TRANSFERRING DEXTEROUS MANIPULATION FROM GPU SIMULATION TO A REMOTE REAL-WORLD TRIFINGER

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a system for learning a challenging dexterous manipulation task involving moving a cube to an arbitrary 6-DoF pose with only 3-fingers trained with NVIDIA's IsaacGym simulator. We show empirical benefits, both in simulation and sim-to-real transfer, of using keypoints as opposed to position+quaternion representations for the object pose in 6-DoF for policy observations and in reward calculation to train a model-free reinforcement learning agent. By utilizing domain randomization strategies along with the keypoint representation of the pose of the manipulated object, we achieve a high success rate of 83% on a remote TriFinger system maintained by the organizers of the Real Robot Challenge. With the aim of assisting further research in learning in-hand manipulation, we make the codebase of our system, along with trained checkpoints that come with billions of steps of experience available, at `https://sites.google.com/view/s2r2`

1 INTRODUCTION

Through cloud-based robotic systems, Robotics as-a-Service (RaaS) promises to alleviate many of the upfront requirements to install and maintain robot hardware Kehoe et al. (2015). The advantages of this model include easier management and scalability, greater flexibility and savings in terms of compute power and utilities. While RaaS has been proposed for parallel-grid computing on demand, collective robot learning, and crowd-sourcing access to remote human expertise, a great need for such a model has emerged in the robotics community during the current pandemic. In this work, we show how large-scale simulation done on a desktop grade GPU and cloud-based robotics can enable roboticists to perform research in robotic learning with modest resources. We focus on 6-DoF object manipulation by using a dexterous multi-finger manipulator as a case study.

Dexterous manipulation requires dealing with high-dimensionality of the system, hybrid dynamics, and uncertainties about the environment Okamura et al. (2000). Prior work has trained a control policy for in-hand manipulation of a block with a Shadow Dexterous Hand OpenAI et al. (2018). This was achieved through scientific compute clusters for distributed training in simulation and access to specialized hardware (the "cage") to receive reliable state information during object interaction in the real-world. While impressive, the requirement of such an exorbitant infrastructure makes this kind of study typically non-reproducible bordering impractical, hence the paucity of results building upon this work for further research into learning manipulation.

Contrarily, Wüthrich et al. (2020) designed an open-sourced low-cost robotic platform for dexterous manipulation called *TriFinger*. They showed that the robot is suitable for deploying learned policies due to its hardware robustness and software safety checks. Building upon this work, the authors organized the 'Real Robot Challenge (RRC)' rea (2020), for which they developed a farm of TriFinger systems. For the challenge the authors provided access to PyBullet simulation of the robot Coumans et al. (2013). This challenge reinforced the inaccessibility of applying learning-based systems for such tasks: most competing teams elected to use structured policies with manually tuned high-level controllers or residual learning on top Funk et al. (2021); Chen et al. (2021); Yoneda et al. (2021).

One of the reasons for the above is the ubiquity of CPU-based simulators Coumans & Bai (2016); Todorov et al. (2012). Due to its low sample generating rate, it is highly time-consuming to tune and train a successful policy on PyBullet (a CPU based simulator) for such a complex task. The TriFinger platform is based on low-cost open-source hardware (Wüthrich et al., 2020). The TriFinger system

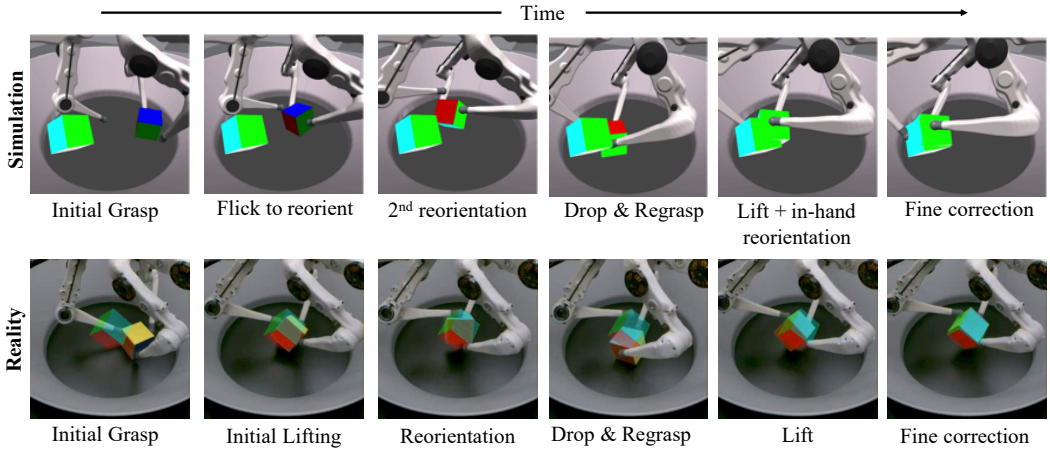


Figure 1: *Top:* Our system learns to manipulate a cube to desired 6-DoF goal poses (position+orientation) entirely in simulation. Training is done on desktop-scale compute using a GPU-based physics engine. *Bottom:* We then transfer to a real robot located thousands of kilometers away from where development work is done.

uses a vision-based tracker to triangulate the object pose. The tracker functions at a low frequency and provides noisy estimates of the cube pose, making reliable policy inference difficult. Lastly, working on the cloud-based platform Trans-Atlantic with limited access to the hardware slows down the iteration cycles. While the overall cost of the system is reduced when compared to a high-end setup like the Shadow Hand used by OpenAI, more noise and delays are present due to the commodity nature of the hardware. This makes sim-to-real transfer non-trivial.

