2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

DeXtreme: Transfer of Agile In-hand Manipulation from Simulation to Reality

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, Gavriel State

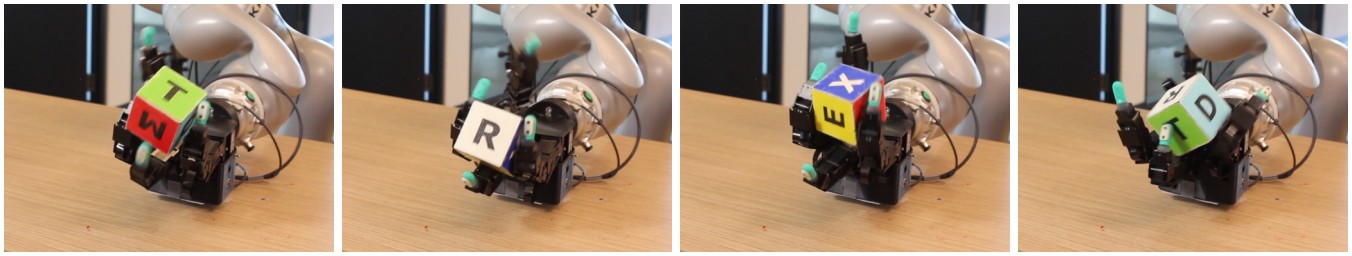


Fig. 1: The DeXtreme system using an Allegro Hand in action in the real world.

***Abstract*— Recent work has demonstrated the ability of deep reinforcement learning (RL) algorithms to learn complex robotic behaviours in simulation, including in the domain of multi-fingered manipulation. However, such models can be challenging to transfer to the real world due to the gap between simulation and reality. In this paper, we present our techniques to train a) a policy that can perform robust dexterous manipulation on an anthropomorphic robot hand and b) a robust pose estimator suitable for providing reliable real-time information on the state of the object being manipulated. Our policies are trained to adapt to a wide range of conditions in simulation. Consequently, our vision-based policies significantly outperform the best vision policies in the literature on the same reorientation task and are competitive with policies that are given privileged state information via motion capture systems. Our work reaffirms the possibilities of sim-to-real transfer for dexterous manipulation in diverse kinds of hardware and simulator setups, and in our case, with the Allegro Hand and Isaac Gym GPU-based simulation. Furthermore, it opens up possibilities for researchers to achieve such results with commonly-available, affordable robot hands and cameras. Videos of the resulting policy and supplementary information, including experiments and demos, can be found on the website.**

I. Introduction

Multi-fingered robotic hands offer an exciting platform to develop and enable human-level dexterity. Not only do they provide kinematic redundancy for stable grasps, but they also enable the repertoire of skills needed to interact with a wide range of day-to-day objects. However, controlling such high-DoF end-effectors has remained a challenging endeavour, and it is not surprising that, even in research, most robotic systems today use parallel-jaw grippers.

In 2018, OpenAI et al. [1] showed for the first time that multi-fingered hands with a purely end-to-end deep-RL based approach could endow robots with unprecedented capabilities for challenging contact-rich in-hand manipulation. However, due to the complexity of their training architecture as well as the *sui generis* nature of their work on sim-to-real transfer, reproducing and building upon their success has proven to be a challenge for the community. Recent advancements in in-hand manipulation with RL have made progress with multiple objects and a pronated hand [2], but those results have only been in simulation.

While the NLP and computer vision communities have reproduced and extended the successes of large-scale models like GPT-3 [3] and DALL-E [4, 5] respectively, similar efforts have remained elusive in robotics due to hardware and infrastructure challenges. Using large-scale data from simulations may provide avenues to unlock a similar step function in robotics capabilities.

This paper builds on top of the prior work in [1]. We use a comparatively affordable Allegro Hand with a locked wrist and four fingers, using only position encoders on servo motors; the Shadow Hand used in OpenAI's experiments costs an order of magnitude more than the Allegro Hand. We also develop a simple vision system that requires no specialised tracking or infrastructure on the hand; the system works on three off-the-shelf RGB cameras compared to OpenAI's expensive marker-based setup, making our system easily accessible for everyone. Furthermore, we use the GPU-based Isaac Gym physics simulator [6] as opposed to the CPU-based MuJoCo [7], which allows us to reduce the amount of computational resources used and the complexity of the training infrastructure. Our infrastructure only requires 8 NVIDIA A40 GPUs, as opposed to OpenAI's use of a CPU cluster composed of 400 servers with 32 CPU-cores each, as well as 32 NVIDIA V100 GPUs [8] (compute requirements for block reorientation). Our more affordable hand, in combination with the simple vision system architecture and accessible compute, dramatically simplifies the process of developing and deploying agile and dexterous manipulation. We summarise our contributions below:

- We demonstrate a system for learning-based dexterous in-hand manipulation that uses low-cost hardware (one order of magnitude less expensive than [1]), uses a purely vision-based pipeline, sets more diverse pose targets, uses orders-of-magnitude cheaper compute, and offers more insights into this problem with detailed ablations.
- While not directly comparable to [1] due to different hardware, our purely vision-based state estimation results not only outperform their best vision-based results, but also fare comparably to their marker-based results.

