

Advancing In-hand Dexterous Manipulation via Machine Learning

by

Tao Chen

S.B. Shanghai Jiao Tong University, 2016

S.M. Carnegie Mellon University, 2019

S.M. Massachusetts Institute of Technology, 2022

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

© 2024 Tao Chen. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Tao Chen
Department of Electrical Engineering and Computer Science
May 17, 2024

Certified by: Pulkit Agrawal
Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Advancing In-hand Dexterous Manipulation via Machine Learning

by

Tao Chen

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Robots are becoming better at navigating and moving around, but they still struggle with using tools, which severely limits their usefulness for household tasks. Using tools requires dexterously manipulating everyday objects like hammers, scissors, knives, screwdrivers, etc. While simple for humans, manipulating everyday objects remains a long-standing challenge that requires breakthroughs in robotic hardware, sensing, perception, and control algorithms. This thesis proposes machine learning techniques that substantially improve the state-of-the-art performance of dexterous manipulation controllers. It focuses specifically on in-hand object reorientation tasks. Previous works on this problem had limitations like using expensive sensors or hands, only working for a few objects, requiring the hand to face upward, slow object motion, etc. This thesis goes a step further by enabling a low-cost robot hand to dynamically reorient diverse objects in mid-air with the hand facing downward using an inexpensive depth camera. To train such a system, the thesis proposes techniques for robots to learn to reorient objects with a downward-facing hand in the air. It also proposes multiple techniques to improve the time efficiency of the learning algorithms. Additionally, it discusses how to reduce the gap between simulation and reality so that controllers trained in simulation can transfer directly to real systems. Furthermore, the thesis explores the use of tactile sensors in dexterous manipulation. It concludes with a discussion of the current system's issues and outlines future research directions for dexterous manipulation.

Thesis supervisor: Pulkit Agrawal

Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

First, I sincerely thank my advisor, Prof. Pulkit Agrawal, for his valuable guidance and unwavering support during my Ph.D. studies. Pulkit was always there for me, even when the research was not going well. He taught me how to do better, higher-quality research and motivated me to achieve more than I thought I could. Pulkit was very supportive emotionally too. He comforted me when I felt stressed or frustrated mentally. I still remember in my early years, Pulkit would even stay up late into the night with me to improve papers, showing me how to make the paper quality better. It's hard to believe how fast time has gone by. In the past five years, Pulkit and I have worked together on so many things - from setting up the robot hardware in the lab in our first year, to me being a teaching assistant for Pulkit's computational sensorimotor learning class, organizing the computational sensorimotor learning seminar, working on many research projects, mentoring undergraduate and master's students, and more. I am truly grateful to Pulkit for enriching my Ph.D. experience and making it such a fruitful and rewarding journey. I could not have accomplished my research achievements without Pulkit's guidance and support.

I would also like to express my gratitude to my thesis committee members: Professor Daniela Rus and Professor Russ Tedrake. Daniela was very responsive to every request I sent her and provided immense support during our conversations. She could quickly identify critical issues during our discussions and encouraged me to view the project from different angles. Russ taught me what it truly means to be a good roboticist. I took his underactuated robotics class, which was one of the best courses I have ever attended. I was very impressed by the time and effort Russ dedicated to teaching, as well as the quality of the course assignments. This inspired me greatly when I was preparing assignments as a TA for the computational sensorimotor learning class. Russ was always encouraging during our conversations and could guide me towards deeper thoughts on various topics. His perspective on robotics was truly inspirational.

I am also thankful to Professor Abhinav Gupta, who was my advisor during my master's program at CMU. When I first arrived at CMU with little research experience, he believed in me and taught me how to conduct research properly. Abhinav has an impressive vision for many areas, and working with him has been truly inspirational for me.

I am also grateful to George Zhao, the founder of the startup company I worked for a year in Shanghai. He saw my potential and offered me a great opportunity to work on real robot products. Throughout that year, George was extremely supportive of my work and my application to graduate school. I had a wonderful learning experience during that time, which was pivotal in my life as I transitioned from a mechanical engineering major into the field of artificial intelligence.

I would also like to express my gratitude to my mentors during my internships: Eric Cousineau, Naveen Kuppaswamy, Dieter Fox, Yu-Wei Chao, Wei Yang, and Adithya Murali. Completing these internships greatly helped me expand my knowledge and broaden my perspectives on robotics. Thank you for making these internship experiences truly enjoyable and productive. In particular, I want to thank Eric and Yu-Wei for being extremely helpful during my internships and providing hands-on support.

I also want to thank my collaborators. Throughout my research, I was fortunate to work with many talented people. This thesis would not have been possible without your help. Thank you Abhishek Gupta, Alberto Rodriguez, Alisha Fong, Andrew Morgan, Anthony Simeonov, Anurag Ajay, Chuang Gan, Dhiraj Gandhi, Donghyun Kim, Edward Adelson, Gabriel Margolis, Ge Yang, Haozhi Qi, Jie Xu, Joshua Gruenstein, Kalyan Vasudev Alwala, Kartik Paigwar, Krishna Murthy Jatavallabhula, Lara Zlokapa, Lerrel Pinto, Marcel Torne, Meenal Parakh, Megha Tippur, Michael Foshey, Neel Doshi, Sameer Pai, Sangbae Kim, Sangwoon Kim, Saurabh Gupta, Shinjiro Sueda, Shuang Li, Siyang Wu, Sizhe Li, Tao Du, Tommi S. Jaakkola, Vikash Kumar, Wojciech Matusik, Xiang Fu, Xiaolong Wang, Yajvan Ravan., Yen-Chen Lin, Yuzhe Qin, Zechu Li, Zhang-Wei Hong, Zhiao Huang, Zhutian Yang (sorted alphabetically).

My Ph.D. life would not have been enjoyable without my incredible labmates from the Improbable AI lab. My sincere thanks go to Alina Sarmiento, Alisha Fong, Alon Z. Kosowsky-Sachs, Andi Peng, Andrew Jenkins, Anthony Simeonov, Antonia Bronars, Anurag Ajay, Avery Lamp, Aviv Netanyahu, Bipasha Sen, Blake Tickell, Brian Cheung, Eric Chen, Gabe Margolis, Haoshu Fang, Idan Shen, Jacob Huh, Joshua Gruenstein, Jyothish Pari, Marcel Torne, Meenal Parakh, Nolan Fey, Richard Li, Ruben Castro, Sameer Pai, Sanja Simonkovj, Seungwook Han, Srinath Mahakali, Steven Li, Tifanny Portela, Yandong Ji, Yanwei Wang, Younghyo Park, Zhang-Wei Hong (sorted alphabetically).

I would also like to express my gratitude to all my friends. Your constant support, encouragement, and kindness have been invaluable to me throughout this journey. You have been there for me during both the highs and lows, offering a listening ear, wise advice, and cheerful company whenever I needed it most. Your friendship has truly enriched my life, and I am deeply thankful to have you all by my side.

Lastly, I owe my deepest gratitude to my family. My girlfriend has been an immense source of support in this journey. I am profoundly thankful for her patience, understanding, and unwavering encouragement as I dedicated most of my time to research as a Ph.D. student. Without her loving companionship and the strength it gave me, this endeavor would have been much more challenging. My parents and grandparents have always been providing unconditional love and backing every step of the way. I would not be where I am today without their support. Family is my bedrock, giving me strength when I feel weak. I am forever grateful for the sacrifices they have made and the encouragement they have given to help me persevere and achieve the milestones.

Contents

Title page	1
Abstract	3
Acknowledgments	5
List of Figures	11
List of Tables	19
1 Introduction	21
2 Visual Dexterity	27
2.1 Overview	27
2.2 Results	29
2.2.1 Extrinsic dexterity: object reorientation with a supporting surface . .	31
2.2.2 Towards object reorientation in air	35
2.2.3 Generalization to objects in daily life	37
2.2.4 Comparison to prior works	37
2.3 Discussion	39
2.4 Materials and Method	41
2.4.1 Training the visuomotor policy	41
2.4.2 Reducing the simulation to reality gap	47
2.4.3 Real-world deployment	49
2.5 Acknowledgements	49
3 Vegetable Peeling: A Case Study in Constrained Dexterous Manipulation	50
3.1 Introduction	50
3.2 Related work	52
3.3 Method	53
3.3.1 Training Setup	54
3.3.2 Teacher Policy Learning: Reorient and Stop	55
3.3.3 Student Policy Learning: Imitate and Stop	57
3.3.4 Peeling	59
3.4 Results	60
3.4.1 Traveling distance for a fixed amount of commanded motion time . .	60

3.4.2	How well does the controller track the commanded motion time? . . .	61
3.4.3	Firm grasp after reorientation	61
3.4.4	Real-world Peeling	62
3.4.5	Ablation study	62
3.5	Discussions	63
3.6	Acknowledgements	64
4	Parallel Q-Learning	65
4.1	Overview	65
4.2	Related Work	66
4.3	Method	67
4.3.1	Scheme Overview	69
4.3.2	Balance between Actor , P-learner , and V-learner	70
4.3.3	Mixed Exploration	70
4.4	Experiments	71
4.4.1	Setup	71
4.4.2	PQL learns faster than baselines	72
4.4.3	How well does mixed exploration perform?	72
4.4.4	Effects of different hyper-parameters	73
4.4.5	Additional Tasks	79
4.5	Discussion and Future Work	80
4.6	Acknowledgements	81
5	Tactile sensing with a dexterous hand	82
5.1	Overview	82
5.2	Related Work	83
5.2.1	Object Classification with Tactile Sensors	84
5.2.2	Object Localization with Tactile Sensing	84
5.3	System Description	85
5.4	Problem Setup	85
5.5	TactoFind: Identifying and Retrieving Objects in the absence of Visual Feedback	86
5.5.1	Object Localization with Tactile Sensing	86
5.5.2	Dynamic Object Identification	87
5.6	Experimental Evaluation	89
5.6.1	Baselines and Evaluation Metrics	89
5.6.2	Simulation Results	90
5.6.3	Real World Results	92
5.6.4	Ablations	92
5.7	Discussion	93
5.8	Acknowledgements	94
6	Discussion	95

A Visual Dexterity

A.1	Supplementary Methods
A.1.1	Nomenclature
A.1.2	Experiment details
A.1.3	Fabrication
A.1.4	Overcoming sim-to-real gap
A.2	Supplementary Discussion
A.2.1	Student policy closely tracks the performance of teacher policy
A.2.2	Symmetric object reorientation
A.2.3	Ablation on the reward terms
A.2.4	Using a different encoder for goal
A.2.5	Stage 0: speeding up vision policy training with visual pre-training
A.2.6	Ablation on the prediction tasks for vision network pre-training
A.2.7	Analysis for object reorientation in the air in simulation
A.2.8	Discussion on precise manipulation

B Parallel Q-Learning

B.1	Pseudo Code
B.2	Training setups
B.2.1	Hyper-parameters
B.2.2	Hardware Configurations
B.2.3	Vision experiment setup
B.3	Additional Experiments

References

List of Figures

2.1	Illustration of the robot system. (A): the front and side views of our real-world setup. The controller is a neural network that uses depth recordings from a single camera along with the joint positions of the manipulator to predict the change in joint positions. (B): Visualization of the same controller reorienting three different objects. The rightmost column shows the target orientation. The first two rows are instances of a four-fingered hand reorienting objects in the air. The last row shows reorientation with the help of a supporting surface (extrinsic dexterity).	30
2.2	Experimental results of reorientation. (A): twelve objects with their IDs. The first seven objects are from the training dataset \mathbb{B} , and the last five are from the testing dataset \mathbb{S} . (B), (C) show the real-world error distribution when using rigid and soft fingertips, respectively, on material M1. (D) shows the error distribution in simulation for each object as a violin plot [59]. The violet rectangle shows the errors within [25%, 75%] percentile and the horizontal bar in the rectangle depicts the median error. Train objects can mostly be reoriented within an error of 0.4 radians, with similar performance for rigid and soft fingertips. The error on test objects is higher, and soft fingertips exhibit better generalization. (E): five table materials. (F) and (G) show the error distribution on different materials for object #5 and #10, respectively.	32
2.3	Different testing scenarios. We test our controller on objects with diverse shapes and reorientation conditions such as using different supporting surfaces such as a tablecloth, an uneven door mat, a slippery acrylic sheet, and a perforated bath mat. We also evaluate performance using fingertips with different softness: rigid 3D-printed (row (A)), and soft elastomer fingertips (rows (B) to (G)). Row (A) to (E) use a three-fingered robot hand. And row (F) to (G) use a four-fingered robot hand. Our policy can reorient real household objects (rows (E,G)) and can operate without the need for a supporting surface (in the air) as shown in row (G).	33

2.4	Benefit and performance of reorientation with a four-fingered hand. (A): When training a controller to reorient objects with a supporting surface, the three-fingered and four-fingered hands achieve similar learning performance. (B): However, when we incentivize the hands to lift the object during reorientation, the four-fingered hand outperforms the three-fingered hand substantially. (C): We tested the controller performance with a four-fingered hand in the air. We collected 20 non-dropping testing cases for one in-distribution object and one out-of-distribution object. The error distribution is similar to that in the case of table-top reorientation. (D) shows the distribution of the episode time both in simulation and the real world. (E): We show the same controller’s performance on twelve objects with a supporting surface. (F): We tested the controller on symmetric objects with a supporting surface. The controller behaves reasonably well even though it was never trained with symmetric objects.	36
2.5	Reorientation of real objects. Examples of reorienting real objects that were not 3D printed using a four-fingered and a three-fingered manipulator.	38
2.6	Teacher and two-stage student training framework. First, a teacher policy is trained using reinforcement learning with privileged state information. Then, a student policy is trained to imitate the teacher using synthetic and complete point clouds as input. The student policy is further fine-tuned using rendered point clouds. During deployment, the student policy can be directly used to control real robots.	43
2.7	Student policy learning. (A): Student vision policy network architecture. (B): Sparse 3D CNN (Convolutional Neural Network) component of the policy network. (C): Proposed two-stage student learning learns faster than single-stage student learning. The dashed vertical line denotes the transition from the first to the second stage of student learning. The performance dip happens due to a change in the distribution of point cloud inputs from being unoccluded in the first stage to being occluded in the second. (D): Post-training evaluation of teacher and student policies on the training dataset \mathbb{B} . For each object, the initial and target orientations are randomly sampled 50 times, resulting in 7500 samples. The empirical cumulative distribution function (ECDF) of the orientation error is plotted. The results show that the students are close to the teacher’s performance. (E), (F), (G): Comparing the ECDFs of the policies being evaluated on dataset \mathbb{B} and dataset \mathbb{S} reveals small generalization gap for all the policies.	46
3.1	We present a dexterous manipulation system that utilizes an Allegro hand mounted on a Franka robot arm to reorient food items for downstream peeling. The other Franka robot arm (the right arm in the figure) uses its gripper to grasp a peeler for peeling. The reorientation controller for the Allegro hand is learned through reinforcement learning, while the peeling is performed via teleoperation. In the figure, we demonstrate the process of reorienting and peeling a melon, a sweet potato, and a squash from top to bottom row. . . .	51
3.2	Robot setup for reorientation and peeling.	51