This effort aims to overcome these limitations through a systems approach to robotics with infrastructure around a GPU-accelerated simulator coupled with a remote Trifinger system for successful sim-to-real transfer of a learned policy, as shown in Figure 1. Using NVIDIA's IsaacGym Liang et al. (2018), we train a policy to learn 6-DoF object manipulation successfully in *under a day on a single desktop-grade GPU*. While not directly comparable to the Trifinger task, this number is in contrast to previous efforts with in-hand manipulation, for example, OpenAI's distributed training infrastructure, which took several days to learn a robust policy for cube rotation on a large distributed server involving large CPU and GPU clusters. Additionally, we investigate different object pose representations for observations and rewards formulation. To allow successful sim-to-real transfer, we perform domain randomization on various physics properties in the simulator along with different noise models for observations and actions.

This paper primarily makes robot systems contributions as follows:

1. We provide a framework for learning similar in-hand manipulation tasks with sim-to-real transfer using far fewer computational resources (1 GPU & CPU) than prior work but that can also benefit from large scale training.
2. We show the benefits of using keypoints as representations of object pose and in reward computation with RL algorithms for in-hand manipulation, especially when reposing in $SE(3)$.
3. We demonstrate the ability to learn a challenging 6-DoF manipulation task, re-posing a cube, with simulation data alone and deploying on a third-party remote physical robot setup.
4. We open-source the software platform to run training in simulation and inference of the resulting policies for other researchers to build on top of.

2 Related Work

Advances in reinforcement learning (RL) algorithms and computational hardware have enabled rapid progress in using these algorithms for tasks on real robots. Techniques such as domain randomization and large-scale training have enabled results across a variety of tasks with sim2real, including in-hand manipulation (OpenAI et al., 2018; 2019), as well as in legged locomotion (Hwangbo et al., 2019; Shi et al., 2020). Active identification of system parameters has also been shown to be helpful in the context of learning manipulation tasks (Chebotar et al., 2019). However, few results have focused on learning-based control over a full 6-DoF pose in-hand. Furthermore, none of the existing systems for

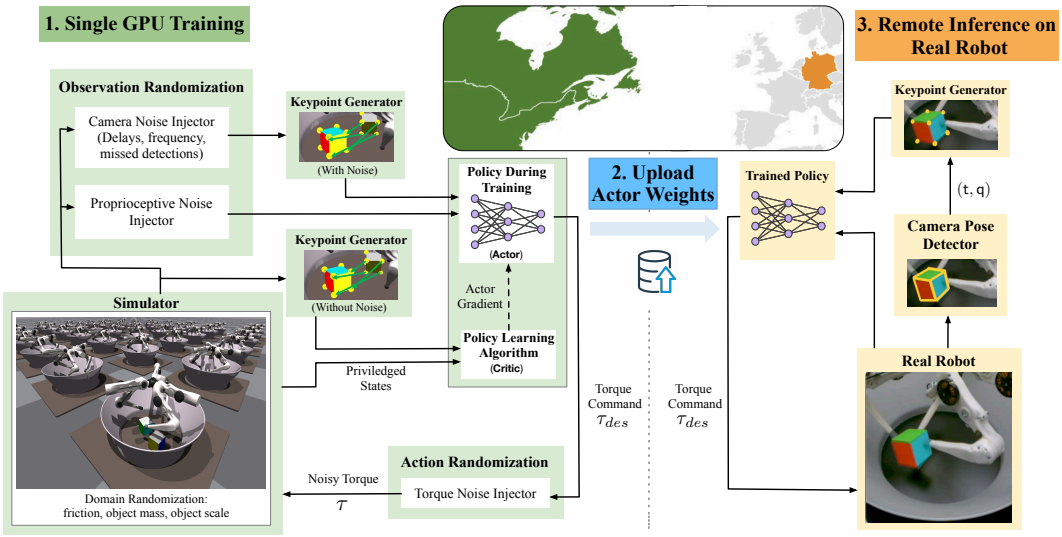


Figure 2: Our system trains using the IsaacGym simulator[1](Makoviychuk et al., 2021) on 16,384 environments in parallel on a single NVIDIA Tesla V100 or RTX 3090 GPU. Inference is then conducted remotely on a TriFinger robot located across the Atlantic in Germany using the uploaded actor weights. The infrastructure on which we perform Sim2Real transfer is provided courtesy of the organisers of the Real Robot Challenge (rea, 2020).

training in-hand manipulation at a large scale have been provided in an accessible manner for further work in robot learning to build on.

The most widely used simulators for robot learning research are MuJoCo Todorov et al. (2012) and Pybullet Coumans & Bai (2016). While both have proven to be successful for various robotics locomotion and manipulation tasks, they are often slow for complicated environments and require CPU clusters putting some limits on their scalability. Brax Freeman et al. (2021), on the other hand, supports GPU / TPU acceleration, but it comes at the cost of simplified physics simulation assumptions and simple environments. IsaacGym (Liang et al., 2018) offers high-fidelity physics modelling and GPU acceleration support. It also supports directly sharing observations and actions through GPU memory between policy network and the physics engine, massively increasing throughput. Part of our contribution is to demonstrate the viability of the approach of GPU-based simulation to sim2real for in-hand manipulation.

