent (see Figure 6).

Figure 6 additionally shows some hyperparameter schedules discovered by PBT. Evidently, meta-optimization tends to prefer smaller policy updates as training progresses, tightening the adaptive learning rate KL-divergence threshold (Section III-C). We hypothesize that high-performing policies become quite sensitive to large SGD steps, and such schedule helps enforce the trust region and prevent destructive parameter updates.

Curiously, only our largest experiment ($|\mathcal{P}| = 32$) was able to find solutions with higher γ. The highest-performing agent in this experiment employs an alternative strategy: for certain reorientations it chooses to place the object back on the table, rotate it, and then pick up again and put the object into the target position. While this approach is slower, it minimizes the risk of dropping the object and leads to better long-term outcomes. We demonstrate this and other learned behaviors in supplementary videos.

V. Conclusions

In this paper, we demonstrate the ability of end-to-end deep RL to learn control policies for sophisticated dexterous manipulators. We employ Population-Based Training at scale to train agents that are able to control high-DoF simulated robotic systems in contact-rich conditions. Although our scenarios only cover a small part of the robotic dexterity domain, we consider solving these object reposing challenges to be an important step on the path toward real-world deployment of robotic systems with human-level object manipulation capabilities.

While our agents demonstrate strong performance in simulation, many additional obstacles need to be overcome before practical applications are feasible. Our policies demonstrate aggressive control on the limits of robot capabilities which can lead to equipment damage in the real world. One promising approach is to utilize Riemannian Motion Policies [19] or Geometric Fabrics [40] to improve safety on the real robot by imposing conservative motion priors.

One of the most important future research directions is closing the sim-to-real gap. One approach that has been shown to improve sim-to-real transfer is the randomization of physical parameters during training. Projects such as Dactyl [2, 31] have demonstrated that training policies with domain randomization is particularly challenging and can require substantial computational budget. Our approach, similar to DeXtreme [10], based on parallelized physics simulation and high-throughput GPU-accelerated learning has the potential to significantly reduce computational requirements for particularly challenging experiments involving dual arm and hand systems and make them accessible to a wider research community.

References

[1] Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring dexterous manipulation from GPU simulation to a remote real-world trifinger. *CoRR*, abs/2108.09779, 2021. URL https://arxiv.org/abs/2108.09779. 2, 3

[2] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *Int. J. Robotics Res.*, 39(1), 2020. doi: 10.1177/0278364919887447. URL https://doi.org/10.1177/0278364919887447. 1, 2, 3, 4, 8

[3] Dhruv Batra, Angel X. Chang, Sonia Chernova, Andrew J. Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied AI. *CoRR*, abs/2011.01975, 2020. URL https://arxiv.org/abs/2011.01975. 2

[4] Sumeet Batra, Zhehui Huang, Aleksei Petrenko, Tushar Kumar, Artem Molchanov, and Gaurav S. Sukhatme. Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Conference on Robot Learning, 8-11 November 2021, London, UK*, volume 164 of *Proceedings of Machine Learning Research*, pages 576–586. PMLR, 2021. URL https://proceedings.mlr.press/v164/batra22a.html. 1

[5] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan D. Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 8973–8979. IEEE, 2019. doi: 10.1109/ICRA.2019.8793789. URL https://doi.org/10.1109/ICRA.2019.8793789. 1

[6] Yuanpei Chen, Yaodong Yang, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuang Jiang, Stephen Marcus McAleer, Hao Dong, Zongqing Lu, and Song-Chun Zhu. Towards human-level bimanual dexterous manipulation with reinforcement learning, 2022. 2

[7] Arthur Flajolet, Claire Bizon Monroc, Karim Beguir, and Thomas Pierrot. Fast population-based reinforcement learning on a single machine. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 6533–6547. PMLR, 2022. URL https://proceedings.mlr.press/v162/flajolet22a.html. 2

[8] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - A differentiable physics engine for large scale rigid body simulation. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/d1f491a404d6854880943e5c3cd9ca25-Abstract-round1.html. 2

[9] Abhishek Gupta, Justin Yu, Tony Z. Zhao, Vikash Kumar, Aaron Rovinsky, Kelvin Xu, Thomas Devlin, and Sergey Levine. Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 6664–6671. IEEE, 2021. doi: 10.1109/ICRA48506.2021.9561384. URL https://doi.org/10.1109/ICRA48506.2021.9561384. 2

[10] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. DeXtreme: Transfer of agile in-hand manipulation from simulation to reality. In *ICRA*, 2023. URL https://arxiv.org/abs/2210.13702. 1, 2, 3, 5, 8

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735. 4

[12] Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-contact iteration method for solving contact dynamics. *IEEE Robotics and Automation Letters*, 3(2):895–902, 2018. URL www.raisim.com. 2

[13] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. URL http://arxiv.org/abs/1711.09846. 1, 2, 5

[14] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio García Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *CoRR*, abs/1807.01281, 2018. URL http://arxiv.org/abs/1807.01281. 2, 6

[15] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement

learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL http://arxiv.org/abs/1806.10293. 2

[16] Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In Danica Kragic, Antonio Bicchi, and Alessandro De Luca, editors, *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 378–383. IEEE, 2016. doi: 10.1109/ICRA.2016.7487156. URL https://doi.org/10.1109/ICRA.2016.7487156. 2