3.3	Object dataset used in this work. We collected meshes of carrot, sweet potato, potato, squash, pumpkin, etc.	54
3.4	(a) shows an example of the rotational axis of a melon. (b) shows an example where the object’s orientation (the blue line) has a large deviation from the desired rotational axis (the green line). We reset the episode when this occurs. (c) shows the policy Architecture for the teacher and the student. In this figure, we use \mathbf{o}_t to represent all the policy input at each time step.	57
3.5	Examples of joint position commands after interpolation sent to a low-level PD controller. \odot represents the actual joint position of the motor. \ominus is the computed desired joint position. \bigcirc on the green line shows the interpolated joint position commands that are sent to the low-level PD controller. (a) shows the case of $\mathbf{q}_{t+1}^{cmd} = \mathbf{q}_t + \mathbf{a}_t$, while (b) shows the case of $\mathbf{q}_{t+1}^{cmd} = \mathbf{q}_t^{cmd} + \mathbf{a}_t$. We can see that (b) generates much smoother joint commands.	58
3.6	(a): the Allegro hand holds a papaya to be peeled. (b): we utilize Grounded SAM to segment the papaya. (c): the 3D point cloud representing the segmented papaya’s exposed surface. (d): we take a slice of this point cloud at the center region along the papaya’s longest axis. (e): the points within this center slice are projected onto the central plane aligned with the axis. (f): we fit a spline curve to the projected points to obtain the desired trajectory for the peeler tip to follow.	59
3.7	(a) shows the objects for evaluation: melon, radish, pumpkin, papaya. (b) shows the traveling distance. Before reorientation begins, we ensure a reference point (point A) is facing upward. After reorientation, we identify the point (point B) now facing upward. We then measure the distance from point A to point B along the contour.	60
3.8	(a): Violin plots showing the distribution of the traveling distance of a point on the object surface after the controller is commanded to rotate the object for 3.5 s and 7 s, respectively. (b): Violin plot showing the distribution of time taken by the controller to transition from rotating the object in hand to firmly holding the object after receiving the stop signal. The x -axis represents the timing of the stop signal sent to the controller after it starts.	61
3.9	(a) shows learning curves of the teacher policies with or without $c_3 \ \mathbf{q}_t - \mathbf{q}^{demo}\ _2^2$ in the reward function. (b) shows the differences between student policies trained with different sensory information (joint positions and velocities vs. joint positions only).	62
3.10	Learning curves of student policies with a Transformer or RNN architecture with respect to the number of samples and wall-clock time, respectively.	63
4.1	Overview of Parallel Q Learning (PQL). We have three concurrent processes running: Actor , P-learner , V-learner . Actor collects interaction data. P-learner updates the policy network. V-learner updates the Q functions.	68
4.2	We experiment on six Isaac Gym tasks: <i>Ant</i> , <i>Humanoid</i> , <i>ANYmal</i> , <i>Shadow Hand</i> , <i>Allegro Hand</i> , <i>Franka Cube Stacking</i>	71

4.3	We compare our methods to the SOTA RL algorithms (PPO, SAC with n -step returns, DDPG with n -step returns). We use 4096 environments for training in all tasks except the PPO baseline on <i>Shadow Hand</i> and <i>Allegro Hand</i> tasks, where we use 16384 as it gives the best performance for PPO on these two tasks as shown in Figure 4.5c. Our methods achieve the fastest learning speed in almost all tasks.	73
4.4	We compared our proposed mixed exploration scheme by applying different constant maximum noise values. We can see that the mixed exploration scheme either outperforms or is on par with other schemes, which can save the tuning effort on the noise level.	74
4.5	We sweep over different numbers of environments (N) on both PPO and PQL (our method). Overall, PQL is less sensitive to the number of environments than PPO on both tasks.	75
4.6	We show the averaged returns in evaluation after a fixed amount of training time ΔT . Across the set of different numbers of environments we experimented with (2048, 4096, 8192, 16384), we found that setting $\beta_{p.v} = 1 : 2$ generally works well. $\Delta T = 60$ mins for <i>Ant</i> , and $\Delta T = 80$ mins for <i>Shadow Hand</i> . The complete learning curves are in Figure B.6.	76
4.7	Given different values of N , we show the effect of different $\beta_{a.v}$. An overall trend we observe is that as N gets bigger, it's more beneficial to update the critic more frequently. We also found that $\beta_{a.v} = 1 : 8$ generally works well given different N values. So one can set $\beta_{a.v} = 1 : 8$ as a good initial value, and tune it if necessary on new tasks.	76
4.8	Effect of different batch sizes. Small batch size usually leads to slower learning. If the batch size is too big, the policy learning can slow down because GPUs have limited cores and it takes more time to process a very big batch of data.	77
4.9	(a) and (b): effect of different replay buffer size. (c) and (d): effect of number of GPUs used for running PQL. PQL can be deployed on a flexible number of GPUs. In complex tasks such as <i>Shadow Hand</i> , it is beneficial to have at least 2 GPUs where the Actor runs on a separate GPU as the simulation itself consumes more GPU compute as the task complexity increases.	78
4.10	(a): <i>DClaw Hand</i> task. (b): We compared our proposed PQL-D method with PPO.	79
5.1	System setup showing object retrieval from an occluded bin using only tactile sensing in the real world. The agent has to localize and identify the object from fingertip tactile sensors alone	83
5.2	Depiction of the full pipeline for tactile-only object retrieval. The agent first localizes several objects in the scene by vertically probing in a discrete grid around the environment, while applying minimal force. The object is then interacted with by radially tapping it gently for identification. The collected data is then used to identify if the object matches the object it is tasked to retrieve and then grasping is performed to actually retrieve the object.	85

5.3	Depiction of the contrastive representation learning architecture for object identification. The sequence of contact points goes through a transformer based encoder to provide embeddings that are trained with the InfoNCE loss. Affinity in this representation space can then be used for object identification	89
5.4	Film strip depicting phases of real world localization and identification with our pipeline. We see that objects move minimally, while shapes are successfully identified and grasped. Red box shows the localization, blue box shows the interaction, and the green box shows the grasping.	90
5.5	Visualization of planar occupancy and resulting clusters (the dots) during localization compared with true object centers (the triangles). While showing non-zero error, the relative object centers are well predicted enough for subsequent object identification.	93
A.1	Object dataset. On the left of the red line, we show the dataset \mathbb{B} (the training dataset). And on the right of the red line, we show the dataset \mathbb{S} (the testing dataset in simulation).	
A.2	3D models for the robot hands. (A): three-fingered robot hand. (B): four-fingered robot hand. (C): fingertips with a rounded skeleton and the grey shell represents soft elastomer.	
A.3	Joint response curves. We identified the dynamics of a finger joint (the top one) and show the results. Three curves are plotted: (1) the command sent to the joint, (2) the joint’s simulated response using the identified dynamics parameters, (3) the joint’s real-world response. The identified dynamics parameters allow the simulated joint to move similarly to the real joint.	
A.4	Joint response curves. Dynamics identification on a middle joint of one finger.	
A.5	Joint response curves. Dynamics identification on a bottom joint of one finger.	
A.6	Reward function ablation. Different learning curves as we vary the values of c_1 (a), c_2 (b), c_3 (c), c_4 (d), c_5 (e).	
A.7	Encoder architectures. (a): we tried using separate encoders for the goal point cloud and the scene point cloud. (b) shows that using separate encoders leads to considerably slower policy learning than using a single encoder on merged goal and scene point clouds.	
A.8	Synthetic data. The synthetic data generation in Stage 0.	
A.9	Different architectures for prediction. In (a) and (b), we designed two architectures, with the difference being whether the output of the vision network is split into entity-specific embeddings or not.	
A.10	Learning curves for pretraining. (a) shows the learning curves under different training conditions. The red dots represent the checkpoints we took for policy learning. (b): After the pre-training, we use the vision network as the policy backbone and train the policy with BC. The State policy is a student policy that takes as input the joint positions, rotation matrix of the relative orientation, and object position, which can be seen as an upper bound for the vision policy. Vision from scratch (stage 1) means the vision policy learning in stage 1 only without stage 0.	

- A.11 **Effects of different prediction tasks.** (a): loss curves for the pre-training in Stage 0. (b): learning curves for training the vision policies with the pre-trained vision networks in Stage 1.
- A.12 **Precise manipulation analysis.** Tests for object reorientation in the air in simulation. We tested each object in Figure A.12a 100 times with random initial pose and goal orientation. (a): the objects we used for testing (same as Figure 3A). (b): We show the relationship between the reorientation error and the distance ($\Delta\theta_0$) between the object’s initial and target orientation on non-dropping tests (around 90%). We randomly sub-sample the tests on in-distribution objects to make sure the total numbers of points are the same for in-distribution objects and out-of-distribution objects in this plot. (c): We categorize the testing results for each object into four cases: Success, Orientation error (where the controller stops the object with an orientation error greater than 0.4 radians), Time out, Object falls.
- A.13 **Precise manipulation analysis.** Following Figure A.12, (a) and (b): With object #5 and #10, we compare the distribution of the orientation error and the elapsed time of the episodes in simulation and in the real world. The controller achieves lower error and uses shorter time in simulation. We can see there is still a gap between the simulation and real-world performance. (c) and (d): we show the distribution of the reorientation error and episode time of the non-dropping testing episodes on all twelve objects in simulation.
- A.14 **Reorientation error analysis.** We plotted the actual and predicted rotational distance between the object and goal orientation from 1200 testing episodes on twelve objects in Figure A.12a. (b) is a zoomed-in version of (a) for $x \in [0, 0.4]$ radians and $y \in [0, 0.4]$ radians. The region between the two red lines indicates an error of less than 0.1 radians. Overall, our rotational distance predictor performs reasonably well, but it has limited accuracy when the actual distance is $\Delta\theta \leq 0.4$ radians, indicating the difficulty of precisely predicting the rotational distance.
- B.1 (a): the *Ball Balancing* task in Isaac Gym. (b): the rendered RGB image from the simulated camera. (c) shows the learning curves regarding the number of environment steps. (d) shows the training wall-clock time. We can see that PQL achieves both better sample efficiency and higher final performance than PPO.
- B.2 Comparison of the learning performance with and without the speed ratio control ($\beta_{p:v}, \beta_{a:v}$) on two RTX3090 GPUs and on 1 RTX3090 GPU, respectively.
- B.3 (a) and (b): effect of n -step return. $n = 3$ performs the best. (c) and (d): effect of GPU models used for running PQL. Overall, we see that PQL works robustly across different GPU models, and running on newer GPUs tends to give faster learning.
- B.4 We apply our parallel Q -learning to SAC. PQL + SAC achieves faster learning than SAC itself.
- B.5 Similar to Figure 4.3, we show that our method (PQL) also achieves better sample efficiency than baselines.

B.6 Learning curves for different $\beta_{p:v}$

B.7 Learning curves for different $\beta_{a:v}$

B.8 Comparison between our implementations of PPO and SAC against the ones provided in RL-games. We can see that both codebases provide similar performance, showing that our implementation is good and reliable. On *Shadow Hand*, our PPO learns even faster and better than the PPO in RL-games.

List of Tables

2.1	Statistics of the orientation error when the hand reorients objects on a table. CI stands for bias-corrected and accelerated (BCa) bootstrap confidence interval. Train stands for testing on the seven objects (Figure 2A) from the training dataset B . Test stands for testing on the five objects from the testing dataset S	34
3.1	Successful lifting rate (10 tests each)	62
5.1	Results on object localization in simulation. For both strategies, we measure the success rate (defined as when each predicted center is within $\delta = 7.5cm$ of the object center), the average prediction error, and the average displacement of the objects during the localization. Our clustering strategy scales to multiple objects much better than particle filtering, likely because multiple objects introduce a higher dimensional search space.	91
5.2	Effectiveness of interaction strategies on object identification in simulation. We measure for each policy the overall accuracy when selecting out of five (previously unseen) objects on both the training and validation object sets, as well as the average number of successful taps the policy gets from the object. These experiments show that both our object position estimation as well as accurate contact detection are essential for successful object identification.	91
5.3	Effects of architecture and objective on object identification in simulation. Using a more expressive architecture like a transformer helps with classification, while the InfoNCE loss outperforms using a triplet loss.	92
5.4	Results on full pipeline object retrieval in simulation and the real world. We find that there are expected drops in performance from simulation to the real system, but the identification is still able to significantly outperform random chance	92
5.5	Effectiveness of interaction strategies on object identification in simulation for <i>static</i> objects. We run our object interaction strategy on both moving and static objects and find that static objects are easier to classify than moving ones, showing how challenging our problem setting is	93
5.6	Effect of the coefficient of friction (μ) on localization and classification performance in simulation. As friction reduce, localization becomes much more difficult and identification accuracy also decreases due to increased object movement.	93

A.1	Hyper-parameter Setup	
A.2	Object mass.	
A.3	Dynamics Randomization and Noise	
B.1	Hyper-parameter setup for six Isaac Gym benchmark tasks	
B.2	Reward scale	
B.3	Hardware configurations on different workstations	
B.4	Hyper-parameter setup for the <i>Ball Balancing</i> task.	

Chapter 1

Introduction

Imagine using a screwdriver to tighten a screw. If we think about all the steps, it becomes evident that our hands perform remarkable fine-grained and dexterous manipulation: grasping the screwdriver from a toolbox, precisely reorienting it in mid-air to align with the screw, and executing a coordinated rotation while applying sufficient pressure to secure the screw in place. This is an example of dexterous manipulation, a family of skills crucial in accomplishing various tasks, such as using tools like scissors, screwdrivers, and knives. In-hand object reorientation is a specific dexterous manipulation problem that aims to manipulate a hand-held object from an arbitrary initial orientation to an arbitrary target orientation. The task of in-hand object reorientation subsumes many challenges of dexterous manipulation and is, therefore, an apt task to study and advance dexterous manipulation. Despite the human ability to perform dexterous manipulation actions effortlessly, such manipulation remains an open challenge in robotics.