3 Method

3.1 Simulation Environment

We train on the IsaacGym simulator (Liang et al., 2018), a simulation environment tailored towards allowing policy learning with a high sampling rate (>50K samples/sec in policy inference on Tesla V100 and around 100K samples/sec on RTX 3090) on a single GPU. This is crucial in our goal of providing an accessible yet generalisable framework for sim2real with in-hand manipulation. We simulate the physical system with $\Delta t = 0.02s$, which we found gave a good balance between simulation fidelity, speed of learning, and ability to learn high frequency motions required in such a manipulation task.

3.2 Representation of the Cube Pose

Our task involves manipulating an object in 6 degrees of freedom. As such, we must represent this pose in multiple stages of our training pipeline. Prior work has shown the benefits of alternate representations of spatial rotation when using neural networks (Zhou et al., 2020). We choose to represent the pose of the object using 8 keypoints sampled in the object's local space at each vertex of the cube, $\mathsf{k}_{\mathsf{local,i}}$ $i = 1, \ldots, 8$. These locations of the keypoints in the object's local frame are constant. We denote the keypoints in the world frame for the current pose of the object, $\mathsf{k}_{\mathsf{curr,i}}$ and the goal pose of the object $\mathsf{k}_{\mathsf{target,i}}$ $i = 1, \ldots, 8$. These are obtained by a straightforward transformation of the

[1]Available for download at: `https://developer.nvidia.com/isaac-gym`

Observation space		Degrees of freedom
Finger joints	position	3 fingers · 3 joints · $1[\mathbb{R}^1] = 9$
	velocity	3 fingers · 3 joints · $1[\mathbb{R}^1] = 9$
Cube pose	keypoints	8 keypoints · $3\ [\mathbb{R}^3] = 24$
Goal pose	keypoints	8 keypoints · $3\ [\mathbb{R}^3] = 24$
Last action	torque	3 fingers · 3 joints · $1[\mathbb{R}^1] = 9$
Total		75

(a) Actor Observations

Observation space		Degrees of freedom
Actor Observations (w/o DR)		75
Cube	velocity	$6\ [\mathbb{R}^6]$
Fingertips state	pose	3 fingers · $7\ [\mathbb{R}^3 \times SO(3)] = 21$
	velocity	3 fingers · $6\ [\mathbb{R}^6] = 18$
	wrench	3 fingers · $6\ [\mathbb{R}^6] = 18$
Finger joints	torque	3 fingers · 3 joints · $1[\mathbb{R}^1] = 9$
Total		147

(b) Critic Observations

Table 1: Asymmetric actor-critic to learn dexterous manipulation. While the actor receives noisy observations, which are added as a part of DR (Section 3.5), the critic receives the same information without any noise and also has access to certain privileged information from simulator.

$\mathsf{k}_{\mathsf{local,i}}$ back into the world frame using the current / goal poses of the object. This results in a set of 8 keypoints in 3-D Euclidean space. When concatenated for policy inference, the set of 8 keypoints yields a 24-D vector representing an object pose. In Sec. 4.2, we contrast this representation to a position+quaterinon formulation used in (OpenAI et al., 2018; Liang et al., 2018), finding that it improves the policy's success rate. We note as long as we are able to get the pose of the object (eg. via (Tremblay et al., 2018)), we are able to obtain the 8 keypoints of the bounding box, and that this does not depend on the object morphology (as shown in Appendix **??**).

3.3 Observation and Action Spaces

The observations of the policy (actor) and value-function (critic) are described in Table 1a and 1b. We use the representation of cube pose described in the previous Sec 3.2. The action space of our policy is torque on each of the 9 joints of the robot. The torque on each joint is limited such that it does not damage the equipment while in operation. A safety damping is applied to the output torques in simulation to mimic those on the real-world robot.

3.4 Reward Formulation & Curriculum

Our reward has three components, each of which we found to be helpful in achieving good training and sim-to-real performance. Following (Hwangbo et al., 2019), we use a logistic kernel to convert tracking error in euclidean space into a bounded reward function. We slightly generalise the given formulation to account for a range of distance scales, defining, $\mathcal{K}(x) = (e^{ax} + b + e^{-ax})^{-1}$, where a is a scaling factor and b controls the sensitivity to the kernel at low values of distance. We use $a = 30$ and $b = 2$.

As noted in Sec 3.2, we use keypoints in order to calculate the reward. The component of the reward corresponding to the distance between the cube's current pose and the desired target pose is given by: $\texttt{object_goal_reward} = \sum_{i=1}^{N} \mathcal{K}(||\mathsf{k}_{\mathsf{curr,i}} - \mathsf{k}_{\mathsf{target,i}}||_2)$, where $\mathsf{k}_{\mathsf{curr,i}}$ and $\mathsf{k}_{\mathsf{target,i}}$ are each of the $N = 8$ keypoints at the corners of the current and target cubes, respectively.