979-8-3503-2365-8/23/$31.00 ©2023 IEEE

Additionally, while they use discrete actions, we find that a continuous action space provides better performance in our settings.
- We will also release both our vision and RL pipelines for reproducibility. We seek to provide a much broader segment of the research community with access to a novel state-of-the-art in-hand manipulation system in hopes of catalyzing further studies and advances.

II. Method

A. Task

We propose a method for performing object reorientation on an anthropomorphic hand. Initially the object to be manipulated is placed on the palm of the hand and a random target orientation is sampled in $SO(3)$[1]. The policy then orchestrates the motion of the fingers so as to bring the object to its desired target orientation. Similar to OpenAI et al. [1], if the object orientation is within a specified threshold of 0.4 radians of the target orientation, we sample a new target orientation. The fingers continue from the current configuration and aim to move the object to its new target orientation. The success criteria is the number of consecutive target orientations achieved without dropping the object or having the object stuck in the same configuration for more than 80 seconds. Importantly, each consecutive success becomes increasingly harder to achieve as the fingers have to keep the object in the hand without dropping, hence testing the policy's ability to model the dynamics on the go.

B. Hardware

Our hardware setup (see Fig 2) consists of an Allegro Hand rigidly mounted at the wrist. We use 3 Intel D415 cameras for object tracking with RGB frames. The cameras are extrinsically calibrated relative to the palm link of the hand. Our object tracking is done entirely using a vision-based system, and in contrast to [1], we do not use any marker-based system to track the cube or fingertip states. The object we learn to manipulate is a 6.5 cm cuboid with coloured and lettered stickers on the sides. These stickers allow the vision system to distinguish between different faces (see Sec. II-G). The pose of the cube is represented with respect to the palm of the robot hand. The camera-camera extrinsics and camera-robot calibration allow us to transform the cube pose from the canonical reference frame of a camera to the palm. Since the cube is represented locally in the palm reference frame, the policy performance is not dependent on the physical location of the setup, enabling us to move the setup freely whenever desired.

C. Policy Learning with RL

RL Formulation: The task of manipulating the cube to the desired orientation is modelled as a sequential decision making problem where the agent interacts with the environment in order to maximise the sum of discounted rewards. In our case, we formulate it as a discrete-time, partially observable Markov Decision Process (POMDP). We use Proximal Policy Optimisation (PPO) [9] to learn a parametric stochastic policy π_θ (actor), mapping from observations $o \in \mathcal{O}$ to actions $a \in \mathcal{A}$. PPO additionally learns a function $V^\pi_\phi(s)$ (critic) to approximate the on-policy value function. Following Pinto et al. [10], the critic does not take in the same observations as the actor, but rather it receives additional observations including states $s \in \mathcal{S}$ in the POMDP. The actor and critic observations are detailed in Table I.

[1]In contrast to previous works [1, 8], which limited it to configurations with flat faces pointing upwards.

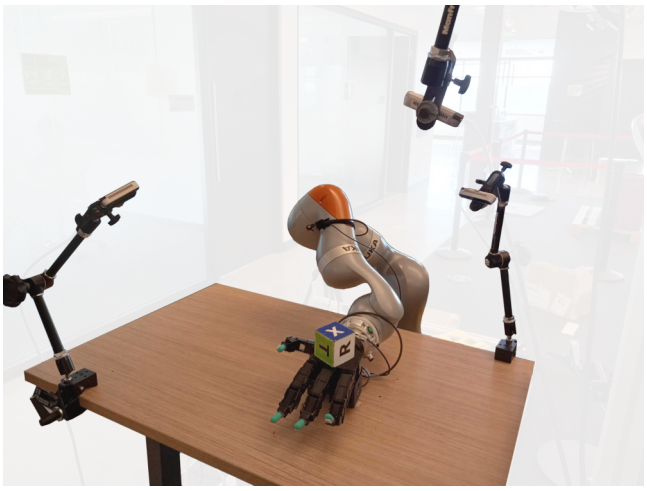


Fig. 2: The hardware setup used in this work, unlike previous work [1], is not housed in a cage, and our system is robust enough to perform the task in an open laboratory environment. The background in the image is alpha-blended for visibility.

Input	Dimensionality	Actor	Critic
Object Position	3D	✓	✓
Object Orientation	4D (Quaternion)	✓	✓
Target Position	3D	✓	✓
Target Orientation	4D (Quaternion)	✓	✓
Relative Target Orientation	4D (Quaternion)	✓	✓
Last Actions	16D	✓	✓
Hand Joints Angles	16D	✓	✓
Stochastic Delays	4D	×	✓
Fingertip Positions	12D	×	✓
Fingertip Rotations	16D (Quaternions)	×	✓
Fingertip Velocities	24D	×	✓
Fingertip Forces and Torques	24D	×	✓
Hand Joints Velocities	16D	×	✓
Hand Joints Generalised Forces	16D	×	✓
Object Scale, Mass, Friction	3D	×	✓
Object Linear Velocity	3D	×	✓
Object Angular Velocity	3D	×	✓
Object Position with Noise	3D	×	✓
Object Rotation with Noise	4D	×	✓
Random Forces on Object	3D	×	✓
Domain Randomisation Params	78D	×	✓
Gravity Vector	3D	×	✓
Rotation Distances	2D	×	✓
Hand Scale	1D	×	✓

TABLE I: Observations of the policy and value networks. The input vector is 50D in size for policy and 265D for the value function.