Dexterous robotic manipulation involves controlling a high degrees-of-freedom (DoF) hand to apply forces to manipulate an object through its fingertips [1]–[3]. Such manipulation poses many challenges, such as frequent making and breaking of contact, real-time feedback control with high-dimensional observation such as camera images, a large control space, objects being in unstable configurations (e.g., when held and manipulated in air), and generalization to different dynamics and geometric properties of objects. These challenges become even more pronounced when robots quickly manipulate objects, as in dynamic dexterous manipulation.

Historically, the challenges of dexterous manipulation have been addressed through various approaches. One line of work [4]–[7] leveraged analytical kinematics and dynamics models of the hand and object. These works either employed trajectory optimization [4]–[6] or model-predictive control [8] to solve for control policies, or utilized kinodynamic planning [7]. Object dynamics can be quite complex when objects have complex geometries, undergo dynamic motion, or possess non-uniform surface properties. For such complex dynamics, employing trajectory optimization or model-predictive control during inference can be computationally expensive. Consequently, it is common to build reduced-order models that simplify the dynamics by making assumptions such as manipulating objects with simple geometries [5], performing static or quasi-static manipulation [6], or utilizing simplifications like small motion of the object or fingers [6]. However, such simplifications are task-dependent, limiting scalability to various tasks, and can also lead to sub-optimal performance. Furthermore, if the object geometry is unknown in advance, using analytical dynamics models is known to

be challenging.

Since building reduced-order dynamics models that work with sensory observations is a challenge, data-driven approaches have been employed to learn dynamics models [9]–[12] directly from sensory observations. The learned model is used to optimize for the desired behavior using model-predictive control. However, when the manipulation task requires the object to be in unstable configurations, such as during in-hand reorientation in the air, collecting data for learning a dynamics model is at least as hard as successfully learning a controller to perform the task. This is because learning a dynamics model that captures unstable configurations requires data where the object is manipulated successfully through the unstable configurations. Instead of relying on random exploration for data collection, one possibility is to use intrinsic rewards [13], [14] to drive the agent’s exploration and learn the dynamics model. However, learned dynamics models only perform well for in-distribution states on which the dynamics model was trained. In the early stages of random exploration, the agent may generate trajectories substantially different from the optimal objection motion, potentially hindering the learning of an effective policy. Instead of autonomously collecting data, an alternative is to collect data via human teleoperation of the robot. Nevertheless, for many in-hand dexterous manipulation tasks, such as reorienting objects in the air with a downward-facing hand, teleoperation is known to be difficult due to the task being inherently unstable, lack of tactile feedback, the morphology gap between the human and robotic hand, etc.

Instead of relying on dynamics models to perform costly inference of actions during deployment, another approach is to learn a policy that outputs actions from sensory observations directly and doesn’t depend on any real-time optimization. Methods that don’t rely on models during inference are known as model-free methods [15]–[19] and can be categorized into two camps: imitation and reinforcement learning (RL). Imitation learning typically involves training policies to mimic demonstration data via supervised learning [20]–[25]. There are various ways to generate demonstration data, such as motion planning, trajectory optimization, or human demonstrations. However, these methods fall short in the case of dynamic dexterous manipulation. Collecting expert demonstration data for dynamic dexterous manipulation is challenging. As mentioned earlier, trajectory optimization does not typically perform well when the manipulation process is dynamic and contact-rich. Moreover, it is challenging for humans to demonstrate dexterous manipulation, especially when there is a significant morphological disparity between the robot and human hands. Furthermore, collecting demonstrations from humans is expensive and time-consuming, substantially limiting the amount of data that can be collected. Consequently, solely training policies with imitation learning is insufficient and lacks robustness over variations in the initial state conditions of the robot and the environment [23].

Some work [26]–[28] resort to directly learning in-hand dexterous manipulation policies with reinforcement learning algorithms in the real world. This benefits from training the policies directly with real-world sensory inputs and dynamics. However, such approaches need to overcome the challenges of resetting the environments between trials [29], measuring task rewards, and collecting data in the real world, which is slow. One way to bypass these issues is to leverage simulation [15], [16], [18], [19] for training policies that are transferred to reality (i.e., sim-to-real). Although policy learning in simulation leverages the simulation engine as a model to generate data, the resulting policy after training is model-free. Consequently,

during inference time, these policies can run in real-time efficiently without dealing with complex dynamics models, which is important for dynamic and dexterous manipulation.

The use of simulation also poses challenges. RL algorithms are typically not data-efficient and require large compute resources. Collecting data in simulation can also be very expensive if the data collection speed is not fast. For example, [15] used 6144 CPU cores and 8 GPUs to train a single object in-hand reorientation policy in simulation. This can make using simulation to train complex robot skills very costly. Furthermore, a substantial sim-to-real gap exists since the simulation model only approximates the real-world dynamics model.

Over the last few years, there has been a significant advancement in GPU-based simulation [30]. Robots can now perform massive parallel exploration in simulation on GPUs, enabling the collection of at least 1000 times more data on a single workstation compared to other CPU-based physics simulations [30]. For example, it only takes an hour to train a Shadow hand to reorient a cube in 3D space in simulation on a commodity GPU [30].

To further lower the time taken to train policies in simulation, it is necessary to improve the data efficiency of RL. Learning a sensorimotor policy with RL is typically data inefficient because the policy needs to simultaneously learn which features to extract from sensory observations, such as vision, and infer the high-rewarding actions. The problem can be made easier if one of two factors were known: sufficient state information or apriori knowledge of high-rewarding actions. Learning a policy via RL from sufficient state information would be much easier than direct learning from sensory observations. Similarly, apriori knowledge of high-rewarding actions would reduce the data requirements of learning from visual observations. Leveraging this intuition, prior works [31], [32] have developed the teacher-student learning framework. The main idea of teacher-student learning is that privileged information, which is information that is readily available in simulations but not in the real world, such as the full object state, can be leveraged to speed up policy learning and improve policy performance. Such a policy cannot be used for real-world deployment as privileged state information is only available in simulation. Therefore, a second real-world deployable policy (student) that operates from sensory observations that can be easily obtained in the real world (e.g., camera images) is trained to mimic the expert policy we trained (teacher). This is typically done using imitation learning. Since it’s a supervised learning process, such distillation from the teacher to the student policy is time efficient. Such a framework has been shown to speed up the overall policy learning for many different applications, including self-driving [32], quadruped locomotion [31], [33]. We also found that it also helps improve the policy learning speed in dexterous manipulation problems [19], [34], [35]. Moreover, a student policy that mimics the teacher policy trained with privileged state information can achieve higher task performance than one directly trained solely on the sensory information available in the real-world environment [34].

Despite being trained via supervised learning, we found that learning vision-based student policies in simulation for challenging tasks such as dynamic in-hand object reorientation can consume much time. For example, as shown in Chapter 2, the naive imitation learning for training a student policy takes about 20 days to learn how to reorient hundreds of objects. A close analysis reveals that a substantial training time is consumed by image rendering in simulation. In Chapter 2, we mitigate this issue using a simple idea: we can leverage the state information and the CAD models of the rigid bodies in simulation to generate a synthetic point cloud for pre-training the vision-based policy. Such synthetic point cloud generation

does not involve any rendering and, hence, is very fast. While there is a difference between the synthetic point cloud and the rendered point cloud, training with the synthetic point cloud provides a good initial policy. We only need to finetune the policy with the rendered point cloud. The two-stage student policy training reduces the demand for rendering and, consequently, the training time. For the task considered in Chapter 2, our pipeline makes training five times faster.

A substantial sim-to-real gap typically exists when transferring the learned policy from simulation to the real world, making policy transfer fail. Two main sources contributing to the difference in performance between simulation and reality (i.e., the sim-to-real gap) are differences in the dynamics and perception. The dynamics gap arises from differences in robot and object dynamics, approximations in the simulator’s contact model, and a lack of knowledge of real-world dynamic parameters such as friction, etc. To bridge this gap, we first performed system identification on the robot dynamics to make the simulated robot behave more similarly to the real robot, and then we used domain randomization to improve the policy’s robustness against unmodeled factors, such as object friction. To perform the system identification on the robot dynamics, we employed a zeroth-order search to find the best simulation parameters for the simulated joints that yield the closest motor response to the real motors. This is feasible in massively parallel simulation, as we can search across tens of thousands of environments in parallel. Notably, only robot dynamics parameters are identified, not the object’s. This is because we want the policy to work on different objects in the real world, and we cannot perform dynamics identification on different objects. To make the policy robust to factors (such as object friction) that are not directly observed in reality, we use domain randomization to make the policy robust to these variations [36]. The perception gap is caused by differences in the statistics of sensor readings and/or noise. To account for this, we add noise to the simulated point cloud.

In Chapter 2, we introduced systematic choices of identifying the robot’s dynamics, employing domain randomization, designing the reward function, and considering hardware factors such as the number of fingers and the fingertip material, which effectively reduced the sim-to-real gap. This enabled us to successfully transfer dynamic in-hand object reorientation skills from simulation to the real world. Moreover, this was achieved using only one camera, whereas other works [15] have relied on multiple cameras.

In-hand object reorientation is a fundamental building block for dexterous manipulation tasks such as tightening a screw with a screwdriver, opening a bottle cap, or peeling vegetables. However, deploying the aforementioned learned in-hand reorientation policies directly to these downstream tasks will likely fail due to additional objectives and constraints required for successful completion. Downstream tasks impose constraints that are absent during the training of object reorientation by itself. For instance, when using a screwdriver, the robot must rotate it along a specific axis and apply sufficient force to drive the screw — constraints not present in normal training. In Chapter 3, we studied peeling vegetables with robot hands as an exemplary case study. Here, the fingers must rotate the vegetable along a particular axis, ensure they do not occlude the top surface after rotation so the vegetable can be peeled, and maintain a sufficiently large holding force to withstand lateral peeling forces. Integrating such task-specific constraints into learned policies poses numerous challenges addressed in this thesis. For the vegetable peeling case, we propose incorporating a demonstration pose from a keyframe into the reward function to encourage the policy to control the fingers properly and

avoid covering the top surface after reorientation. This keyframe pose ensures the fingers strive to stay close to the desired configuration. Obtaining it is straightforward: manually move the fingers to the target pose and record the robot’s state.

As mentioned earlier, the teacher policy is typically trained with reinforcement learning. Chapter 4 presents a method for speeding up RL policy learning in massively parallel simulation. In a massively parallel simulation setup, one common choice for training a policy is on-policy RL algorithms such as PPO [37]. Part of the reason is that it is easy to scale PPO to tens of thousands of environments on a single workstation. However, on-policy methods tend to have worse data efficiency than off-policy methods [38]. Many of the existing off-policy algorithm implementations do not scale well with a larger number of environments and have poor wall-clock time efficiency. If we can scale off-policy methods to tens of thousands of environments on a single workstation, we can potentially get a much more time-efficient learning algorithm. Chapter 4 proposes a scalable framework that adapts off-policy algorithms to leverage many parallel environments for data collection and improves the time efficiency of off-policy methods. We experimentally show that such techniques can substantially boost RL policy’s learning speed on commonly used RL benchmark tasks.

While the teacher-student learning framework enables rapid policy learning, sensor observations can substantially impact policy performance. In certain dexterous manipulation tasks, relying solely on vision can be sub-optimal. The cameras may become occluded during in-hand manipulation, providing no sensory input and significantly challenging the learning task. In the task demonstrated in Chapter 2, some failure cases occurred when the object slipped from the robot’s grasp. We hypothesize that this issue might arise due to the absence of appropriate tactile sensing capabilities to detect object slipping. There are also scenarios where robots need to find objects in the dark (e.g., fetching objects from a bag) where vision sensors are ineffective. To address such limitations of visual manipulation, incorporating additional sensory modalities is beneficial. For example, equipping dexterous hands with tactile sensors can be invaluable when visual information is unavailable. Chapter 5 explores using an omnidirectional GelSight sensor with dexterous hands for localizing, identifying, and grasping target objects in the dark using tactile sensing alone. Imagine a common daily task: retrieving a mouse from a backpack and properly reorienting it to place on a table. We propose a framework enabling robots to explore and fetch the target object using touch sensors analogous to this real-world scenario. The key idea is that the robot hand can collect object shape information by tapping along the contours, gathering contour data. This data can then train a tactile sensory data representation for object identification.

This thesis advances in-hand dexterous manipulation research through machine-learning techniques. The main contributions are as follows:

1. We developed the first real-world system capable of dynamically reorienting objects with complex and previously unseen geometries in the air by any desired amount in real-time, using only a single depth camera.
2. We proposed a novel two-stage teacher-student policy learning framework to learn policies with superior performance and accelerate vision-based student policy learning. Furthermore, we scaled up off-policy methods for massively parallel simulation on a single workstation, significantly enhancing the overall time efficiency of policy learning.

3. We propose a set of systematic choices of identifying the robot’s dynamics, employing domain randomization, designing the reward function, and considering hardware factors such as the number of fingers and the fingertip material, which effectively reduced the sim-to-real gap in dynamic dexterous manipulation tasks.
4. We studied and demonstrated how dexterous manipulation skills should be trained while incorporating task-specific constraints to effectively solve downstream tasks such as peeling vegetables.

Chapter 2

Visual Dexterity

2.1 Overview

The human hand’s dexterity is vital to a wide range of daily tasks such as re-arranging objects, loading dishes in a dishwasher, fastening bolts, cutting vegetables, and other forms of tool use both inside and outside households. Despite a long-standing interest in creating similarly capable robotic systems, current robots are far behind in their versatility, dexterity, and robustness. In-hand object reorientation, illustrated in Figure 1, is a specific dexterous manipulation problem where the goal is to manipulate a hand-held object from an arbitrary initial orientation to an arbitrary target orientation [4], [5], [7], [18], [39]–[41]. Object reorientation occupies a special place in manipulation because it is a pre-cursor to flexible tool use. After picking a tool, the robot must orient the tool in an appropriate configuration to use it. For example, a screwdriver can only be used if its head is aligned with the top of the screw. Object reorientation is, therefore, not only a litmus test for dexterity but also an enabler for many downstream manipulation tasks.