In order to encourage the fingers to reach the cube during initial exploration, we give a reward for moving the fingers towards the cube, which was also found to be helpful in (Ahmed et al., 2020). This term is defined by sum of the movement of each fingertip towards the goal per timestep: $\texttt{fingertip_to_object} = \sum_{i=1}^{3} \Delta_i^t$, where Δ denotes the change across the timestep of the fingertip distance to the centroid of the object, $\Delta_i^t = ||\mathsf{ft}_{\mathsf{i,t}} - \mathsf{p}_{\mathsf{curr,centroid,t}}||_2 - ||\mathsf{ft}_{\mathsf{i,t-1}} - \mathsf{p}_{\mathsf{curr,centroid,t-1}}||_2$, and ft_{i} denotes the position of the i-th fingertip.

Finally, we define a penalty on the movement of each finger, preventing sudden motions that may be difficult to execute reliably on the real robot: $\texttt{fingertip_velocity_penalty} = \sum_{i=1}^{3} ||\dot{\mathsf{ft}}_{\mathsf{i}}||_2^2$

Our total reward is defined as:

$$R(s,a) = w_{fo} \times \texttt{fingertip_to_object} \times \mathbb{1}(\texttt{timesteps} \leq 5e7) + w_{fv} \times \texttt{fingertip_velocity_penalty} + w_{og} \times \texttt{object_goal_reward} \tag{1}$$

where $w_{fo} = -750$, $w_{fv} = -0.5$ and $w_{og} = 40$ are the weights of each reward component. We also found in initial experimentation that the curriculum reducing the weight of the `fingertip_to_object` reward to 0 after 50 million steps was needed in order to allow the robot to perform nonprehensile manipulation of the cube (releasing it in order to reorient). However, having the reward term during the initial phases of training dramatically sped up learning by encouraging the robot to interact with the cube.

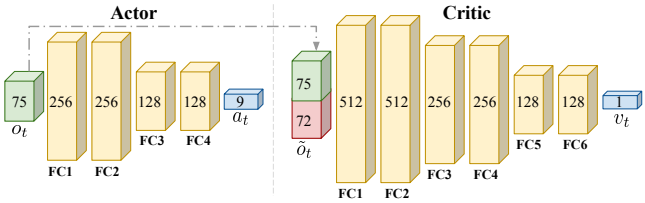


Figure 3: The actor and critic networks are parameterized using fully-connected layers with `ELU` activation functions Clevert et al. (2016).

3.5 Domain Randomization

Domain Randomization (DR) is a straightforward yet practical technique for improving the robustness of policies in sim2real transfer (Tobin et al., 2017; Peng et al., 2018; Mandlekar* et al., 2017). We choose our Domain Randomization parameters to account for modelling errors in the environment as well as noise in sensor measurement noise. These parameters are listed in Table 2. In addition to these randomizations, we apply random forces to the cube in the same manner as described in (OpenAI et al., 2018) in order to improve the stability of grasps and represent unmodelled dynamics. We mimic the dynamics of the camera on the real system, described in Sec 3.7, by repeating the observation of the keypoints for 5 frames.

Parameter	Range	σ	σ_{corr}
Observation Noise			
Cube Position[2]	[-0.30, 0.30]	0.002	0.000
Cube Orientation[2]	[-1.00, 1.00]	0.020	0.000
Finger Joint Position	[-2.70, 1.57]	0.003	0.004
Finger Joint Velocity	[-10.00, 10.0]	0.003	0.004
Action Noise			
Applied Joint Torque	$[-0.36, 0.36]$	0.02	0.01

Parameter	Scaling Distribution
Environment Parameters	
Object Scale	uniform$(0.97, 1.03)$
Object Mass	uniform$(0.70, 1.30)$
Object Friction	uniform$(0.70, 1.30)$
Table Friction	uniform$(0.50, 1.50)$
External Forces	Refer to (OpenAI et al., 2018, pp. 9)

Table 2: For observations and actions, σ and σ_{corr} are the standard deviation of additive gaussian noise sampled every timestep and at the start of each episode, respectively. For environment, the parameters represent scaling factor applied to the nominal values in the real robot model.

3.6 Policy Architecture & Learning

We train using the Proximal Policy Optimization algorithm (Schulman et al., 2017), using the implementation from (Makoviichuk & Makoviychuk, 2021), which vectorizes observations and actions on GPU allowing us to take advantage of the parallelization provided by the simulator (see Sec 3.1). We use the following hyper-parameters: discount factor $\gamma = 0.99$, clipping $\epsilon = 0.2$. The learning rate is annealed linearly over the course of training from $5\text{e}{-3}$ to $1\text{e}{-6}$; detailed hyperparameters are described in Appendix **??** The inputs to the policy are described in Table 1b. We use an asymmetric actor critic approach (Pinto et al., 2017) with an actor that has 4 hidden layers, 2 of size 256 followed by 2 of size 128, and 9 outputs which are scaled to the torque ranges of the real robot and a critic that has 2 layers of size 512, followed by 2 layers of size 256 and 128 each and produces a scalar value function as output.