We use a high-performance PPO implementation from rl-games [11] with the following hyper-parameters: discount factor[2] γ=0.998, clipping ϵ=0.2. While in some experiments the learning rate was updated adaptively based on a fixed KL threshold 0.016, our best result was obtained using linear

[2]We found that following [1] and setting the higher discount of 0.998 (as opposed to 0.99 as used with MLPs [6]) was essential to allowing us to train LSTMs.

scheduling of the learning rate for the policy (start value $lr = 1e{-}4$) and a fixed learning rate for the value function ($lr = 5e{-}5$). Our best policy $\pi_\theta : \mathcal{O} \times \mathcal{H} \rightarrow \mathcal{A}$ was a Long Short-Term Memory (LSTM) network [12] taking in environment observations o and previous hidden state $h \in \mathcal{H}$. We use an LSTM backpropagation through time (BPTT) truncation length of 16. The LSTM has 1024 hidden units with layer normalization and is followed by 2 multilayer perceptron (MLP) layers with sizes 512 and ELU activation [13]. The action space $\mathcal{A}$ of our policy is the PD controller target for each of the 16 joints on the robot hand. The value function LSTM layer has 2048 hidden units with layer normalization as well, followed by 2 MLP layers with 1024 and 512 units and ELU activation. The output of the policy is low-pass filtered with an exponential moving average (EMA) smoothing factor. During training this factor is annealed from 0.2 to 0.15. Our best results in the real world were obtained with an EMA of 0.1, which provided a balance between agility and stability of the motion, preventing the hardware from breaking or burning motor cables.

D. Reward Formulation

The reward formulation is given in Table II.

Reward	Formula	Weight
Rotation Close to Goal	$1/(\mathsf{d}+0.1)$	1.0
Position Close to Fixed Target	$\|\|\mathsf{p}_{\mathsf{object}} - \mathsf{p}_{\mathsf{goal}}\|\|$	-10.0
Action Penalty	$\|\|a\|\|^2$	-0.001
Action Delta Penalty	$\|\|\mathsf{targ}_{\mathsf{curr}} - \mathsf{targ}_{\mathsf{prev}}\|\|^2$	-0.25
Joint Velocity Penalty	$\|\|\mathsf{v}_{\mathsf{joints}}\|\|^2$	-0.003
Reset Reward	**Condition**	**Value**
Reach Goal Bonus	$\mathsf{d} < 0.1$	250.0

TABLE II: Reward terms are computed, multiplied by their weight, and summed to produce the reward at each timestep. d represents the rotational distance from the object's current to the target orientation. $\mathsf{p}_{\mathsf{object}}$ and $\mathsf{p}_{\mathsf{goal}}$ are the position of the object and goal respectively. a is the current action. $\mathsf{targ}_{\mathsf{curr}}$ and $\mathsf{targ}_{\mathsf{prev}}$ are the current and previous joint position targets. $\mathsf{v}_{\mathsf{joints}}$ is the current joint velocity vector.

E. Simulation

Our aim in this paper is to learn dexterous manipulation behaviours. Current on-policy learning algorithms can struggle to accomplish this on real robots due to the number of samples required. Hence, we learn our behaviours entirely in simulation. We use the GPU-based Isaac Gym physics simulator [6], which models contacts differently than MuJoCo's soft-contact model [7] used in [1]. Isaac Gym gives the advantage of being able to simulate thousands of robots in parallel on a single GPU, mitigating the need for large amounts of CPU resources.

F. Domain Randomisation

It is widely known that there is a "sim-to-real" gap between physics simulators and real life [14]. Compounding this is the fact that the robot as a system can change from day to day (*e.g.*, due to wear-and-tear) and even from timestep to timestep (*e.g.*, stochastic noise). To help overcome this, we introduce various kinds of randomisations [15] into the simulated environment as listed in Table III.

Parameter	Type	Distribution	Initial Range	ADR-Discovered Range
Hand				
Mass	Scaling	uniform	[0.4, 1.5]	[0.4, 1.5]
Scale	Scaling	uniform	[0.95, 1.05]	[0.95, 1.05]
Friction	Scaling	uniform	[0.8, 1.2]	[0.54, 1.58]
Armature	Scaling	uniform	[0.8, 1.02]	[0.31, 1.24]
Effort	Scaling	uniform	[0.9, 1.1]	[0.9, 2.49]
Joint Stiffness	Scaling	loguniform	[0.3, 3.0]	[0.3, 3.52]
Joint Damping	Scaling	loguniform	[0.75, 1.5]	[0.43, 1.6]
Restitution	Additive	uniform	[0.0, 0.4]	[0.0, 0.4]
Object				
Mass	Scaling	uniform	[0.4, 1.6]	[0.4, 1.6]
Friction	Scaling	uniform	[0.3, 0.9]	[0.01, 1.60]
Scale	Scaling	uniform	[0.95, 1.05]	[0.95, 1.05]
External Forces	Additive	Refer to [1]	–	–
Restitution	Additive	uniform	[0.0, 0.4]	[0.0, 0.4]
Observation				
Obj. Pose Delay Prob.	Set Value	uniform	[0.0, 0.05]	[0.0, 0.47]
Obj. Pose Freq.	Set Value	uniform	[1.0, 1.0]	[1.0, 6.0]
Obs Corr. Noise	Additive	gaussian	[0.0, 0.04]	[0.0, 0.12]
Obs Uncorr. Noise	Additive	gaussian	[0.0, 0.04]	[0.0, 0.14]
Random Pose Injection	Set Value	uniform	[0.3, 0.3]	[0.3, 0.3]
Action				
Action Delay Prob.	Set Value	uniform	[0.0, 0.05]	[0.0, 0.31]
Action Latency	Set Value	uniform	[0.0, 0.0]	[0.0, 1.5]
Action Corr. Noise	Additive	gaussian	[0.0, 0.04]	[0.0, 0.32]
Action Uncorr. Noise	Additive	gaussian	[0.0, 0.04]	[0.0, 0.48]
RNA α	Set Value	uniform	[0.0, 0.0]	[0.0, 0.16]
Environment				
Gravity (each coord.)	Additive	normal	[0, 0.5]	[0, 0.5]