A reorientation system ready for the real world should satisfy multiple criteria: it should be able to reorient objects into any orientation, generalize to new objects, and operate in real-time using data from commodity sensors. Some seemingly benign setup choices can make the system impractical for real-world deployment. For instance, consider the choice of placing multiple cameras around the workspace to reduce occlusion in viewing the object being manipulated [15], [16]. For a mobile manipulator, such camera placements are impractical. Similarly, performing reorientation under the assumption that the hand is below the object (upwards facing hand configuration) [12], [15], [16] instead of the hand holding the object from the top (downwards facing hand configuration) is much easier. With a downward-facing hand, the hand must manipulate the object while simultaneously counteracting gravity. Small errors in finger motion can result in the object falling down. The upward-facing hand assumption makes control easier, but it limits the downstream use of the reorientation skill in many tool-use applications.

Even without real-world setup constraints, object reorientation is challenging because it requires coordinated movement between multiple fingers resulting in a high-dimensional control space. The robot must control the amount of applied force, when to apply it, and where the fingers should make and break contact with the object. The combination of

continuous and discrete decisions leads to a challenging continuous-discrete optimization problem that is often computationally intractable. For computational feasibility, a majority of prior works constrain manipulation to simple convex shapes such as polygons or cylinders [6], [11], [15], [17], [28], [41]–[49]. Other simplifying assumptions include designing specific movement patterns of fingers [43], [50], assuming fingers never make and break contact with the object [6], [51], hand being in an upward-facing configuration [5], [12], [15] or the manipulation being quasi-static [50], [52]. Such assumptions restrict the applicability of reorientation to a limited set of objects, scenarios, or orientations (for example, along only a single axis).

Complementary to the control problem is the issue of measuring the state information the controller requires, such as the object’s pose, surface friction, whether the finger is in contact with the object, etc. Touch sensors provide local contact information but are not widely available as a plug-and-play module. The difficulty in using visual sensing is that fingers occlude the object during reorientation. Recent works employed RGBD (RGB and depth) cameras to estimate object pose but require a separate pose estimator to be trained per object, which limits their generalization to new object shapes [15], [16], [50], [53].

Due to challenges in perception and control, no prior work has demonstrated a real-world ready reorientation system. Although controlling directly from perception is hard, given the full low-dimensional representation of relevant state information such as the object’s position, velocity, pose, and manipulator’s proprioceptive state, it is possible to build a controller using deep reinforcement learning (RL) that successfully reorients diverse objects in simulation [18]. RL effectively leverages large amounts of interaction data to find an approximate solution to the computationally challenging optimization problem of solving for reorientation. However, as a result of requiring large amounts of data and full state information, today, such RL controllers can only be trained in simulation. This leaves at least two open questions: how to train controllers with sensors available in the real world such as visual inputs and whether controllers trained in simulation transfer to the real world (sim-to-real transfer problem).

The difficulty in training RL controllers from visual inputs stems from the learner’s need to simultaneously solve the problem of inferring the relevant state information (feature learning) and determining the optimal actions. If the optimal actions were known in advance, it would be simpler to train a model that predicts these actions from visual inputs (supervised learning). Such a two-stage teacher-student training paradigm, where first a control policy is trained via RL with full state information (teacher) and then a second student policy trained via supervised learning to mimic the teacher has been successfully used for several applications [18], [32], [54]–[56]. We found the major roadblock in learning a visual policy that works across diverse objects is the slow speed of rendering in simulation which resulted in training times of over 20 days with our compute resources. Such slow training makes experimentation infeasible. We devised a two-stage approach for training the vision policy that first uses a synthetic point cloud without the need for rendering and is then finetuned with rendered point cloud to reduce the sim-to-real gap. Our pipeline makes training 5× times faster. The second consideration was the use of a sparse convolution neural network to represent the policy to process point clouds at the speed required for real-time feedback control (12Hz in our case). By directly predicting actions from point clouds, our approach bypasses the problem of consistently defining pose/keypoints across different objects, allowing for generalization to new shapes.

The next challenge is in overcoming the sim-to-real gap. In dynamic in-hand object reorientation, both the robot and the object move quickly. Achieving precise control in a system with fast-changing dynamics is challenging. It becomes even more challenging when using a downward-facing hand as control failures are irreversible. Therefore, dynamic in-hand object reorientation poses a substantial sim-to-real transfer challenge. Some reasons for the sim-to-real gap are differences in motor/object dynamics, perception noise, and modeling approximations made by the simulator. For instance, contact models in fast simulators tend to be a crude approximation of reality, especially for non-convex objects [57]. Whether sim-to-real transfer of reorientation controller is even possible for these complex object shapes remained unclear.

The systematic choices of identifying the manipulator dynamics (details in Method section), domain randomization [36], the design of reward function, and the hardware considerations, including the number of fingers and the fingertip material, reduced the sim-to-real gap. We conducted experiments in the challenging downward-facing hand configuration. We tested the controller’s ability to make use of an external support surface for reorientation (extrinsic dexterity [3]) and the harder condition when the object is in the air without any supporting surface. The results show progress towards developing a real-time controller capable of dynamically reorienting new objects with complex shapes and diverse materials by any amount in the full space of rotations ($SO(3)$, special orthogonal group in three dimensions) using inputs from just a single commodity depth camera and joint encoders. While there is substantial room for improvement, especially in achieving precise reorientation, our results provide evidence that sim-to-real transfer is possible for challenging tasks involving dynamic and contact-rich manipulation in less-structured settings than previously demonstrated.

Finally, many prior efforts used custom or expensive manipulators (such as the Shadow Hand [12], [15], [16] costing over \$100,000) and often relied on sophisticated sensing equipments such as a motion capture system. Such a hardware stack is hard to replicate due to its cost and complexity. In contrast, our hardware setup costs less than \$5,000 and uses only open-source components, making it easier to replicate. Furthermore, our platform is not specific to object reorientation and can be used for other dexterous manipulation tasks. Due to the low barrier to entry, and the evidence that such a system can tackle a challenging manipulation task, our platform can democratize research in dexterous manipulation.

2.2 Results

We trained a single controller to reorient 150 objects from an arbitrary initial to a target configuration in simulation. The learned controllers are deployed in the real world on the open-source three-fingered D’Claw manipulator [58] and a modified four-fingered version with nine and twelve degrees of freedom (DoFs), respectively. The robot’s observation is a depth image captured from a single Intel RealSense camera and the proprioceptive state of the fingers. The goal is provided as the point cloud of the object in a target configuration in the $SO(3)$ space. The initial configuration of the object is a random transformation in $SE(3)$ (special Euclidean group in three dimensions) space within the range of the robot’s fingers – either the object is set on a table or handed over by a human to the robot.

We experimented with the hand in the downward-facing configuration in two settings:

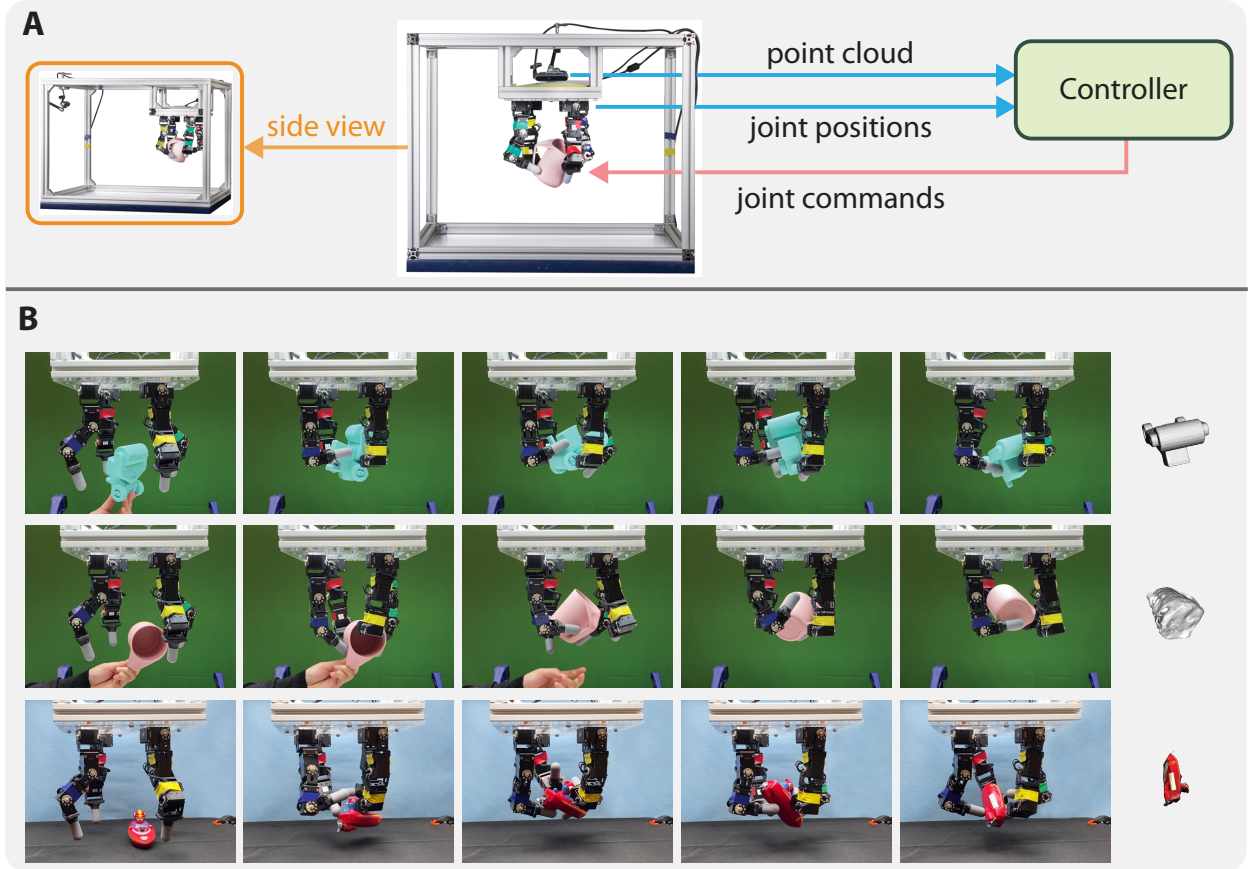


Figure 2.1: **Illustration of the robot system.** (A): the front and side views of our real-world setup. The controller is a neural network that uses depth recordings from a single camera along with the joint positions of the manipulator to predict the change in joint positions. (B): Visualization of the same controller reorienting three different objects. The rightmost column shows the target orientation. The first two rows are instances of a four-fingered hand reorienting objects in the air. The last row shows reorientation with the help of a supporting surface (extrinsic dexterity).

with and without a supporting table. Our system runs in real-time at a control frequency of 12 Hz using a commodity workstation. Figure 1 shows the intermediate steps of manipulating three objects to target orientations depicted in the rightmost column. The proposed controller reorients a diverse set of new objects with complex geometries not used for training. The main text movie provides a short summary of our results with audio. Movie S1 shows our system reorienting many objects and provides a more detailed summary of our major findings. Movie S2 visualizes the setting where the robot is tasked with a sequence of target orientations. In such a scenario, it has to stop when it reaches the current target orientation and then restart to achieve the next target.

For quantitative evaluation, we use seven objects from the training dataset (\mathbb{B}), which we refer to as in-distribution, and five objects from the held-out test dataset (\mathbb{S}), which we refer to as out-of-distribution (OOD). Objects are shown in Figure 2A. We test each object 20 times with random initial and goal orientation in each testing condition. We 3D print these

objects to ensure the shape of objects in simulation and the real world is identical, which is helpful in evaluating the extent of sim-to-real transfer. While the shape of these seven objects is included in the training set, the surface properties such as friction of the real-world objects, may not correspond to any object used for training in simulation. Evaluation on five OOD objects tests generalization to shapes. To further showcase generalization to shapes and different material properties, we also present results on some rigid objects from daily life. The orientation errors are measured using an OptiTrack motion capture system that tracks object pose. We define error as the distance between the goal and the object’s orientation when the controller predicts it has reached the goal and stops. The motion capture is only used for evaluation and is not required by our controller otherwise.

2.2.1 Extrinsic dexterity: object reorientation with a supporting surface

We first report results on the easier problem of reorienting objects when the table is present below the hand to support the object. Using an external surface to aid reorientation has been referred to as extrinsic dexterity [3] and is necessary in many real-world use cases. Visualization of the proposed controller reorienting a diverse set of objects is provided in Figure 3. To demonstrate the versatility of our system, we present results of the robot manipulating objects of different shapes, materials, surfaces, fingertip materials, and varying numbers of fingers.

Reorientation using a three-fingered manipulator with rigid and soft fingertips

With table support, we found three fingers to suffice for the reorientation task. The error distribution for different objects, when tested on a table surface covered with a white cloth (material M1 in Figure 2E), is shown in Figure 2B using a violin plot [59]. Although the overall error distribution is more informative, for ease of comparison, in Table 1, following the success threshold used in previous work [15], we report summary statistics of success rate measured as the percentage of tests with error within 0.4 or 0.8 radians. The seven train objects can be reoriented within an error of 0.4 radians 81% of the time. On the five OOD test objects, the success rate is lower at 45%. As expected, the performance is better with a relaxed error threshold of 0.8 radians and worse at stricter thresholds.