3.7 Policy Inference on Remote Real Robot

We evaluate our policy remotely on the TriFinger system (Wüthrich et al., 2020) provided by the organisers of the real robot challenge (Funk et al., 2021). The cube is tracked on the system using 3 cameras, described in (Wüthrich et al., 2020). We convert the position+quaternion representation output by this system into the keypoints representation described in 3.2 and use it as input to the policy. Observations of the cube pose from the camera system are provided at 10Hz. Proprioceptive measurements are available at up to 1KHz. Our policy is evaluated at 50Hz, matching the simulation timestep. We repeat the camera-based cube-pose observations for subsequent rounds of policy evaluation to allow the policy to take advantage of the higher-frequency proprioceptive data available to the robot. The resulting observations are identical to what we use in simulation (Table 1a).

4 Experiments

In our experiments, we aim to answer the following four questions pertaining to learning a robust policy for this task, as well as evaluating how well it transfers to the real world:

[2]The noise to keypoints is not applied directly. Rather the noise is added to the cube pose in the world frame before computing the keypoints through it.

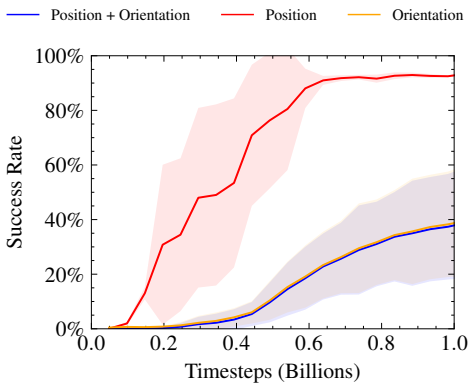


Figure 4: Training curves on a reward function similar to prior work Funk et al. (2021); Chen et al. (2021) for the setting with DR. We take the average of 5 seeds; the shaded areas show standard deviation, noting that curves for `Orientation` and `Position+Orientation` overlap during training. It is worth noting that the nature of the reward makes it very difficult for the policy to optimize, particularly achieving an orientation goal.

1. How well does our system using large-scale simulation train on this task with a reward function similar to what has been previously proposed?
2. How does training performance change when we use a different representation - keypoints - for reward computation and policy input?
3. Is our policy robust to sensor noise and varying environment parameters, and is robustness improved by use of Domain Randomization during training?
4. How well do our policies, trained entirely in simulation, transfer to the real TriFinger system?

4.1 Experiment 1: Training

The aim in our 6-DoF manipulation task is to get the position and orientation of the cube to a specified goal position and orientation. We define our metric for 'success' in this task as getting the position within 2 cm, and orientation within 22°of the target goal pose as used in (OpenAI et al., 2018); comparable to mean results obtained in (Funk et al., 2021). Following previous works dealing with similar tasks (OpenAI et al., 2018; 2019; Ahmed et al., 2020), we attempted applying a reward based on the position and orientation components of error individually.

We spent considerable effort experimenting with a variety of kernels and tuning the parameters of the translation / rotation based reward. The best candidate reward of this format was: $\texttt{object_goal_reward} = \mathcal{K}(||\mathsf{t}_{\mathsf{curr}} - \mathsf{t}_{\mathsf{target}}||_2) + \frac{1}{3\times|\mathsf{rot_dist}|+0.01}$, where $\texttt{rot_dist} = 2 \times \arcsin(\min(1.0, ||\mathsf{q}_{\mathsf{diff}}||_2))$, $\mathsf{q}_{\mathsf{diff}} = \mathsf{q}_{\mathsf{curr}}\mathsf{q}^*_{\mathsf{target}}$. The argument of $\mathcal{K}$ is the logistic kernel that takes L2 norm between the current and target cube position as input, and $\mathsf{rot_dist}$ is the distance in radians between the current and target cube orientation. We use the alternative scaling parameter $a = 50$ in $\mathcal{K}$, which we found to work better in this reward formulation (see Sec. 3.4). We use the same weightings for each of the 3 components of the reward as in Sec 3.4.

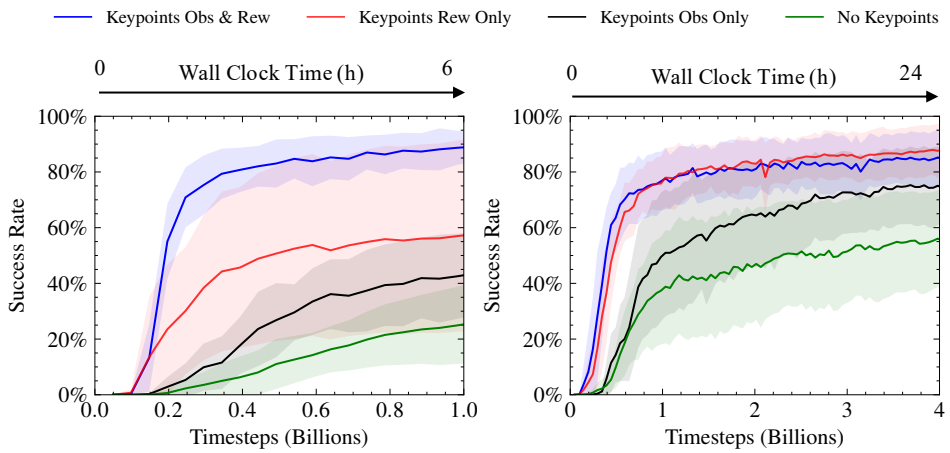
The results are shown in Figure 4. We found that while this formulation of the reward was good at allowing PPO to learn a policy to get the cube to the goal, even after 1 Billion steps in an environment with no Domain Randomization it was learning very slowly to achieve the orientation goal.