TABLE III: Domain randomisation parameter ranges for policy learning.

Automatic Domain Randomisation: In our best policies, we set the parameters of the domain randomisations via a vectorised implementation of Automatic Domain Randomisation (ADR, introduced in OpenAI et al. [8]). ADR allows us to train policies with enough randomisation for behaviour exploration early in training while producing final policies that are robust to the largest range of environment conditions possible at the end of training. The range of randomisations for each environment parameter in ADR is modelled as a uniform distribution $\phi \sim U(a, b)$, where a and b represent the upper and lower bounds of that randomisation dimension. Environments sample each dimension of their environment parameters from these bounds. Some percentage (in our case, 40%) of the vectorised environments are dedicated to evaluation. In these environments, one of the environment dimensions is sampled from the lower or upper boundary; for each dimension, the number of consecutive successes on that dimension is measured. For example, if the average consecutive successes in the $n = 256$ queue on a particular boundary exceeds the upper threshold $t_H = 20$, the range is widened with a lower bound $a \leftarrow a - \Delta$. If the performance is below the lower threshold $t_L = 5$, then the range is tightened on that bound. Note that performance on the upper and lower boundaries on each dimension is measured separately. Unlike in [8], we choose the size of step Δ separately for each dimension. When training on multi-GPUs (8 for our best policies), we ran ADR separately on each one. This was done for two reasons: firstly, to avoid additional synchronisation overhead of buffers. Secondly, to partially mitigate the disadvantage of ADR caused by the failure to model the joint distribution - having multiple independent parameter sets to some extent will allow multiple sets of extreme parameters. All randomisations are set by ADR.[3]

Physics Randomisations: We apply physics randomisations to account for both changing real-world dynamics and the inevitable gaps between physics in simulation and reality. These include basic properties such as mass, friction

[3]Except for scale, which is randomised within a fixed range as collision morphologies cannot be changed at run-time.

and restitution of the hand and object. We also randomly scale the hand and object to avoid over-reliance on exact morphology. On the hand, joint stiffness, damping, and limits are randomised. Furthermore, we add random forces to the cube in a similar fashion to [1].

Non-physics Randomisations: In addition to normal physics randomisations, Table III lists action and observation randomisations, which we found to be critical to achieving good real world performance. To make our policies more robust to the changing inference frequency and jitter resulting from our ROS-based inference system, we add stochastic delays to cube pose and action delivery time as well as fixed-for-an-episode action latency. To the actions and observations, we add correlated and uncorrelated additive Gaussian noise. To account for unmodelled dynamics, we use a Random Network Adversary (RNA, see [8]), which uses a randomly generated neural network each episode to introduce much more structured noise patterns into the environment. As we are doing simulation on GPU rather than CPU, instead of using a new network per environment-episode and wasting memory on thousands of individual MLPs, we generate a single network across all environments and use a unique and periodically refreshed dropout pattern per environment. Actions from the RNA network are blended with those from the policy by $\mathbf{a} = \alpha \cdot a_{\text{RNA}} + (1 - \alpha) \cdot a_{\text{policy}}$, where α is controlled by ADR.

G. Pose Estimation

Data Generation and Processing: We use NVIDIA Omniverse Isaac Sim with Replicator[4] to generate 5M images of the cube in hand. Each image is 320×240 in resolution and contains visual domain randomisations as summarised in Table IV to add variety to the training set. Such visual domain randomisations allow the network to be robust to different camera parameters and visual effects that may be present in the scene. In addition, we apply data augmentations during training on a batch in order to add even more variety to the training set. As such, a single rendered image from the dataset can provide multiple training examples with different data augmentation settings, thereby saving both rendering time and storage space. For instance, motion blur (important in our case where we have a fast-moving object to track) can be especially time consuming at rendering time. Instead we generate it on the fly via motion-blur data augmentation by smearing the image with a Gaussian kernel with the blur direction and extent chosen randomly. The data augmentations used on the images are listed in Table V. Each augmentation is applied with a fixed probability to ensure that the batch consists of a combination of the original as well as augmented images.