Qualitatively observing the robot behavior revealed that some causes of failure were the object overshooting the target orientation or the finger slipping across the object, especially for OOD objects. One explanation is that rigid hemispherical fingertips contact the object in a very small area (close to making a point contact), which makes small errors in the action commands more pronounced. Further, we found that the fingertip material had low friction resulting in slips which made manipulation harder. To mitigate these issues, we designed and fabricated soft fingertips that cover the rigid 3D-printed skeleton with a soft elastomer (see Figure S2c in the supplementary material). Soft fingertips provide higher friction and deform when contact happens (compliance), increasing the contact area between the finger and the object. The error distribution in Figure 2C shows using soft fingers doesn’t affect performance on train objects but improves generalization to OOD objects. Results in Table 1 confirm the findings – success rate on OOD objects increases from 45% to 55% when switching from rigid

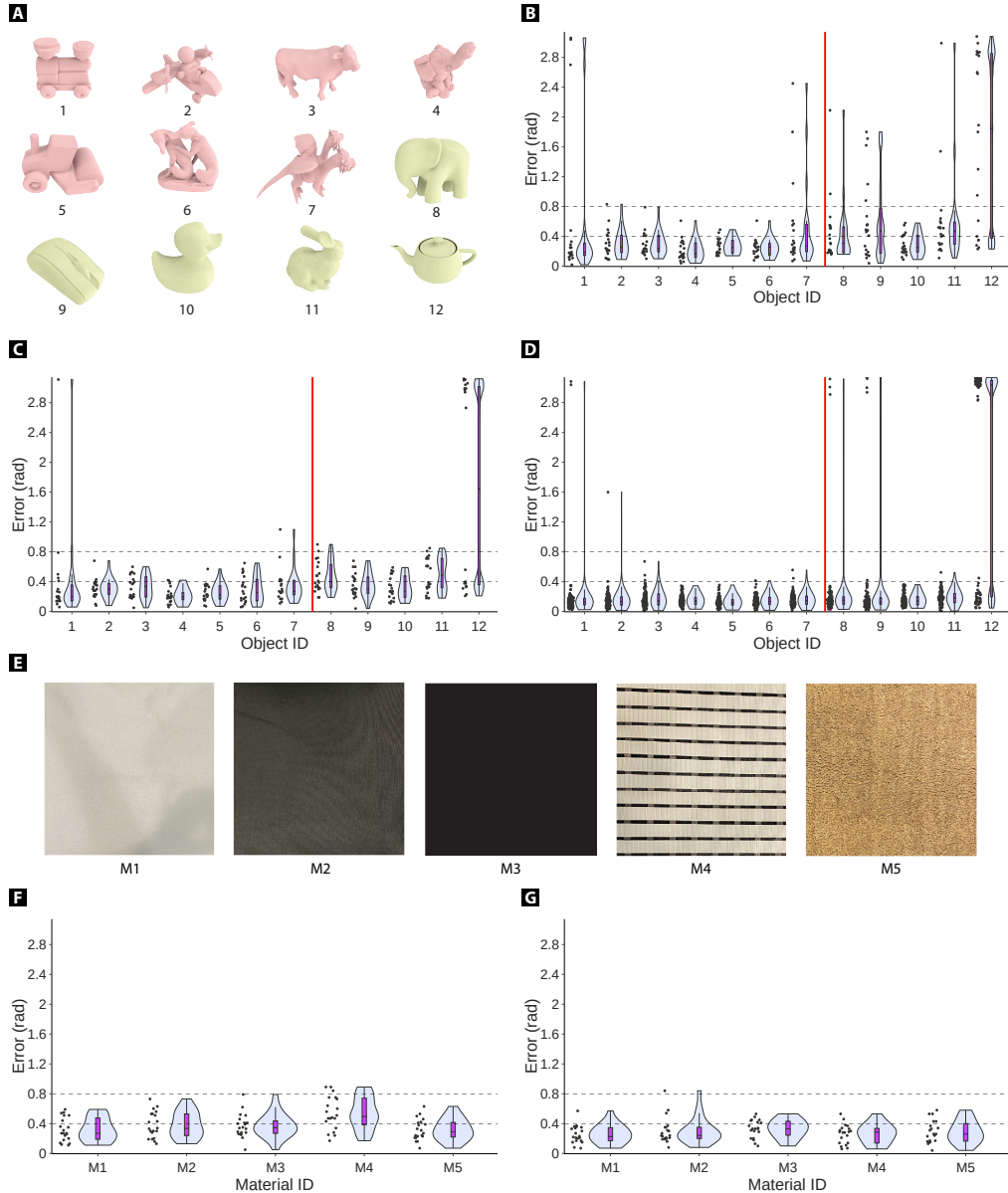


Figure 2.2: **Experimental results of reorientation.** (A): twelve objects with their IDs. The first seven objects are from the training dataset \mathbb{B} , and the last five are from the testing dataset \mathbb{S} . (B), (C) show the real-world error distribution when using rigid and soft fingertips, respectively, on material M1. (D) shows the error distribution in simulation for each object as a violin plot [59]. The violet rectangle shows the errors within [25%, 75%] percentile and the horizontal bar in the rectangle depicts the median error. Train objects can mostly be reoriented within an error of 0.4 radians, with similar performance for rigid and soft fingertips. The error on test objects is higher, and soft fingertips exhibit better generalization. (E): five table materials. (F) and (G) show the error distribution on different materials for object #5 and #10, respectively.

to soft fingertips. Qualitatively, we noticed that soft fingertips behave less aggressively than

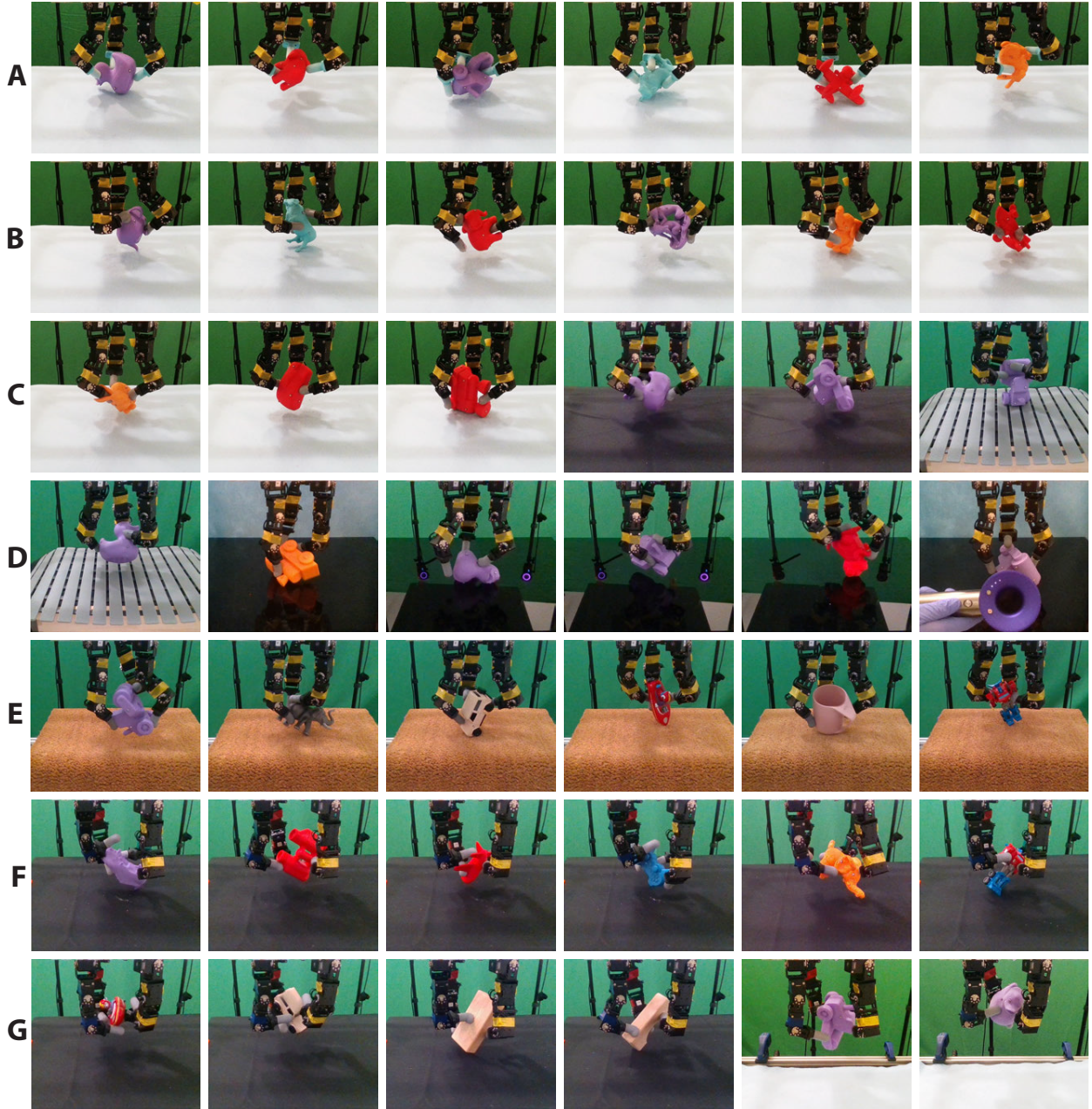


Figure 2.3: **Different testing scenarios.** We test our controller on objects with diverse shapes and reorientation conditions such as using different supporting surfaces such as a tablecloth, an uneven door mat, a slippery acrylic sheet, and a perforated bath mat. We also evaluate performance using fingertips with different softness: rigid 3D-printed (row (A)), and soft elastomer fingertips (rows (B) to (G)). Row (A) to (E) use a three-fingered robot hand. And row (F) to (G) use a four-fingered robot hand. Our policy can reorient real household objects (rows (E,G)) and can operate without the need for a supporting surface (in the air) as shown in row (G).

rigid fingertips resulting in smoother object motion. We, therefore, use soft fingertips in the

Table 2.1: **Statistics of the orientation error when the hand reorients objects on a table.** CI stands for bias-corrected and accelerated (BCa) bootstrap confidence interval. **Train** stands for testing on the seven objects (Figure 2A) from the training dataset \mathbb{B} . **Test** stands for testing on the five objects from the testing dataset \mathbb{S} .

	with rigid fingertips (real)		with soft fingertips (real)		in simulation	
	Train	Test	Train	Test	Train	Test
≤ 0.4 radians (22.9°) 95% CI	81% [73%, 90%]	45% [32%, 58%]	79% [71%, 86%]	55% [44%, 62%]	96% [94%, 97%]	85% [82%, 88%]
≤ 0.8 radians (45.8°) 95% CI	95% [88%, 98%]	75% [46%, 91%]	98% [96%, 99%]	86% [58%, 96%]	98% [97%, 99%]	87% [84%, 90%]
95% CI of the median of orientation errors (radian)	[0.20, 0.27]	[0.29, 0.46]	[0.21, 0.28]	[0.33, 0.42]	[0.12, 0.13]	[0.15, 0.18]

rest of the experiments. It’s worth noting that although the controller was trained using a rigid-body simulator, its performance does not degrade when applied to soft fingertips.

The reorientation error can result from imperfect training, sim-to-real gap, generalization gap, or failures at detecting if the object is at the target orientation, which triggers the controller to stop. In Figure 2D, we report the error distribution in simulation. Although the trained controller is not perfect in simulation, the errors in simulation follow the same trend as in the real world (Figure 2C) but are lower, indicating some sim-to-real gap. As shown in Table 1, the performance gap between the simulation and the real world is smaller with a relaxed error threshold of 0.8 radians than with a threshold of 0.4 radians, illustrating the difficulty in precise reorientation. For some objects (#1, #12), the error distribution is bi-modal both in simulation and the real world. The test runs with high errors largely result from incorrect detection of when to stop. For instance, object #12 appears nearly symmetric in the point cloud representation, which often leads to errors close to 180°. Although it is hard to quantitatively disentangle errors originating from incorrect action prediction and the stopping criterion, based on our experience with the system, we hypothesize that the latter contributes more which is supported by the analysis in Supplementary Discussion (see Discussion on precise manipulation).

Object reorientation on different supporting materials

Changing the table surface changes the dynamics of object motion. We tested if our controller is robust to a diverse set of materials: a rough cloth (M1), a smooth cloth (M2), a slippery acrylic sheet (M3), a bathtub mat with perforations resulting in non-stationary object dynamics depending on the object’s position on the mat (M4), and a door mat with uneven texture (M5). The materials have different surface structures, roughness, and friction, leading to different system dynamics. We evaluate with one in-distribution object (object #5) and one out-of-distribution object (object #10). Figure 2F and Figure 2G show that our controller performs similarly on different supporting materials, demonstrating its robustness.

2.2.2 Towards object reorientation in air

As the controllers discussed above were trained with a supporting surface, when the supporting surface was removed, the manipulator consistently dropped the object resulting in failures. Prior work used a specialized training procedure of configuring the object in a good pose at the start of each training episode and a manually designed gravity curriculum [18] to learn in-air (without supporting surface) reorientation controllers. Consequently, it was necessary to train separate controllers for reorientation with a supporting surface and in the air. It is preferable to have a single controller capable of in-air reorientation and use the supporting surface, if available, to recover from any dropping failures. We achieved this desideratum by employing a four-fingered hand and designing a reward function that penalizes contact between the object and the supporting surface to discourage the controller from using external support for reorientation. When the controller is trained on a supporting surface with the proposed reward function, in-air reorientation emerges.

Although both three and four-fingered hands can reorient objects on a supporting surface (Figure 4A), only the four-fingered hand was capable of in-air reorientation (Figure 4B). We hypothesize this to be the case because, with four fingers, more finger configurations can reorient the object, making it easier for policy optimization to find one solution. Furthermore, we hypothesize that the redundancy in the number of fingers makes the system more robust to errors in action prediction.

SO(3) object reorientation in air

Figure 1B shows how our controller trained in simulation reorients different real-world objects in the air. In-air reorientation can fail if the object is not accurately reoriented or if the robot drops the object. Because in-air reorientation is more challenging, it is possible that the controller is less accurate at reorienting objects. On evaluation with two objects, we found the distribution of orientation error in trials where the objects are not dropped (Figure 4C) to be similar to reorientation with the supporting surface, indicating that the controller doesn't lose reorientation precision in the more challenging in-air scenario. In simulation analysis, we did not notice any notable correlation between orientation error and the distance between the initial and target orientations (Figure S12b in the supplementary material), indicating that the controller performs similarly in the full SO(3) space.

Our controller performs dynamic reorientation. The median time for manipulation across objects and randomly sampled orientation distances in the full SO(3) space is less than 7s (Figure 4D), which makes it a fast in-air reorientation controller operating in the full SO(3) space. Figure 4D also shows that the reorientation times in the real world are longer than in simulation, which we believe is due to real-world contact dynamics being different from simulation.