4.2 Experiment 2: Representation of Pose

The poor results in Experiment 1 (Sec 4.1) lead us to search for alternative representations of cube pose in the calculation of the reward and policy observations; these are described in Sec 3.2 & 3.4. We compared our method of using keypoints to represent the object pose and using positions and quaternions along two axes. Firstly, using it as the policy input as compared to a position and quaternion representation, and secondly, using it to calculate the reward as compared to a reward based on the linear and angular rotational distances individually.

For the observations, in order to provide a fair comparison between position/quaternion and keypoints as policy input, we ensured that we applied observation noise and delays in the same manner (by applying them in the position and quaternion space before transforming to keypoints, as noted in Sec. 3.5). Also note that both representations only rely on the spatial pose information of the cube to compute. We represent the pose of the cube with a 7-dim vector involving translation and quaternion (t, q). We provide the position and quaternion of the goal pose as input to the actor and critic, replacing the keypoints in Tables 1a and 1b.

For the reward, in order to provide a fair comparison to the keypoints reward, as mentioned previously we spent many hours tuning the kernels and parameters used in the translation based reward, described



(a) No DR. (b) With DR.

Figure 5: Success Rate over the course of training without and with domain randomization. Each curve is the average of 5 seeds; the shaded areas show standard deviation. Note that training without DR is shown to 1B steps to verify performance; use of DR didn't have a large impact on simulation success rates after initial training.

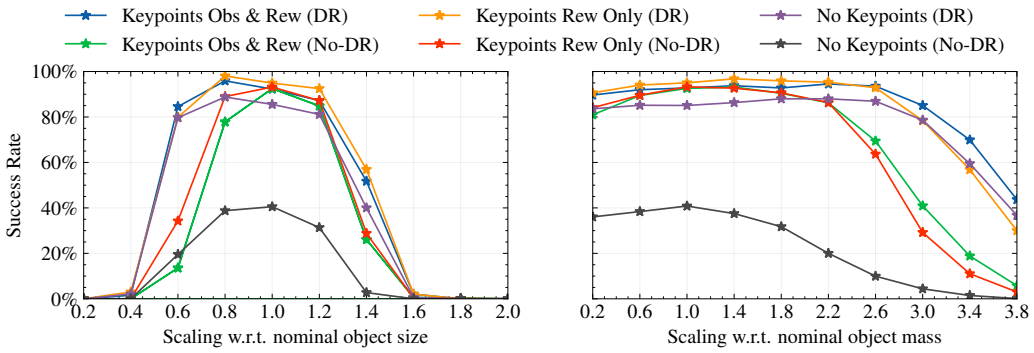


Figure 6: We show the robustness to varying the object parameters outside the DR range it was trained for. In the evaluations, all the other DR is turned off to ensure a controlled setting. Each success is evaluated over 1024 runs with random goal and object initialization.

in Experiment 1. In comparison, little effort was spent tuning the keypoints function, with only one tweak to the weightings in the logistic kernel, showing the relative simplicity of working with this formulation.

Figure 5 shows the results of training, with both timesteps and wall-clock time. In the curve without any Domain Randomization, we trained for 1 Billion steps over the course of 6 hours on a single GPU. Using Keypoints in observations and the reward function performs the best of the four policies, also exhibiting a low variance among seeds.

When Domain Randomization is applied, the two curves with a keypoints-based reward are far better in terms of success rate at the end of training and in terms of convergence rate, however in this case having observations be keypoints seems to matter somewhat less. This is perhaps due to the longer training (4B steps & 24 hours on a single GPU) overwhelming the inductive bias introduced by using keypoints as representations. However, using keypoints to compute the reward provided a large benefit in both cases, showing the improvement caused by calculating the reward in Euclidean space rather than mixing linear and angular displacements through addition.

4.3 Experiment 3: Robustness of Policies in Simulation

In order to investigate the impact that Domain Randomization (DR, see Sec 3.5) has on the robustness of policies of a hand in this configuration, we ran experiments by varying parameters outside of the normal domain randomization ranges in simulation. Figure 6 shows the results. We find that, despite only being randomised initially within a range of 0.97-1.03x nominal size, our policies with Domain Randomization achieve over an 80% success rate even with a scale of 0.6 and 1.2x nominal size, while those without DR have a success rate that drops off much more quickly outside the normal

Object	Success Rate
Cube 6.5cm^3 [Training Object]	92.1%
Ball radius=3.75cm	96.7%
Cuboid 2x8x2cm	2.0%
Cuboid 2x8x4cm	42.9%
Cuboid 4x8x4cm	92.4%
Cuboid 2x6.5x2cm	2.44%
Cuboid 2x6.5x4cm	48.9%
Cuboid 4x6.5x4cm	94.6 %
YCB Mug (`025_mug`)	68.8 %
YCB Banana (`011_banana`)	28.0 %
YCB Potted Meat Can (`010_potted_meat_can`)	81.1 %
YCB Foam Brick (`061_foam_brick`)	91.7 %

