A Appendix

A.1 Training

A.1.1 Learning Algorithm Details

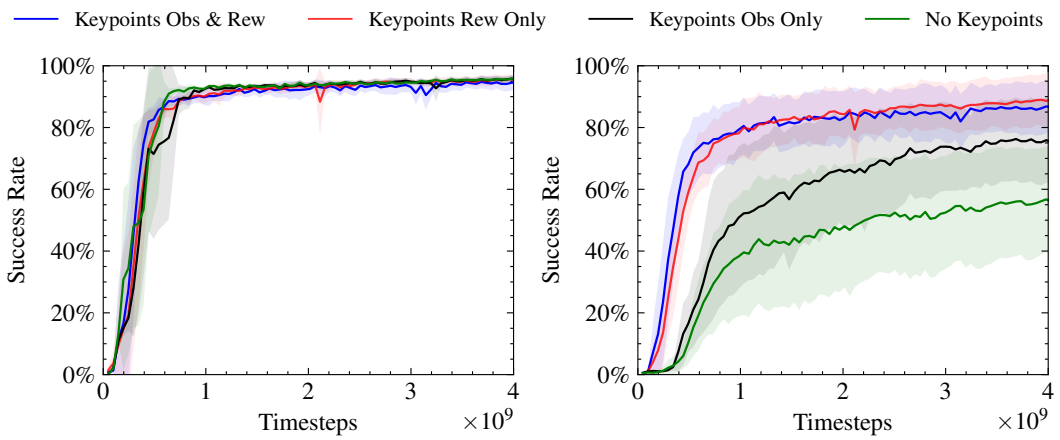
Hyperparameter	Value
Discount Factor	0.99
GAE Discount Factor (τ)	0.95
Learning Rate (start of training)	5e-4
Learning Rate (end of training, linear decay)	1e-6
Batch Size	65356
Mini-batch Size	16384
Number of Epochs	8
Clip Range (ϵ)	0.2
Entropy Coefficient	0

Table 1: PPO Hyperparameters.

We used the open-source version of PPO from (Makoviichuk & Makoviychuk, 2021) which provides the ability to work with highly vectorised environments. The hyperparameters used are listed in Table 1.

A.2 Success for rotation and position

We break out position and rotation success rates individually in Figure 1. They show that the keypoint-based reward formulation fixes the issues identified in *Experiment 1* from the paper, namely that summing position and orientation components of reward leads to poor orientation success rate. Using keypoints improves orientation performance without sacrificing achieving the position goal. It is still apparent that progress can be made with reducing this gap as the orientation reward still continues improving until 4 Billion steps of experience, and this is a direction of ongoing work.



(a) Position Goal at End of Episode (a) Orientation Goal at End of Episode

Figure 1: End of episode success for keypoints and no keypoints.

A.2.1 Reproducing results on a consumer GPU

The reward curves and times stated in the main text were produced on a single NVIDIA V-100 GPU. We were able to reproduce these results on a desktop machine with a consumer-grade NVIDIA RTX3090 GPU. This produced the same reward curves but actually reduced the training times from around 24 hours to 20 hours, showing the ability of our system to train on a desktop.

A.2.2 THROUGHPUT COMPARISON BETWEEN CPU AND GPU

We tested the throughput of the simulator on the same desktop computer with 3090 GPU and 12-core i9-7920X processor to compare the speed of end2end training (simulation, inference and training on CPU) to standard training with simulation on the CPU and neural network inference and training on the GPU. Using the Trifinger environment in Isaac Gym, we got 45,000 steps per second while training using the former while only 6200 using the latter, a speedup of more than 7x.

A.3 DETAILS OF SIM2REAL TRANSFER

A.3.1 SUCCESS THRESHOLDS

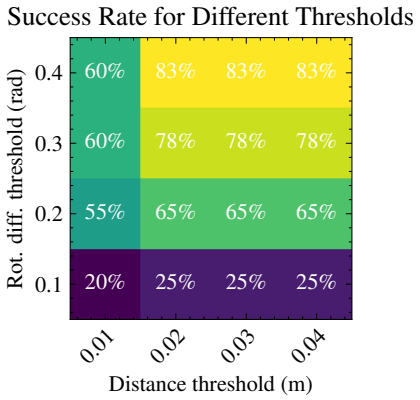


Figure 2: We investigated the impact of varying the thresholds on success rate. The results are shown in the heatmap.

The success rates on the real robot for different thresholds of position and orientation are shown in Figure 2. We see a graceful degradation as the success thresholds are tightened. We note that these are necessarily based on noisy camera observations due to the remote nature of the setup; at 0.01m of position and 0.1rad of orientation error this becomes a particular problem. Note also that 'success' for us is based off a different metric than some other works (eg. (OpenAI et al., 2018)): we define 'success' as being within the goal at the end of an episode instead of achiving it at any point during it. This is because part of the challenge of the *Trifinger* orientation task is being able to grasp *and hold* the cube in position, as the upside-down orientation of *Trifinger* making this challenging.

A.3.2 HARDWARE SETUP

As mentioned in the main text, we perform inference on the Trifinger platform remotely. The interface is described in the corresponding whitepaper (Wüthrich et al., 2020).

Inference, including camera tracking and running the network, is performed on CPU on the same computer that hand-written solutions to last year's real robot challenge (Funk et al., 2021; Chen et al., 2021; Yoneda et al., 2021) were written on. An entire setup to run our system, including training, inference and physical robot hardware, could be purchased for less than US$10,000.

A.3.3 SOFTWARE DETAILS

Inference is done in the Python; the time from getting the observations to sending the actions to the hardware platform is on the order of 5-8ms, a delay consisting of generating keypoints observations and running the policy. Reducing this delay by moving our inference code to C++ is a direction for future improvements to our system.

A.3.4 POSE FILTERING

Unlike some previous works using visual information to perform in-hand manipulation, our system uses the pose estimator provided in (Wüthrich et al., 2020). This performs iterative optimization without reference to the history, and thus can provide temporally inconsistent quaternion inputs to the

policy, with the quaternion value flipping between $+q$ and $-q$. We found that this destabilised the policies which were provided position and quaternion inputs during inference, and so implemented a simple filter over the input: if the quaternion from the last camera measurement q_{last} was within 0.2 of the negated quaternion from a new camera measurement $-q_{new}$, we used $-q_{new}$ in the policy input. While this had no impact on the keypoints model (it performs an analytic transformation prior to policy inference which is invariant to this issue) we found it important to perform this transformation to allow stable grasps in policies which took raw quaternions as input and thus to provide a fair comparison.

We tried using an Extended Kalman Filter using the formulation from (Davison et al., 2007) in order to account for the noise in camera observations. However, we did not find that the performance of our policies on the real-robot was noticeably improved as compared with policies, likely due to the high variance in the unknown acceleration in in-hand manipulation.

REFERENCES

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.

Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6): 1052–1067, 2007. doi: 10.1109/TPAMI.2007.1049.

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.

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

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.

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.