We also collect configurations of the cube in-hand generated by our policies running in the real world and play them back in Isaac Sim to render data where the pose estimates are not fully reliable *i.e.* the pose estimator is not accurate all the time. This happens due to the sim-to-real gap as a result of either sparse sampling or insufficient lighting randomisations for those configurations. Playback in simulation enables dense sampling of pose and lighting around these configurations with more randomisations. This allows us to generate larger datasets that improve the reliability of the pose estimator in the real world *i.e.* closing the perception loop between real and sim by mining configurations from the current best policy in the real world and using them in sim to render more images. We use the same intrinsics of the cameras in the real world, but randomise extrinsics when rendering data.

Parameter	Probability Distribution
albedo desaturation	uniform$(0.0, 0.3)$
albedo add	uniform$(-0.2, 0.2)$
albedo brightness	uniform$(0.5, 1.0)$
diffuse tint	normal$(1.0, 0.3)$
reflection roughness constant	uniform$(0.0, 1.0)$
metallic constant	uniform$(0.0, 0.5)$
specular level	uniform$(0.0, 1.0)$
enable camera noise	bernoulli(0.3)
enable scan lines	bernoulli(0.3)
scan line spread	uniform$(0.1, 0.5)$
enable vertical lines	bernoulli(0.1)
enable film grain	bernoulli(0.2)
grain amount	uniform$(0.0, 0.15)$
grain size	uniform$(0.7, 1.2)$
enable random splotches	bernoulli(0.1)
colour amount	uniform$(0.0, 0.3)$
hand visibility	bernoulli(0.75)
camera pose	Hemispherical shell[5]
camera f/l multiplier	uniform$(0.99, 1.01)$

TABLE IV: Ranges of vision parameter randomisations.

Data Augmentation Type	Probability
CutMix (see [16])	0.5
Random Blurring	0.5
Random Background	0.6
Random Rotation	0.5
Random Brightness and Contrast	0.5
Random Cropping and Resizing	0.5

TABLE V: Various data augmentations applied to the images on the fly during training. We also set a probability for each one of them.

Training setup and inference: We use a torchvision Mask-RCNN [17]-inspired network that regresses to the bounding box, segmentation, and keypoints located at the 8 corners of the cube. The bounding box localises the cube in the image and the keypoint head regresses to the positions of the 8 corners of the cube within the bounding box. The networks are trained with cross-entropy loss for segmentation and keypoint location regression, and smooth L1 loss for bounding box regression. We use the ADAM optimiser with a learning rate of 1e-4. The network runs on three cameras at an inference rate of 20Hz on an NVIDIA RTX 3090 GPU and a 32-core AMD Ryzen Threadripper CPU. To make the pose estimate reliable for the downstream policy, we first perform classic PnP [18] on each of the three cameras independently and then filter out the ones where the projected keypoints from the PnP pose do not match the inferred keypoints from the network. We triangulate the keypoints from the filtered cameras and register them against the model of the cube to obtain the pose in a canonical reference frame. We use the OpenCV implementation of PnP and roma [19] for registering keypoints against the model of the cube. We benchmark the pose on a test set consisting of 50K images and provide results in Table VI. Since we do not use any marker based system in the real world, we can only precisely evaluate the

[4]See https://developer.nvidia.com/isaac-sim

[5]Camera pose is randomly sampled from a hemispherical shell with thickness 0.3m around the hand, with a random focus selected with a 0.2m radius around the hand origin.

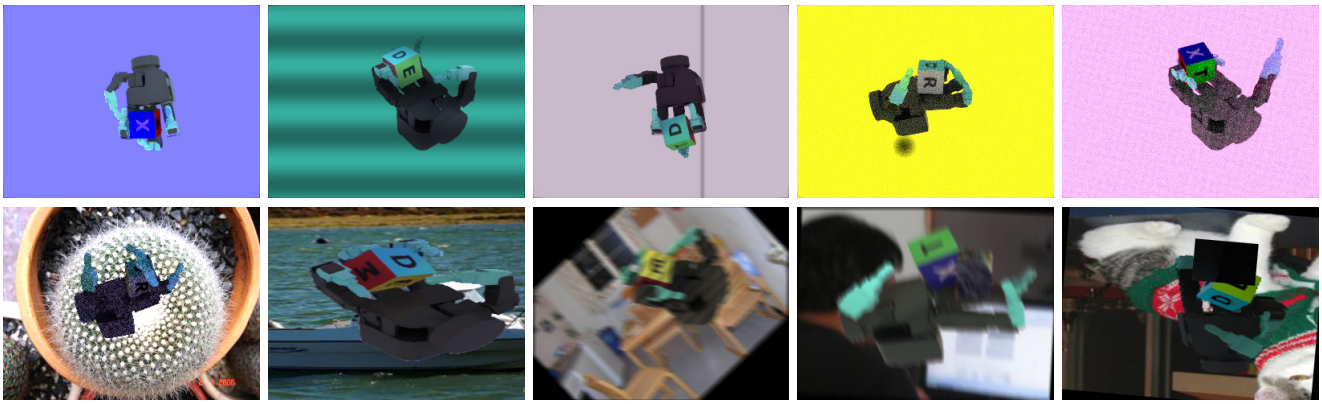


Fig. 3: Illustration of generated data. *Top row:* Images generated from NVIDIA Isaac Sim, demonstrating different camera locations, lighting conditions, and camera poses. *Bottom row:* Images after data augmentations such as CutMix, random cropping and rotation, and motion blur are applied during training.