Table 3: 0-shot object transfer performance. Numbers calculated from N=1024 trials in simulation. For these, we disabled Environment and Observation & Action Randomizations (though still only gave 10Hz observations to simualte camera tracking). Objects prefixed with YCB were used from the YCB dataset. (Calli et al., 2015)

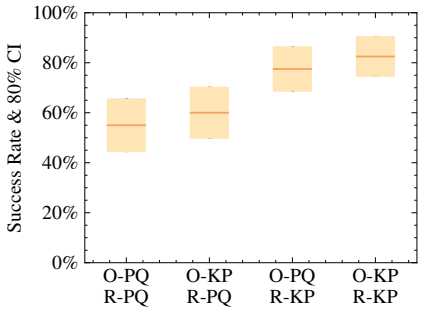


Figure 7: Success Rate on the real robot plotted for different trained agents. O-PQ and O-KP stand for position+quaternion and keypoints observations respectively, and R-PQ and R-KP stand for linear+angular and keypoints based displacements respectively, as discussed in Sec 4.2. Each mean made of N=40 trials and error bars calculated based on an 80% confidence interval.

range. We find similar results when scaling the object mass relative to the nominal range, however in this case we find that the policies using keypoints-based reward even without DR is much more robust at masses 3x nominal.

We experimented with our system to see what the 0-shot transfer performance to different object morphologies was. In order to do this, we swapped the objects in the simulator, ran inference the O-KP+R-KP policy (see Section 4) that produced the best sim2real transfer results, and measured the success rate. We do not change the keypoints representation, but rather keep the 8 keypoints as if they lie on the original 6.5cm^3 cube despite the changing object morphology. We were only able to perform these experiments in simulation, as in the remote Trifinger setup we did not have the capacity to swap out objects. However, using an off the shelf pose detector (eg. (Tremblay et al., 2018)) we are confident that the same system would produce good sim2real transfer results.

We tested a sphere of diameter similar to the cube side length, cuboids of different sizes, and a few objects from the YCB dataset (Calli et al., 2015). The results are listed in Table 3. Our our policy generalises surprisingly well on different object morphologies, for example by achieving nearly 70% accuracy on a mug. However, it struggles with long and skinny objects. This is unsurprising given the difficulty in grasping the cube at less than 0.5x the original scale (or 3cm).

4.4 Experiment 4: Simulation to Remote Real Robot Transfer

We ran experiments on the real robot to determine the success rate of the policies trained with Domain Randomization under the metric defined in Sec 4. We performed $N = 40$ trials for each policy; the results for each of the four ablations on keypoints which we tested are shown in Figure 7.

Out of the four models discussed in 4.2, the best policy achieved a success rate of 82.5%. This was achieved with the use of keypoints used in observations of the policy as well as the reward function during training (O-KP+R-KP). The policy using position+quaternion representations but with a reward calculated with keypoints (O-PQ+R-KP) achieved a 77.5% success rate. These first two policies were well within each others' confidence intervals. This is likely due to the impact of the better representation of keypoints being mitigated somewhat after 4 Billion steps of training, as discussed in 4.2. In contrast, neither of the policies trained using the position & quaternion based reward achieved good success rates, with the policy using keypoints-based observations (O-KP+R-

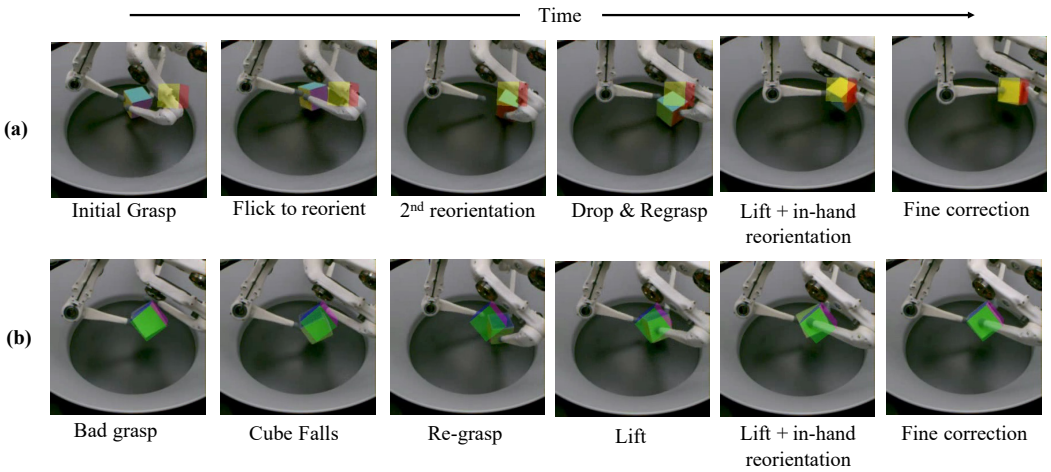


Figure 8: (a) Achieving a challenging goal at the edge of the workspace. (b) Recovery from dropping due to a bad grasp.

PQ) achieving only a 60% success rate while the one with position and quaternion observations (O-PQ+R-PQ) only achieved a 55% success rate. These results show the importance of having a reward function which effectively balances learning to achieve the goal in R^3 and $SO(3)$ in order to have policies with a high success rate in simulation and thus a high corresponding success rate after real robot transfer.