Simulation analysis reveals that object dropping is the most notable source of errors (Figure S12c). Dropping rates vary substantially across objects. Real-world results follow the same trend. The dropping rate of a shape used in training, the truck (object #5), was 23%, much lower than the dropping rate of 56% for an out-of-distribution duck-shaped object (#10). The dropping rate for the duck object shape in the simulation was around 20% showing a sim-to-real gap. However, it remains unclear if the difference in performance can

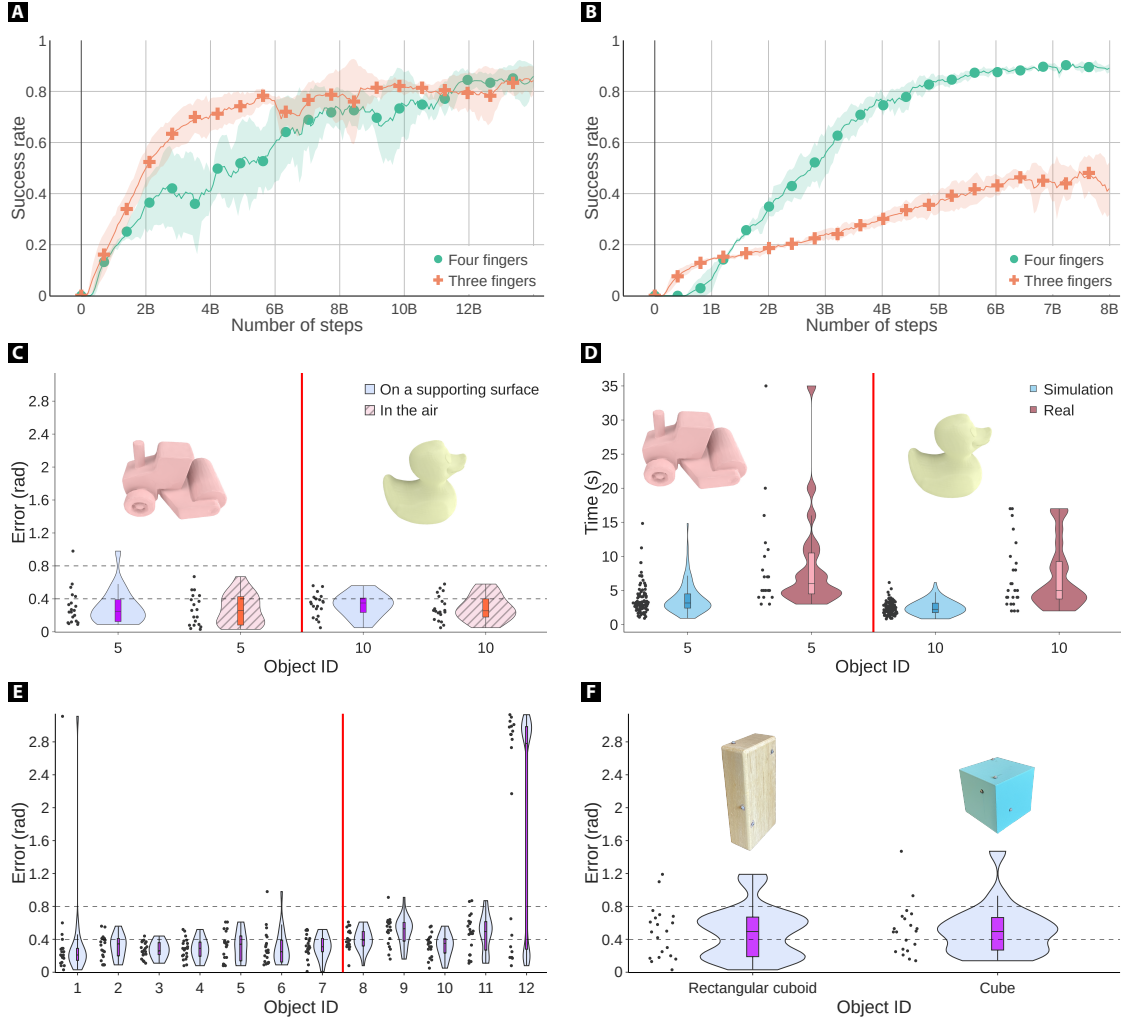


Figure 2.4: **Benefit and performance of reorientation with a four-fingered hand.** (A): When training a controller to reorient objects with a supporting surface, the three-fingered and four-fingered hands achieve similar learning performance. (B): However, when we incentivize the hands to lift the object during reorientation, the four-fingered hand outperforms the three-fingered hand substantially. (C): We tested the controller performance with a four-fingered hand in the air. We collected 20 non-dropping testing cases for one in-distribution object and one out-of-distribution object. The error distribution is similar to that in the case of table-top reorientation. (D) shows the distribution of the episode time both in simulation and the real world. (E): We show the same controller’s performance on twelve objects with a supporting surface. (F): We tested the controller on symmetric objects with a supporting surface. The controller behaves reasonably well even though it was never trained with symmetric objects.

be attributed to the simulator being an approximate model of the real world or whether the object in the real world is much harder to manipulate. This is because, even though the simulation and real-world experiments used the object with the same shape, properties such as surface friction that are critical in reorientation can be different. If an object is curved and has a smooth surface, which is the case with the duck, small differences in friction can

substantially change the task difficulty. We chose to report results on the duck as it was used in prior work [50] and is among the harder objects to reorient and thus also highlights the limitations of our controller.

If a table is present below the hand (for example, the setup shown in the third row of Figure 1B) and the object is dropped, we notice that our controller picks up the object and continues reorienting – an instance of recovery from failures. It is possible that the reward term encouraging in-air reorientation might hurt on-table reorientation. However, the error distribution for on-table reorientation with the updated reward function (Equation 6)(Figure 4E) is similar to earlier on-table experiments. Moreover, although our controller is trained using objects with asymmetry or reflective symmetry, which makes learning much easier, we noticed some generalization to symmetric objects (Figure 4F, more discussion in Supplementary Discussion). The in-air, on-table, and dropping recovery results demonstrate that it is possible to build a single controller that works across different scenarios.

Qualitatively looking at the reorientation behavior, it might appear that the object is not always moving toward the target orientation. One possibility is that the manipulator randomly moves the object until it gets close to the target orientation by chance and then stops. To rule out this possibility, we provide videos in Movie S1 showing that for the same initial but different target orientation, the object motions are different. And for the same initial and target orientation, object motions across trials are similar, which would not be the case if the object was randomly being reoriented.

2.2.3 Generalization to objects in daily life

In previous experiments, we used 3D-printed objects for quantitative evaluation. However, real-world objects have varying object dynamics due to differences in material properties, non-uniform mass distribution, and other factors that can vary across the object surface. To test the generalization ability of our controller on such objects, we conducted a qualitative evaluation on a few household objects. Since we did not have the CAD (Computer Aided Design) model of these objects to generate point clouds in target orientations, we used a free iPad App called Scaniverse to scan the objects. Note that the scan was only required to specify the target orientation, and the scanned object cloud was imperfect (see Figure 5), resulting in noisy goal specification. Figures 1B and 5 illustrate examples of reorienting such objects. The results illustrate that the controller exhibits a certain degree of robustness against noise in the goal specification and some ability to generalize to new materials and shapes.

2.2.4 Comparison to prior works

Unfortunately, a strictly fair comparison with prior work is not possible as we make fewer assumptions (such as no object-specific pose trackers, reorientation in full $SO(3)$ space, and not being quasi-static), and there are substantial differences in hardware/sensing. Nevertheless, to contextualize our research within the existing literature, we present an approximate comparison to the closest work that reported reorientation results on a duck-shaped object with a downward-facing but under-actuated hand of different morphology and mechanical properties [50]. They reported a success rate of 60% (3 out of 5 tests) for reorienting the duck



Figure 2.5: **Reorientation of real objects.** Examples of reorienting real objects that were not 3D printed using a four-fingered and a three-fingered manipulator.

quasi-statically (reorientation time of more than 70s compared to ~ 7 s for our controller) to within 0.1 radians, but only in a subset of the $SO(3)$ space (rotation only along two axes). Further, they used a precise object-specific pose tracker (error < 2 degrees or 0.034 radians). If we assume perfect stopping criteria (the agent stops reorientation if the object is within 0.1 radians of the target), then for the duck-shaped object, we achieve a success rate of 71% when dynamically reorienting in the full $SO(3)$ space in simulation. Due to challenges in setting up precise stopping in the real world, we could not run these evaluations in the real world. Even if we did, the differences in material properties between the duck used by us and prior research [50] would make the comparison unfair. Comparing our simulation and their real-world results is also unfair. However, the results indicate that with more assumptions, such as the precise stopping criterion, the performance of our system improves. Improving the precision of our system without any additional assumptions is an exciting avenue for future research.

The differences in experimental setups with other prior works [15], [16], [49], [52] and

concurrent work [60] are even larger. For instance, OpenAI’s work [15] reported results on reorientation with a single object (no generalization), with a simple shape (cube), an upward-facing hand, and an extensive sensing system consisting of three RGB cameras, a motion capture system, and a different hand. Moreover, their success criterion was the number of times an object passes through a target pose, and they never trained their controller to stop the object at the target pose, which we experimentally found harder to learn. In the broader context of manipulation, the ability to stop at the target pose is vital: If the robot uses a tool, it must reorient it to the desired pose and hold the tool in that pose.

The focus of our work is not to increase the reorientation performance on a single object; rather, our work expands the scope of object reorientation to operate in more general and pragmatic settings. The result is a single controller for reorienting multiple objects, evidence of some generalization to new objects, and dynamic reorientation in the air without a highly specialized perception system. At the same time, there remains ample scope for improving performance, and we hope that our conscious use of open-source hardware, commodity sensing, computing, and fast-learning framework (Figure 6 and Figure 7) will facilitate future research in enhancing performance and comparing results.

2.3 Discussion

Solving contact-rich tasks typically requires optimizing the location at which the robotic manipulator contacts the object [4], [61], [62]. One would assume predicting the contact location requires knowledge of the object’s shape. However, inputs to the teacher policy have no information about object shape, yet it could reorient diverse and new objects. One possibility is that the agent gathers shape information by integrating information across the sequence of touches made by the fingers. However, the teacher policy is not recurrent, ruling out this possibility. The surprising observation of reorientation without knowledge of shape was made by earlier work in the context of a reorientation system in simulation [18]. However, because real-world results were not demonstrated, it remained unclear if such an observation was an artifact of the simulator or the property of the reorientation problem. With real-world evaluation, we have more confidence that shape information may not be as critical to object reorientation as one might apriori think. However, this is not to suggest that shape is not useful at all. The results show that one can go quite far without shape information, but the performance, especially on precise manipulation and in generalization to new shapes, can likely be improved by incorporating shape features into the teacher policy, an exciting direction for future research.

Typically, having more fingers introduces more optimization variables, making the optimization problem harder in the conventional view. However, we have some evidence to the contrary (Figure 4B). Having more fingers can make it easier for deep reinforcement learning to find a solution, especially in challenging manipulation scenarios such as in the air, similar to how over-parameterized deep networks find better solutions (a conjecture). We conjecture that over-parameterized hardware results in a larger pool of good solutions (more ways to reorient an object with more fingers), making it easier for current optimizers in deep learning to find a good solution.

In designing the proposed system, we either devised or made several technical choices:

two-stage student training, representing both the camera recordings and proprioceptive readings as a point cloud, sparse convolution neural network for real-time control, limited range of domain randomization due to system identification, system identification using parallel GPU simulation, use of soft material on fingertips, using a larger number of fingers instead of the conventional wisdom of using fewer fingers. These choices, however, are not specific to in-hand reorientation but can be applied to a broad spectrum of vision-based manipulation tasks involving rigid bodies. We hope that the knowledge of these choices, along with a low-cost platform, can further the goal of democratizing research in dexterous manipulation.

Limitations and Possible Extensions Object reorientation with a downward-facing hand has notable room for improving precision and reducing the drop rate. We hypothesize that one possible cause for dropping objects is that the control frequency of 12Hz is not fast enough. The robot dynamically manipulates the object, and it takes a fraction of a second to lose control. It might be challenging to determine when the object is slipping from the fingers in real-time using visual feedback at 12Hz. Feedback control at a higher frequency may mitigate such failures but either requires more efficient neural network architectures or more processing power.

Another hypothesis for object dropping is missing information regarding whether the finger is in contact with the object, if the object is slipping, or how much force is being applied. We conjecture that explicit knowledge of contact, contact force, and other signals such as slip can substantially improve performance. Currently, the robot relies purely on occluded vision observations to infer contacts. Augmenting the robot’s observation with touch sensors is therefore an exciting direction for future investigation.

We also found that inaccurate prediction of rotational distance is another cause for imprecise object reorientation. The prediction of rotational distance is less accurate when the actual rotational distance is less than 0.4 radians (see Discussion on precise manipulation in Supplementary Discussion).

We hypothesize that generalization and precision can be improved by training on a larger object dataset, investigating RGB sensing to complement depth sensing to capture fine geometric structures and reduce noise, and integrating visual and tactile sensing to obtain more complete point clouds. Further, there remains a sim-to-real gap that future research should investigate.

We used D’Claw manipulators in this work as it is open-source and low-cost. However, many aspects of the D’Claw, such as the finger design and the number of fingers, are sub-optimal. For instance, although we observed some robustness to the softness of fingertips, different softness and skeleton designs can notably affect the longevity of fingertips. We manually iterated over many soft fingertip designs, which was time-consuming. Similarly, the fingertips have a hemispherical shape, quite different from humans and presumably not optimal. The performance of the task can be improved by better hardware design: the shape of fingers, the degrees of actuation on each finger, the placement of fingers, and the choice of materials. Manually iterating over these choices is infeasible. A promising future direction is to utilize a computational approach for automatically designing the hand for specific tasks [63].

In summary, we presented a real-time controller that can dynamically reorient complex and new objects by any desired amount using a single depth camera. The system is both simple and affordable, which aligns with the objective of making dexterous manipulation research accessible to a wider audience.

2.4 Materials and Method

Given a random object in a random initial pose, the robot is tasked to reorient the object to a user-provided target orientation in $SO(3)$ space. We train a single vision-based object reorientation controller (or policy) in simulation to reorient hundreds of objects. The controller trained in simulation is directly deployed in the real world (zero-shot transfer). The choices in our experimental setup have been made to support future deployment of reorientation in service of tool use and on a mobile manipulator.

Object datasets We use two object datasets in this work: **Big dataset** (\mathbb{B}) and **Small dataset** (\mathbb{S}). \mathbb{B} contains 150 objects from internet sources. \mathbb{S} contains 12 objects from the ContactDB [64] dataset. These two datasets do not have overlapped shapes. More details on the object dataset are in Supplementary Methods.

Simulation setup We use Isaac Gym [30] as the rigid body physics simulator. We train all the policies on a table-top setup: hands face downward with a supporting table.

Success criteria During training, the success criterion for reorienting an object acts as both a reward signal and a criterion for success to end the episode. A straightforward success criterion is judging whether an object’s orientation is close to the target orientation (orientation criterion). However, a controller trained using this criterion tends to cause the object to oscillate around the target orientation. To address this issue, the success criterion is expanded to explicitly penalize finger and object movements. For further details on how we designed the success criteria for training, please refer to Supplementary Methods.

2.4.1 Training the visuomotor policy

We model the problem of learning the controller, π , as a finite-horizon discrete-time decision process with horizon length T . The policy π takes as input sensory observations (\mathbf{o}_t) and outputs action commands (\mathbf{a}_t) at every time step t . Learning π using RL is data inefficient when the observation (\mathbf{o}_t) is high-dimensional (for example, point clouds). The reason is that the policy needs to simultaneously learn which features to extract from visual observations and what are the high-rewarding actions. The problem would be simplified if one of these factors were known: learning a policy via RL from sufficient state information would be much easier than direct learning from sensory observations. Similarly, apriori knowledge of high-rewarding actions would reduce the data requirements of learning from visual observations.