[17] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Sci. Robotics*, 5(47): 5986, 2020. doi: 10.1126/scirobotics.abc5986. URL https://doi.org/10.1126/scirobotics.abc5986. 1, 2

[18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016. URL http://jmlr.org/papers/v17/15-522.html. 2

[19] Anqi Li, Ching-An Cheng, Muhammad Asif Rana, Man Xie, Karl Van Wyk, Nathan D. Ratliff, and Byron Boots. RMP2: A structured composable policy class for robot learning. In Dylan A. Shell, Marc Toussaint, and M. Ani Hsieh, editors, *Robotics: Science and Systems XVII, Virtual Event, July 12-16, 2021*, 2021. doi: 10.15607/RSS.2021.XVII.092. URL https://doi.org/10.15607/RSS.2021.XVII.092. 8

[20] Zexiang Li, Ping Hsu, and Shankar Sastry. Grasping and coordinated manipulation by a multifingered robot hand. *The International Journal of Robotics Research*, 8(4):33–50, 1989. doi: 10.1177/027836498900800402. URL https://doi.org/10.1177/027836498900800402. 2

[21] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *CoRR*, abs/1807.05118, 2018. URL http://arxiv.org/abs/1807.05118. 6

[22] Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, S. M. Ali Eslami, Daniel Hennes, Wojciech M. Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, Noah Y. Siegel, Leonard Hasenclever, Luke Marris, Saran Tunyasuvunakool, H. Francis Song, Markus Wulfmeier, Paul Muller, Tuomas Haarnoja, Brendan D. Tracey, Karl Tuyls, Thore Graepel, and Nicolas Heess. From motor control to team play in simulated humanoid football. *Sci. Robotics*, 7(69), 2022. doi: 10.1126/scirobotics.abo0235. URL https://doi.org/10.1126/scirobotics.abo0235. 1

[23] Denys Makoviichuk and Viktor Makoviychuk. RL Games, 2021. URL https://github.com/Denys88/rl_games/. 4, 5, 7

[24] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470, 2021. URL https://arxiv.org/abs/2108.10470. 1, 2, 4, 7

[25] Matthew T. Mason and J. Kenneth Salisbury. *Robot Hands and the Mechanics of Manipulation*. MIT Press, Cambridge, MA, USA, 1985. ISBN 0262132052. 2

[26] Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Sci. Robotics*, 4 (26), 2019. doi: 10.1126/scirobotics.aau4984. URL https://doi.org/10.1126/scirobotics.aau4984. 1, 2

[27] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Sci. Robotics*, 7(62), 2022. doi: 10.1126/scirobotics.abk2822. URL https://doi.org/10.1126/scirobotics.abk2822. 1

[28] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A. Preiss, Nora Ayanian, and Gaurav 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 2019, Macau, SAR, China, November 3-8, 2019*, pages 59–66. IEEE, 2019. doi: 10.1109/IROS40897.2019.8967695. URL https://doi.org/10.1109/IROS40897.2019.8967695. 1

[29] Yashraj S. Narang, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Ádám Moravánszky, Gavriel State, Michelle Lu, Ankur Handa, and Dieter Fox. Factory: Fast contact for robotic assembly. *CoRR*, abs/2205.03532, 2022. doi: 10.48550/arXiv.2205.03532. URL https://doi.org/10.48550/arXiv.2205.03532. 1

[30] Allison M. Okamura, Niels Smaby, and Mark R. Cutkosky. An overview of dexterous manipulation. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000, April 24-28, 2000, San Francisco, CA, USA*, pages 255–262. IEEE, 2000. doi: 10.1109/ROBOT.2000.844067. URL https://doi.org/10.1109/ROBOT.2000.844067. 3

[31] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *arXiv preprint*, 2019. 1, 2, 3, 4, 5, 8

[32] Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav S. Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 FPS with asynchronous reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7652–7662. PMLR, 2020. URL http://proceedings.mlr.press/v119/petrenko20a.html. 2, 6

[33] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively

parallel deep reinforcement learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 91–100. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/rudin22a.html. 2

[34] J. Kenneth Salisbury and John J. Craig. Articulated hands: Force control and kinematic issues. *The International Journal of Robotics Research*, 1(1):4–17, 1982. doi: 10.1177/027836498200100102. URL https://doi.org/10.1177/027836498200100102. 2

[35] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438. 4, 5

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. 4

[37] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*, pages 1205–1212. IEEE, 2021. doi: 10.1109/IROS51168.2021.9636053. URL https://doi.org/10.1109/IROS51168.2021.9636053. 1

[38] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. 2

[39] Xingchen Wan, Cong Lu, Jack Parker-Holder, Philip J. Ball, Vu Nguyen, Binxin Ru, and Michael Osborne. Bayesian generational population-based training. In *First Conference on Automated Machine Learning (Main Track)*, 2022. URL https://openreview.net/forum?id=HW4-ZaHUg5. 2

[40] Karl Van Wyk, Mandy Xie, Anqi Li, Muhammad Asif Rana, Buck Babich, Bryan Peele, Qian Wan, Iretiayo Akinola, Balakumar Sundaralingam, Dieter Fox, Byron Boots, and Nathan D. Ratliff. Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior. *IEEE Robotics Autom. Lett.*, 7(2):3202–3209, 2022. doi: 10.1109/LRA.2022.3143311. URL https://doi.org/10.1109/LRA.2022.3143311. 8

[41] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*, 2022. 2