We noticed a variety of emergent behaviours used to achieve sub-goals within the overall cube-reposing task. We display some of these in the panel in Figures 1 and 8. The most prominent of these is "dropping and regrasping". In this maneuver, the robot learns to drop the cube when it is close to the correct position, re-grasp, and pick it back up. This enables the robot to get a stable grasp on the cube in the right position. The robot learns to use the motion of the cube to the correct location in the arena as an opportunity to simultaneously rotate it on the ground to make achieving the correct grasp in challenging target locations far from the center of the fingers' workspace. Our policy is also robust to dropping - it can recover from a cube falling out of the hand and retrieve it from the ground.

5 Summary

This paper emphasizes the empirical value of a systems approach to robot learning through a case study in dexterous manipulation. We introduced a framework for learning in-hand manipulation tasks and transferring the resulting policies to the real world. Using GPU-based simulation, we showed how this can be done with order of magnitude fewer computational resources than prior work. Furthermore, we show how RL algorithms for in-hand manipulation can benefit from using keypoints as opposed to the more ordinary angular and linear displacement-based reward and observation computation. This paper shows a path for democratization of robot learning and a viable solution through large scale simulation and robotics-as-a-service.

REFERENCES

Real robot challenge, 2020. URL https://real-robot-challenge.com.

Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Manuel Wüthrich, Yoshua Bengio, Bernhard Schölkopf, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *CoRR*, abs/2010.04296, 2020. URL https://arxiv.org/abs/2010.04296.

Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, 2015. doi: 10.1109/MRA.2015.2448951.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience, 2019.

Claire Chen, Krishnan Srinivasan, Jeffrey Zhang, and Junwu Zhang. Dexterous manipulation primitives for the real robot challenge. *CoRR*, abs/2101.11597, 2021. URL https://arxiv.org/abs/2101.11597.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In Yoshua Bengio and Yann LeCun (eds.), *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/1511.07289.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

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, 2021. URL http://github.com/google/brax.

Niklas Funk, Charles B. Schaff, Rishabh Madan, Takuma Yoneda, Julen Urain De Jesus, Joe Watson, Ethan K. Gordon, Felix Widmaier, Stefan Bauer, Siddhartha S. Srinivasa, Tapomayukh Bhattacharjee, Matthew R. Walter, and Jan Peters. Benchmarking structured policies and policy optimization for real-world dexterous object manipulation. *CoRR*, abs/2105.02087, 2021. URL https://arxiv.org/abs/2105.02087.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, Jan 2019. ISSN 2470-9476. doi: 10.1126/scirobotics.aau5872. URL http://dx.doi.org/10.1126/scirobotics.aau5872.

Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015. doi: 10.1109/TASE.2014.2376492.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning. In *Proceedings of The 2nd Conference on Robot Learning*. PMLR, 2018. URL http://proceedings.mlr.press/v87/liang18a.html.

Denys Makoviichuk and Viktor Makoviychuk. Rl games, 2021. URL https://github.com/Denys88/rl_games/.

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*, 2021. URL https://arxiv.org/abs/2108.10470.

Ajay Mandlekar*, Yuke Zhu*, Animesh Garg*, Li Fei-Fei, and Silvio Savarese (* equal contribution). Adversarially Robust Policy Learning through Active Construction of Physically-Plausible Perturbations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

Allison M Okamura, Niels Smaby, and Mark R Cutkosky. An overview of dexterous manipulation. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pp. 255–262. IEEE, 2000.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, 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. *CoRR*, abs/1808.00177, 2018. URL http://arxiv.org/abs/1808.00177.

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. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113.

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

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *CoRR*, abs/1710.06542, 2017. URL http://arxiv.org/abs/1710.06542.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Fan Shi, Timon Homberger, Joonho Lee, Takahiro Miki, Moju Zhao, Farbod Farshidian, Kei Okada, Masayuki Inaba, and Marco Hutter. Circus anymal: A quadruped learning dexterous manipulation with its limbs, 2020.

Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL http://arxiv.org/abs/1703.06907.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Proceedings of The 2nd Conference on Robot Learning*, 2018. URL https://arxiv.org/abs/1809.10790.

Manuel Wüthrich, Felix Widmaier, Felix Grimminger, Joel Akpo, Shruti Joshi, Vaibhav Agrawal, Bilal Hammoud, Majid Khadiv, Miroslav Bogdanovic, Vincent Berenz, Julian Viereck, Maximilien Naveau, Ludovic Righetti, Bernhard Schölkopf, and Stefan Bauer. Trifinger: An open-source robot for learning dexterity. *CoRR*, abs/2008.03596, 2020. URL https://arxiv.org/abs/2008.03596.

Manuel Wüthrich, Felix Widmaier, Ossama Ahmed, and Vaibhav Agrawal. Trifinger object tracking, 2020. URL https://github.com/open-dynamic-robot-initiative/trifinger_object_tracking.

Takuma Yoneda, Charles B. Schaff, Takahiro Maeda, and Matthew R. Walter. Grasp and motion planning for dexterous manipulation for the real robot challenge. *CoRR*, abs/2101.02842, 2021. URL https://arxiv.org/abs/2101.02842.

Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks, 2020.