Prior work has employed this intuition to ease policy learning by decomposing the learning process into two steps [18], [32], [54], [56]. In the first step, a teacher policy is trained in simulation with RL using low-dimensional state space that includes privileged information.

In the case of in-hand object reorientation, privileged information includes quantities such as fingertip velocity, object pose, and object velocity that can be directly accessed from the simulator but can be challenging to measure in the real world. Because the teacher policy operates from a low-dimensional state space, it can be more efficiently trained using RL. Next, to enable operation in the real world, one can either train a perception system to predict the privileged information [15], [53] or train a second student policy to predict high-rewarding teacher actions from raw sensory observations via supervised learning [18], [32], [54], [56].

An underlying assumption of the two-stage training paradigm is that a low-dimensional state for learning a teacher policy can be identified. Because there are no tools available to theoretically analyze if a particular choice of state space is sufficient for policy learning, selecting the state inputs for the teacher policy is a manual process based on human intuition. At first, object reorientation might seem to require knowledge of object shape since the controller must reason about where to make contact. If object shape is necessary, then it will not be possible to reduce depth observations into a low-dimensional state. However, past work found that even without any shape information, it is possible to train RL policies to achieve good reorientation performance on a diverse set of objects in simulation [18]. Therefore, teacher-student training can be leveraged to simplify the learning of object reorientation.

To deploy the policy in the real world, some prior works train a perception system to predict the object pose [15], [16]. However, object pose is only defined with respect to a particular reference frame. Choosing a common frame of reference across different objects is not possible. As a consequence, pose estimators cannot generalize across objects. Therefore, we choose to train an end-to-end student policy that takes as input the raw sensory observations and is optimized to match the actions predicted by the teacher policy via supervised learning [25]. Because supervised learning is considerably more data efficient than RL, such an approach solves the hard problem of learning a policy from raw sensory observations.

The teacher-student training paradigm has been used to learn object reorientation policy in simulation from visual and proprioceptive observations [18]. However, a separate policy was trained per object. Secondly, it required more than a week to train the student vision policy for a single object on an NVIDIA V100 GPU. We developed a two-stage student training (Teacher-student²) framework (Figure 6) that substantially speeds up the vision student policy learning. Using this framework, we were able to learn a vision policy that operates across a diverse set of objects and generalizes to objects with different shapes and physical parameters.

Teacher policy: reinforcement learning with privileged information

The learning of teacher policy ($\pi^\mathcal{E}$) is formulated as a reinforcement learning problem where the robot observes the current observation ($\mathbf{o}_t^\mathcal{E}$), takes an action (\mathbf{a}_t), and receives a reward (r_t) afterward. A single policy ($\pi^\mathcal{E}$) is trained across multiple objects using proximal policy optimization (PPO) [37] to maximize the expected discounted episodic return: $\pi^{\mathcal{E}*} = \arg \max_{\pi^\mathcal{E}} \mathbb{E} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]$. Since the observation \mathbf{o}_t at a single time step t does not convey the full state information such as the geometric shape of an object, our setup is an instance of Partially Observable Markov Decision Process (POMDP). However, for the sake of simplicity and based on the finding that knowledge of object shape may not be critical

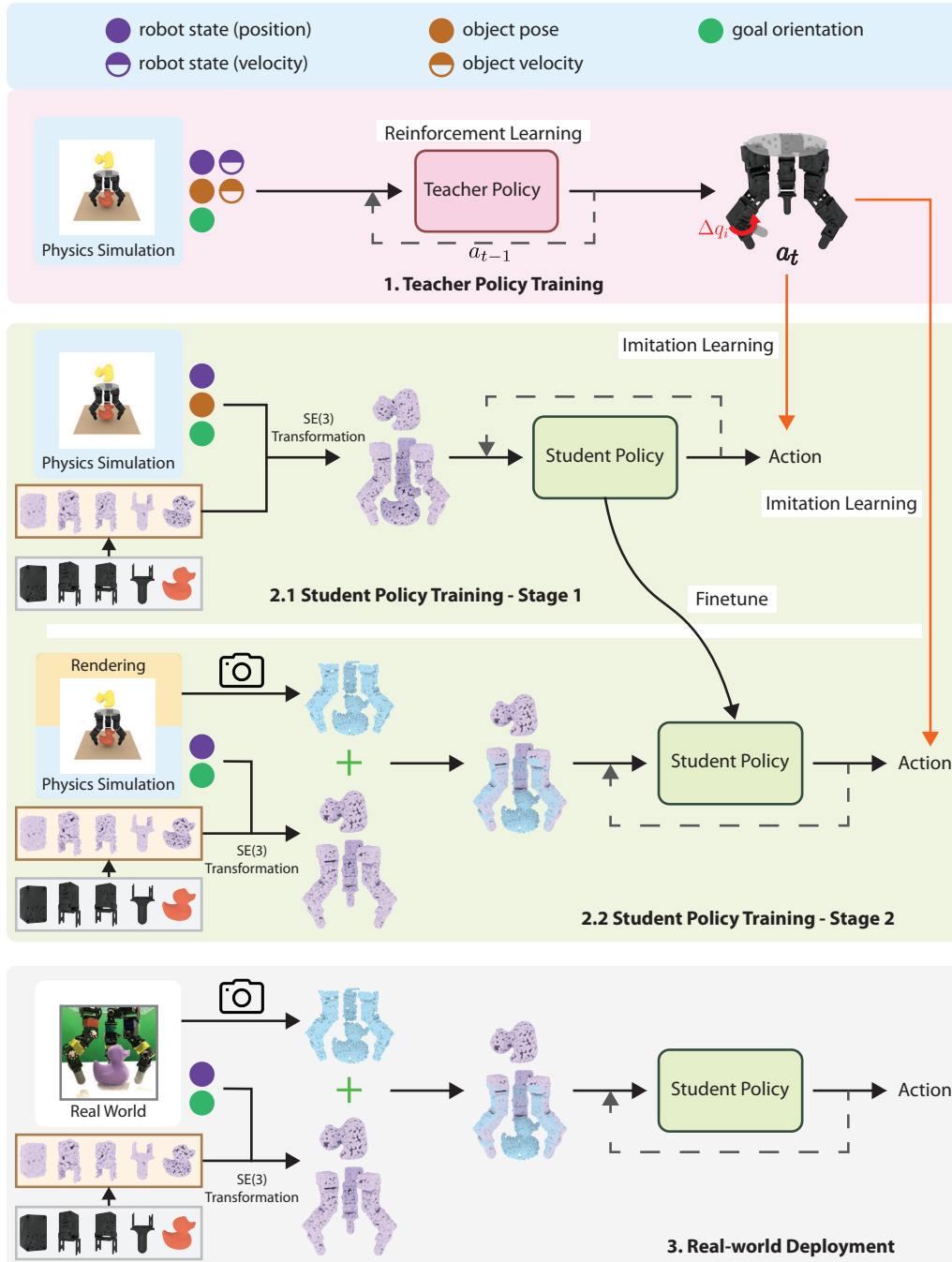


Figure 2.6: **Teacher and two-stage student training framework.** First, a teacher policy is trained using reinforcement learning with privileged state information. Then, a student policy is trained to imitate the teacher using synthetic and complete point clouds as input. The student policy is further fine-tuned using rendered point clouds. During deployment, the student policy can be directly used to control real robots.

as discussed above, we chose to model the policy as a Markov Decision Process (MDP): $\mathbf{a}_t = \pi^{\mathcal{E}^*}(\mathbf{o}_t; \mathbf{a}_{t-1})$. The policy also takes as input the previous action (\mathbf{a}_{t-1}) to encourage

smooth control.

Observation space The inputs to the teacher policy, \mathbf{o}_t , include proprioceptive state information, object state, and target orientation. Details are shown in Supplementary Methods.

Action space We use position controllers to actuate the robot joints at a frequency of 12Hz. The policy outputs the relative joint position changes $\mathbf{a}_t \in \mathbb{R}^{3G}$. Instead of directly using \mathbf{a}_t , we use the exponential moving average of actions $\bar{\mathbf{a}}_t = \alpha\mathbf{a}_t + (1 - \alpha)\bar{\mathbf{a}}_{t-1}$ for smooth control, where $\alpha \in [0, 1]$ is a smoothing coefficient. In our experiments, we set $\alpha = 0.8$. Given the smoothed action $\bar{\mathbf{a}}_t$, the target joint position at the next time step is: $\mathbf{q}_{t+1}^{tgt} = \mathbf{q}_t + \bar{\mathbf{a}}_t$.

Reward We first describe the reward function for the hand to reorient objects on a table. The first term in the reward function (Equation 1) is the success criteria for the task. However, since this only provides sparse reward supervision, the criteria by itself is insufficient for successful learning. Therefore we add additional reward shaping [65] terms to encourage reorientation. We use a dense reward term that encourages minimization of the distance ($\Delta\theta_t$) between the agent’s current and target orientation (Equation 2). We penalize the agent for moving fingertips far away from the object (Equation 3). Without this term, fingers barely made any contact with the object during training. We also penalize the agent for expending energy (Equation 4) and for pushing the object too far from the robot’s hand (Equation 5) in which case the episode is also terminated. The reward terms are mathematically expressed as:

$$r_{1t} = c_1 \mathbb{1}(\text{Task successful}) \quad \text{sparse task reward} \quad (2.1)$$

$$+ c_2 \frac{1}{|\Delta\theta_t| + \epsilon_\theta} \quad \text{dense task reward} \quad (2.2)$$

$$+ c_3 \sum_{i=1}^G \left\| \mathbf{p}_t^{f_i} - \mathbf{p}_t^o \right\|_2^2 \quad \text{keep fingertip close to the object} \quad (2.3)$$

$$+ c_4 |\dot{\mathbf{q}}_t|^T |\boldsymbol{\tau}_t| \quad \text{energy reward} \quad (2.4)$$

$$+ c_5 \mathbb{1}(\|\mathbf{p}_t^o\|_2^2 > \bar{p}) \quad \text{penalty for pushing the object away} \quad (2.5)$$

where $c_1, c_2 > 0$, and $c_3, c_4, c_5 < 0$ are coefficients, $\mathbb{1}$ is an indicator function, ϵ_θ and \bar{p} are constants, $\mathbf{p}_t^{f_i}$ is the fingertip position of i^{th} finger, \mathbf{p}_t^o is the object center position, $\boldsymbol{\tau}_t$ is the vector of the joint torques.

Using the aforementioned reward function, we were able to train reorientation policies that used the support of the table. Next, to enable the more challenging behavior of reorienting objects in the air, we added a penalty for the contact between the object and table (Equation 7) and a penalty for using the penultimate joint instead of the fingertip for reorientation (Equation 8). Although the term in Equation 8 is not critical, it results in more natural-looking behaviors. The overall reward function is:

$$r_{2t} = r_{1t} \tag{2.6}$$

$$+ c_6 \mathbb{1}(\text{object contacts with the table}) \tag{2.7}$$

$$+ c_7 \sum_{i=1}^N \mathbb{1}(p_{t,z}^{f_i} > \bar{p}_z) \tag{2.8}$$

where $c_6, c_7 < 0$ are coefficients.

Student policy - imitation learning from depth observations

The student policy (π^S) is trained in simulation with the purpose of being deployed in the real world. Since the sim-to-real gap for depth data is less pronounced than RGB data, we only use the depth images provided by the camera along with readings from joint encoders. We represent the depth data as a point cloud in the robot’s base link frame. To enable the neural network representing π^S to model the spatial relationship between the fingers and the object, we express the robot’s current configuration by showing the policy a point cloud representing points sampled on the surface of the fingers. We concatenate the point cloud obtained from the camera along with the generated point cloud of the hand. We denote this scene point cloud as \mathbf{P}_t^s .

Goal representation Instead of providing the goal orientation as a pose which has generalization issues discussed above, the goal is represented as the object’s point cloud in the target orientation \mathbf{P}^g . In other words, the policy sees how the object should look in the end (see the top left of Figure 7A).

Observation space The input to π^S is the point cloud $\mathbf{P}_t = \mathbf{P}_t^s \cup \mathbf{P}^g$ (see Figure 7A). We also did an ablation study on different ways to process the goal point cloud in Supplementary Discussion S5.4. The results show that merging \mathbf{P}_t^s and \mathbf{P}^g before they are input to the network leads to faster learning.

Architecture The critical requirement for the vision policy is to run at a high enough frequency to enable real-time control. For fast computation, we designed a sparse convolutional neural network to process point cloud (\mathbf{P}_t) using the Minkowski Engine [66] (see Figure 7A). Compared to the architecture used in [18], our convolutional network has a higher capacity to make it possible to learn the reorientation of multiple objects. Without direct access to object velocity, it is necessary to integrate temporal information in π^S , for which we use the gated recurrent unit [67] in the network.

Optimization The student policy π^S is trained using DAGGER [25] to imitate the teacher policy π^E .