Experiment	Avg. Rotation Error	Avg. Translation Error		
		X	Y	Z
Sim	5.3±0.11°	1.9±0.1 mm	4.1±0.2 mm	6.9±0.4 mm

TABLE VI: Rotation and translation error on test dataset with 90% confidence intervals.

performance of the pose estimator in simulation. Our ablation studies in III-B do test the strength of the pose estimator for manipulation in the real world. One important difference between our approach and OpenAI et al. [1] is that our pose estimation is not done end-to-end. Since we detect keypoints in the image and use geometric computer vision to obtain the pose, our pose estimator is not tied to a fixed camera setup as used in OpenAI et al. [1].

III. Results

In the following section, we present the results we achieved in object reorientation in the simulations and then real world using the methods described in Section II. We then follow it up with tests of policy robustness in reality and simulation.

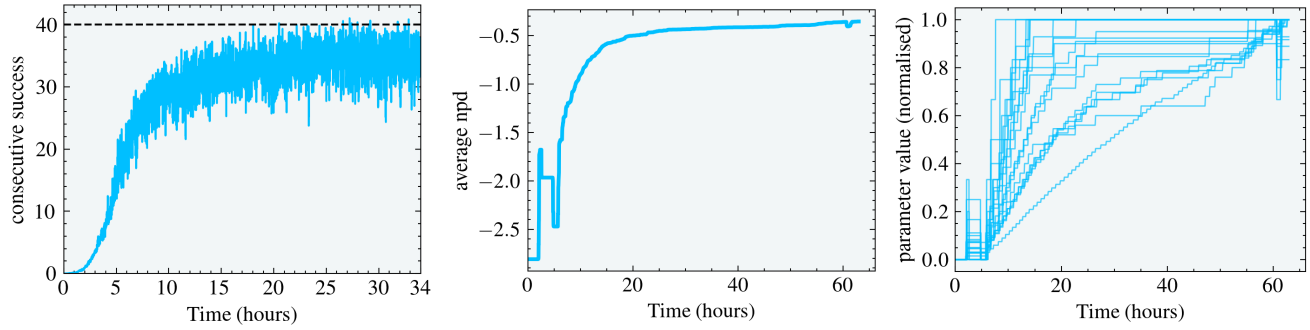
A. Training in Simulation

For all of our experiments, we use a simulation dt of $\frac{1}{60}s$ and a control dt of $\frac{1}{30}s$. We train with 16384 agents per GPU and use a goal reaching orientation threshold of 0.1 rad but test with 0.4 rad as in [1, 6] for all experiments in the real world. All policies are trained with randomisations described in Table III. Most importantly, our ADR policies using the same compute resources – 8 NVIDIA A40s – achieved the best performance in the real world after training only for 2.5 days in contrast to [8] that trained for 2 weeks to months. Various training curves for our task a) with manual DR, b) with automatic domain randomisation (ADR), and c) ADR parameter evolution are presented in Figure 4.

B. Real World Policy Performance

It is worth noting that, while in simulations, state information is derived directly from physics buffers, in all of our real-world experiments we use the pose estimator described in section II-G to track the state of the cube and provide it as input to the policy. The qualitative results of this are best illustrated by the accompanying videos at `https://dextreme.org`. As in [1], we observe a large range of different behaviours in the policies deployed on the real robot. Our real-world quantitative results measuring consecutive successes are illustrated in Table VII. **We demonstrate performance which significantly improves upon the best vision policies from [1] and is competitive with ADR policies given high-quality state information from a motion capture system in [8]**. We collect 10 trials for each policy to obtain the average consecutive successes and also collect different set of trials across different days to understand the inter-day variability that may arise due to different dynamics, temperature, and lighting conditions. We find that our policies do not show a dramatic drop in performance, indicating that they are mostly robust to inter-day variations. We believe such inter-day variations are important to benchmark in robotics [20] and have endeavoured to highlight this specifically in this challenging task. We benchmark both ADR and non-ADR (manually-tuned DR ranges) policies in Table VII and like [1] find that the policies trained with ADR perform the best, suggesting that the sheer diversity of data gleaned from the simulator endows the policies with extreme robustness needed in the real world. We find that on an average, the trials with ADR achieve more consecutive successes than non-ADR policies.

Additionally, we also benchmark our results beyond the basic experiment of goal reaching by making the policy hold the cube at a target orientation N times in a row (we use $N = 10$), *i.e.*, we count the number of times the cube orientation is within the threshold of 0.4 rad, as described in III-A, and refresh the goal only if the counter reaches N and reset the counter to zero every time it goes outside the threshold. In the basic experiment of goal reaching without the hold, the cube may shoot past the target, making it difficult to tell if the target was achieved merely due to noise in the pose estimation or if the cube orientation was indeed accurately estimated. Therefore, holding the cube for N times in a row ensures that the goal was not achieved by chance, highlighting the robustness of the pose estimator. We conduct this experiment only with our ADR policies as shown in (Table VII 2^{nd} row). It is worth noting that the policy was not trained explicitly to hold the cube and this is meant to be a test of accuracy of pose. We chose N=10 based on our simulation experiments and found that setting N too high led to a dramatic drop in the performance because the LSTM was not trained for such scenarios. On the other hand, setting it



(a) Manual DR training curve.

(b) ADR nats per dimension (npd) evolution.

(c) ADR parameter evolution.

Fig. 4: Training curve evolution for (a) manual domain randomisation, which takes ∼24 hours to reach an average of 35 consecutive successes, (b) ADR experiments showing how the average nats per distribution (npd), an indicator of the extent of randomisation, increase as a function of training and (c) shows how parameters (normalised) involved in ADR evolve over time. All curves are from experiments in simulations.

Experiment	Cons. Success Trials (sorted)	Average	Median
Best Model	1, 6, 6, 10, 10, 18, 18, 36, 61, 112	27.8 ± 19.0	14.0
	3, 4, 7, 16, 19, 22, 29, 31, 58, 77	26.6 ± 13.2	20.5
	1, 5, 5, 11, 12, 12, 33, 36, 42, 51	20.8 ± 9.8	12.0
Best Model (goal frame count=10)	6, 8, 10, 16, 16, 17, 20, 33, 39, 45	21.0 ± 7.4	16.5
	9, 11, 13, 13, 15, 16, 27, 29, 32, 36	20.1 ± 5.4	15.5
	2, 3, 3, 9, 11, 12, 14, 15, 43, 44	16.6 ± 8.4	11.5
Non-ADR Model	2, 3, 7, 7, 13, 16, 22, 23, 26, 29	14.8 ± 5.4	14.5
	1, 1, 3, 7, 8, 11, 14, 17, 22, 35	11.9 ± 5.8	9.5
	0, 7, 8, 8, 9, 10, 10, 11, 17, 20	10.0 ± 3.0	9.5

TABLE VII: The results of running different models on the real robot. We run 10 trials per policy [1] to benchmark the average consecutive successes. Individual rows within each experiment indicate running the experiment on different days [20] and ± indicates 90% confidence interval. Our best model was trained with ADR while non-ADR experiments had DR ranges manually tuned. The second experiment shows results when the cube is held at a goal for additional consecutive frames once the target cube pose is reached.

too low did not change the simulation performance; $N = 10$ was a good balance between performance and LSTM stability. This also lets us separate the drop in performance due to LSTM instability from pose estimation errors in the real world.

IV. Related Work

In-hand manipulation is a longstanding challenge in robotics. Classical methods [21, 22, 23] have focused on directly leveraging the robots' kinematic and dynamic model to analytically derive controllers for the object in hand. These approaches work well while an object maintains no-slip contacts with the hand, but struggle to achieve good results in dynamic tasks where complex sequences of making-and-breaking of contacts is needed.

Reinforcement learning has proven to be a powerful method for learning complex behaviours in robots with many degrees of freedom. Hwangbo et al. [24] and Lee et al. [25] showed how robust legged locomotion can be learned even over challenging terrain using a combination of deep RL, fast simulation, and domain randomisation. A crucial inspiration for our work is OpenAI et al. [1] on learning in-hand reorientation of cubes via reinforcement learning in simulation and subsequent OpenAI et al. [8] extending this task to solving Rubik's cubes. A variety of recent works [6, 26, 27, 28] have leveraged new RL techniques and simulators in order to reproduce or extend in simulation the anthropomorphic in-hand manipluation capabilities shown in [1]. However, these works have not shown sim-to-real transfer, demonstrating that this remains a significant challenge for learned in-hand manipulation. There have also been a variety of recent approaches attempting in-hand manipulation from the perspective of finger gaiting [29, 30]. However, these often fail to reproduce the agile dexterity present in human hands as the limitations of such a sequential approach to control place limitations on speed. Similarly, Allshire et al. [31] and Shi et al. [32] achieve real-world in hand manipulation using reinforcement learning, but on a platform that cannot mimic the dexterity of a human hand. Sievers et al. [33] learned in-hand manipulation using tactile sensing, however the lack of vision meant that the kinds of re-posing behaviour and speed of the manipulation were limited. There has also been research investigating using reinforcement learning directly on hardware platforms as opposed to in simulation [34, 35]. However, this limits the complexity of learning that can be done due to the small number of trials available on real hardware platforms, as well as the wear-and-tear imposed.

Pose estimation for robotic manipulation is a widely studied area [36, 37]. However, relatively few works outside of OpenAI et al. [1] have applied it to the problem of dense in-hand manipulation, which introduces challenges that exclude many off-the-shelf pose estimators (due to large degree of occlusions, motion blur, *etc.*).

V. Conclusion

At the heart of this work is an important message that sim-to-real for contact-rich manipulations can scale, and dexterity with hands is one of the means to show this. While large-scale industries *e.g.* automobile, aerospace, and electronics have long leveraged simulation as a fundamental development tool, the robotics community has remained skeptical of the use of simulators. The famous catchphrase, "*simulations are doomed to succeed*", captures the criticism the community has shown at times. We hope that our work and OpenAI et al. [1] provide important evidence that simulations, despite various approximations *e.g.* simplified friction model, discrete time-steps *etc.*, can endow robots with the skills required to work in challenging real-world environments.

References

[1] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, vol. abs/1808.00177, 2018. [Online]. Available: http://arxiv.org/abs/1808.00177 1, 2, 3, 4, 5, 6

[2] T. Chen, J. Xu, and P. Agrawal, "A system for general in-hand object re-orientation," *Conference on Robot Learning*, 2021. 1

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165 1

[4] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," 2021. [Online]. Available: https://arxiv.org/abs/2102.12092 1

[5] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," 2022. [Online]. Available: https://arxiv.org/abs/2204.06125 1

[6] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," *CoRR*, vol. abs/2108.10470, 2021. [Online]. Available: https://arxiv.org/abs/2108.10470 1, 2, 3, 5, 6

[7] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033. 1, 3

[8] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," *CoRR*, vol. abs/1910.07113, 2019. [Online]. Available: http://arxiv.org/abs/1910.07113 1, 2, 3, 4, 5, 6

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017. 2

[10] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1710.06542 2

[11] D. Makoviichuk and V. Makoviychuk, "rl-games: A high-performance framework for reinforcement learning," https://github.com/Denys88/rl_games, May 2022. 2

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 3

[13] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.07289 3

[14] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life*, F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, Eds., 1995. 3

[15] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810. 3

[16] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," *CoRR*, vol. abs/1905.04899, 2019. [Online]. Available: http://arxiv.org/abs/1905.04899 4

[17] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870 4

[18] Wikipedia, "Perspective n point." [Online]. Available: https://en.wikipedia.org/wiki/Perspective-n-Point 4

[19] R. Brégier, "Deep regression on manifolds: a 3d rotation case study," *CoRR*, vol. abs/2103.16317, 2021. [Online]. Available: https://arxiv.org/abs/2103.16317 4

[20] Google, "The importance of a/b testing in robotics," 2021. [Online]. Available: https://ai.googleblog.com/2021/06/the-importance-of-ab-testing-in-robotics.html 5, 6

[21] J. K. Salisbury and J. J. Craig, "Articulated hands: Force control and kinematic issues," *The International Journal of Robotics Research*, vol. 1, no. 1, pp. 4–17, 1982. [Online]. Available: https://doi.org/10.1177/027836498200100102 6

[22] M. T. Mason and J. K. Salisbury, *Robot Hands and the Mechanics of Manipulation*. Cambridge, MA, USA: MIT Press, 1985. 6

[23] Z. Li, P. Hsu, and S. Sastry, "Grasping and coordinated manipulation by a multifingered robot hand," *The International Journal of Robotics Research*, vol. 8, no. 4, pp. 33–50, 1989. [Online]. Available: https://doi.org/10.1177/027836498900800402 6

[24] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning Agile and Dynamic Motor Skills for Legged Robots," *Science Robotics*, Jan 2019. 6

[25] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, p. eabc5986, 2020. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abc5986 6

[26] T. Chen, J. Xu, and P. Agrawal, "A system for general in-hand object re-orientation," 2021. [Online]. Available: https://arxiv.org/abs/2111.03043 6

[27] W. Huang, I. Mordatch, P. Abbeel, and D. Pathak, "Generalization in dexterous manipulation via geometry-aware multi-task learning," 2021. [Online]. Available: https://arxiv.org/abs/2111.03062 6

[28] V. Caggiano, H. Wang, G. Durandau, M. Sartori, and V. Kumar, "Myosuite – a contact-rich simulation suite for musculoskeletal motor control," 2022. [Online]. Available: https://arxiv.org/abs/2205.13600 6

[29] R. Higo, Y. Yamakawa, T. Senoo, and M. Ishikawa, "Rubik's cube handling using a high-speed multi-fingered hand and a high-speed vision system," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6609–6614. 6

[30] A. S. Morgan, K. Hang, B. Wen, K. E. Bekris, and A. M. Dollar, "Complex in-hand manipulation via compliance-enabled finger gaiting and multi-modal planning," *RAL*, vol. abs/2201.07928, 2022. [Online]. Available: https://arxiv.org/abs/2201.07928 6

[31] A. Allshire, M. Mittal, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wuthrich, S. Bauer, A. Handa, and A. Garg, "Transferring Dexterous Manipulation from GPU Simulation to a Remote Real-World TriFinger," *CoRR*, 2021. 6

[32] F. Shi, T. Homberger, J. Lee, T. Miki, M. Zhao, F. Farshidian, K. Okada, M. Inaba, and M. Hutter, "Circus anymal: A quadruped learning dexterous manipulation with its limbs," 2020. 6

[33] L. Sievers, J. Pitz, and B. Bäuml, "Learning purely tactile in-hand manipulation with a torque-controlled hand," 2022. [Online]. Available: https://arxiv.org/abs/2204.03698 6

[34] V. Kumar, E. Todorov, and S. Levine, "Optimal control with

learned local models: Application to dexterous manipulation," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 378–383. 6

[35] V. Kumar, A. Gupta, E. Todorov, and S. Levine, "Learning dexterous manipulation policies from experience and imitation," *CoRR*, vol. abs/1611.05095, 2016. [Online]. Available: http://arxiv.org/abs/1611.05095 6

[36] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," in *CoRL*, 2018. 6

[37] Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic, "Cosypose: Consistent multi-view multi-object 6d pose estimation," 2020. [Online]. Available: https://arxiv.org/abs/2008.08465 6