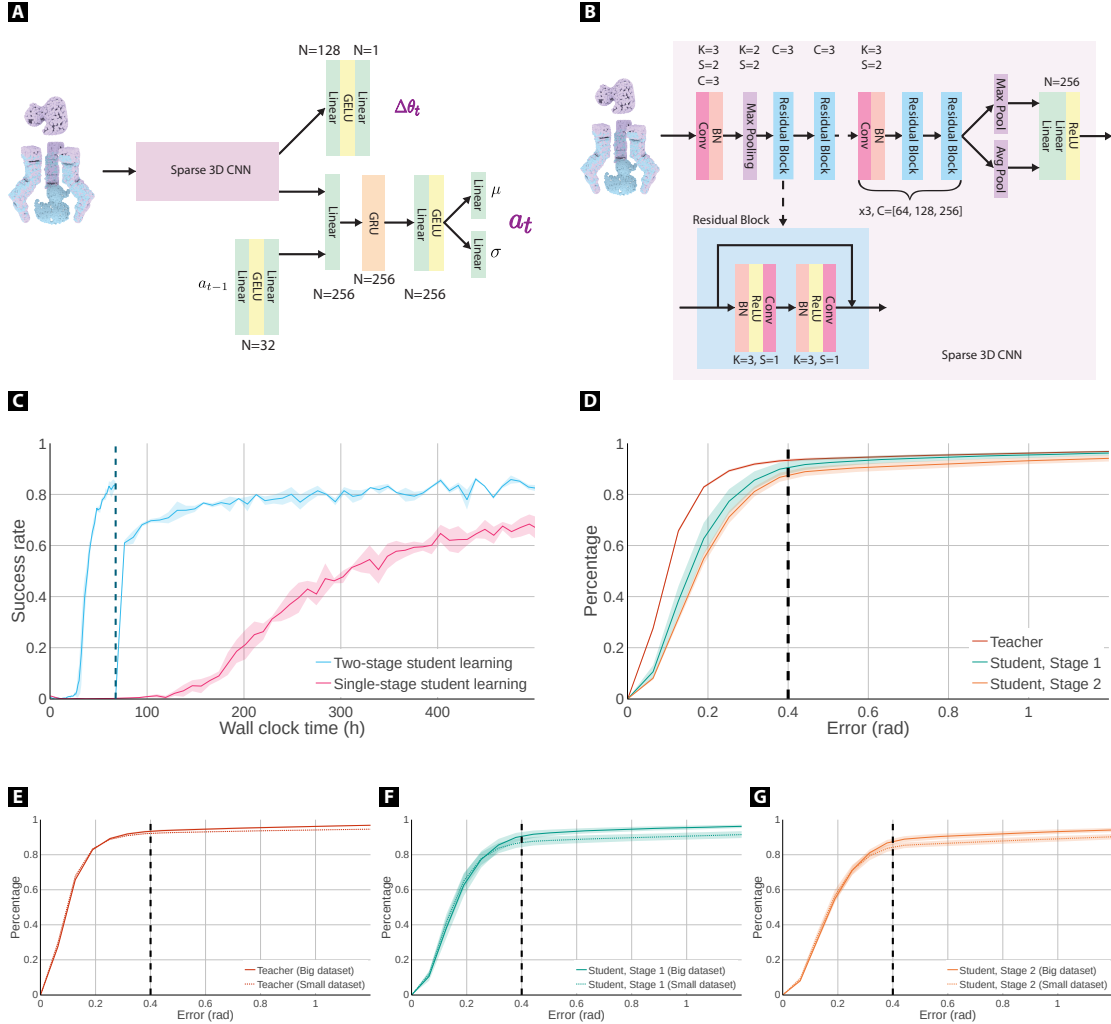


Figure 2.7: **Student policy learning.** (A): Student vision policy network architecture. (B): Sparse 3D CNN (Convolutional Neural Network) component of the policy network. (C): Proposed two-stage student learning learns faster than single-stage student learning. The dashed vertical line denotes the transition from the first to the second stage of student learning. The performance dip happens due to a change in the distribution of point cloud inputs from being unoccluded in the first stage to being occluded in the second. (D): Post-training evaluation of teacher and student policies on the training dataset \mathbb{B} . For each object, the initial and target orientations are randomly sampled 50 times, resulting in 7500 samples. The empirical cumulative distribution function (ECDF) of the orientation error is plotted. The results show that the students are close to the teacher’s performance. (E), (F), (G): Comparing the ECDFs of the policies being evaluated on dataset \mathbb{B} and dataset \mathbb{S} reveals small generalization gap for all the policies.

Need for two-stage student learning We found training a vision policy in simulation to be slow, consuming 20+ days on an NVIDIA V100 GPU (Figure 7C). The main reason for slow training is that the simulator performs rendering to generate a point cloud which consumes a substantial amount of time and GPU memory. To reduce training time, we

generated synthetic point clouds by uniformly sampling points on the object and robot meshes used by the simulator. The synthetic point cloud is also complete (no occlusions), which makes training easier. The vision policy (π_1^S) can be trained with synthetic point cloud in less than three days, which is a $7\times$ speedup (**stage 1**; see Figure 7C). However, the policy, π_1^S , cannot be deployed in the real world because it operates on an idealized point cloud (no occlusions). Therefore, once the student reaches high performance, we initiate **stage 2**, where the policy is finetuned with the rendered point cloud. Such finetuning is quick in wall-clock time (around one day), and the resulting policy (π_2^S) performs better than training from scratch with rendered point clouds (see Figure 7C). It is possible to further reduce the training time of the student policy by employing visual pre-training with passive data that we discuss in Supplementary Discussion S5.5. An additional benefit of the two-stage student policy training is that π_1^S is agnostic to the camera pose. Therefore a policy from a new viewpoint (π_2^S) can be quickly obtained by finetuning using rendered point clouds from that camera pose. Training the vision policy from scratch is not necessary.

Stage 1: details of synthetic point cloud In stage 1, the simulation is not used for rendering but only for physics simulation. We generate the point cloud for each link on the manipulator and object by sampling K points on their meshes in the following way: let the point cloud of link l_j in the local coordinate frame of the link be denoted as $\mathbf{P}^{l_j} \in \mathbb{R}^{K \times 3}$. Given link orientation ($\mathbf{R}_t^{l_j} \in \mathbb{R}^{3 \times 3}$) and position ($\mathbf{p}_t^{l_j} \in \mathbb{R}^{3 \times 1}$) at time step t , the point cloud can be computed in the global frame, $\mathbf{P}_t^{l_j} = \mathbf{P}^{l_j}(\mathbf{R}_t^{l_j})^T + (\mathbf{p}_t^{l_j})^T$. The point cloud representation of the entire scene is the union of point clouds of all the links, the object being manipulated, and the object in the goal orientation: $\mathbf{P}_t^s = \bigcup_{j=1}^{j=M} \mathbf{P}_t^{l_j}$ where M is the total number of links (bodies) in the environment. The point cloud \mathbf{P}_t^s can be efficiently generated using matrix multiplication.

Stage 2: details of rendered point cloud In stage 2, at each time step, we acquire depth images from the simulator and convert them into point clouds (which we call exteroceptive point cloud) using the camera’s intrinsic and extrinsic matrices. Note that such a point cloud is incomplete due to occlusions. We also convert the joint angle information into poses of the links on the robot hand via forward kinematics and then generate the complete point cloud of the robot (which we call proprioceptive point cloud). Note that such a proprioceptive point cloud of a robot can be easily obtained in the real world in real-time from the joint position readings. The policy input is the union of the exteroceptive and the proprioceptive point cloud.

2.4.2 Reducing the simulation to reality gap

There are two main sources of the gap between simulation and reality. The first one is dynamics gap that arises from differences in the robot dynamics, the approximation in the simulator’s contact model, and differences in object dynamics that depends on material properties such as friction. The other source is perception gap caused by differences in statistics of sensor readings and/or noise. One way to reduce these gaps is to train a single policy across many different settings of the simulation parameters (domain randomization [36]). The success of

domain randomization hinges on the hope that the real world is well approximated by one of the many simulation parameter settings used during training. The chances of such a match increase by randomizing parameters over a larger range. However, excessive randomization may result in an overly conservative policy with low performance [68]. Therefore, we make design choices that reduce the need for domain randomization and use it only when needed.

The perception gap is reduced by using only depth readings, which is more similar between simulation and reality than RGB. To account for noisy depth sensing, we add noise to the simulated point cloud. The dynamics gap can be reduced by identifying simulation parameters closest to the real world. While such identification is possible for the robotic manipulator, it is infeasible for object dynamics that vary in material and mass distribution. Therefore, we perform system identification on the robot dynamics and use only small randomization to account for unmodeled errors. We use a larger range of domain randomization on the object and environment dynamics. To make the policy more robust to unmodeled real-world physics, we apply random forces on the object during training which pressures the policy to reorient objects while being robust to external disturbance. Lastly, to increase compliance and friction between the object and the manipulator, we use soft fingertips. Such a choice makes the system more tolerant of errors in control commands. Empirically we noticed that soft fingertips make the robot less aggressive and reduce overshoot.

Identification of robot dynamics

We build the Unified Robot Description Format (URDF) model for the manipulator using its CAD model, which provides accurate kinematics parameters, but the dynamics parameters, such as joint damping and stiffness, must be estimated. One way of identifying dynamics parameters is to leverage the equations of motion (or the dynamics model) and solve for the unknown variables using a dataset of motion trajectories. The Isaac Gym simulator has a built-in dynamics model. But because the simulator’s code is not open-source, we do not have access to the precise dynamics model nor the gradients of dynamics parameters. We, therefore, used a black-box approach that leverages the ability of Isaac Gym to perform massively parallel simulations. We spawn many simulations with different dynamics parameters and use the one that has the closest match to the real robot’s motion.

Let $\lambda_i \in \Lambda$ denote the dynamics parameter of the i^{th} simulated robot (C^{λ_i}), where Λ denotes the entire set of dynamics parameter values over which search is performed. To evaluate the similarity between the motion of C^{λ_i} and the real robot (C^{real}), we compute the score: $h(\mathbf{q}_A^{C^{real}}(\cdot), \mathbf{q}_A^{C^{\lambda_i}}(\cdot)) = -\left\| \mathbf{q}_A^{C^{real}}(\cdot) - \mathbf{q}_A^{C^{\lambda_i}}(\cdot) \right\|_2^2$, where $\mathbf{q}_A^C(\cdot)$ represents the joint position trajectories of a robot C given action commands $\mathbf{A}(\cdot)$ which are detailed in Supplementary Methods. The closer the motion of the simulated robot is to that of the real world, the higher the score will be. We use the black-box optimization method of Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [69], an instance of evolutionary search algorithms, to determine the optimal dynamics parameter: $\lambda^* = \arg \max_{\lambda \in \Lambda} h(\mathbf{q}_A^{C^{real}}(\cdot), \mathbf{q}_A^{C^\lambda}(\cdot))$. Note that it might be impossible to find a simulated robot that exactly matches the real robot due to the approximate parameterization of real-world dynamics in simulation and the stochasticity in the real-world resulting from actuation/sensing noise. More details on the identification are in Supplementary Methods.

2.4.3 Real-world deployment

Real-world observation It includes the joint positions of each motor in the manipulator and the depth image from a RealSense camera. Details how the joint positions and depth image are converted into a unified point cloud input can be found in Supplementary Methods.

Stopping criteria To automatically stop the robot, we train a predictor that re-uses features from the policy network to predict $|\Delta\theta_t|$ (see Figure 7A). The robot is stopped when $\Delta\theta_t^{pred} < \bar{\theta}$ and $\|\mathbf{a}_t\| < \bar{a}$.

More details on the stopping criteria, the real-world experimental setup, and the procedure for quantitative evaluation are in Supplementary Methods.

2.5 Acknowledgements

I want to thank the people who helped me with this project: Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. They all did important work that made it possible for the project to be finished successfully.

Chapter 3

Vegetable Peeling: A Case Study in Constrained Dexterous Manipulation

3.1 Introduction

Having robots perform food preparation tasks has been of great interest in robotics. Imagine the scenario of making mashed potatoes, where a critical step is to peel potatoes. Humans peel potatoes by grasping the potato in one hand and using the second hand to actuate a peeler to remove the potato’s skin. After a part of the potato is peeled, it is rotated while being held in the hand (i.e., *in-hand manipulation*) and peeled again. The sequence of rotating and peeling continues until all of the potato’s skin is removed. In this work, we present a robotic system that can re-orient different vegetables using an Allegro hand in a way that their skin can be peeled using another manipulator. Our setup is shown in Figure 3.1 and Figure 3.2.

In-hand rotation of vegetables is an instance of dexterous manipulation problem [7], a family of tasks that involves continuously controlling the force on an object while it is moving with respect to the fingertips [3], [39]. The challenges in dexterous manipulation stem from the frequent making and breaking of contact, issues in contact modeling, high-dimensional control space, perception challenges due to severe occlusions, etc. A body of work made simplifying assumptions such as manipulating convex objects [4]–[7], small finger motions [49], [50], [62], slow or quasi-static motion or manipulating a few specific objects [12], [50], [62] to leverage trajectory optimization or planning-based methods to achieve in-hand object re-orientation [4]–[7], [12], [49], [50], [62]. Another line of work has used reinforcement learning for in-hand re-orientation [18], [19], [46], [70], [71] and recent works have leveraged simulation to train policies capable of dynamically re-orienting a diverse set of new objects in real-time and in the real world [18], [19].

There are several challenges in adapting re-orientation controllers for a downstream task such as peeling vegetables. These challenges stem from the fact that controllers optimized for re-orientation [15], [19], [46], [70], [71] are only optimized to continuously reorient the object and not to satisfy numerous constraints arising from task-specific requirements. For instance, peeling vegetables requires the hand to **first stop** re-orienting the object and then for the peeler to peel the vegetable. Many prior works solve a version of the re-orientation problem

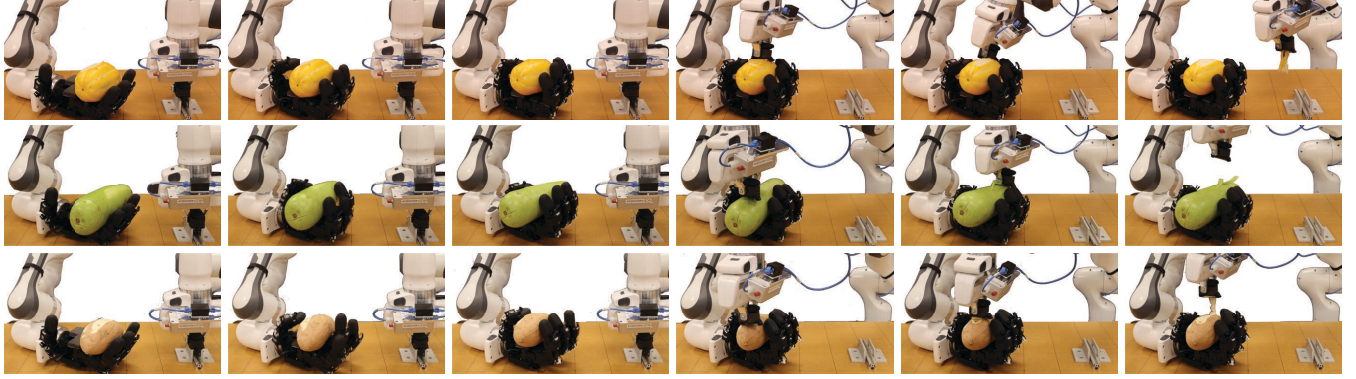


Figure 3.1: We present a dexterous manipulation system that utilizes an Allegro hand mounted on a Franka robot arm to reorient food items for downstream peeling. The other Franka robot arm (the right arm in the figure) uses its gripper to grasp a peeler for peeling. The reorientation controller for the Allegro hand is learned through reinforcement learning, while the peeling is performed via teleoperation. In the figure, we demonstrate the process of reorienting and peeling a melon, a sweet potato, and a squash from top to bottom row.

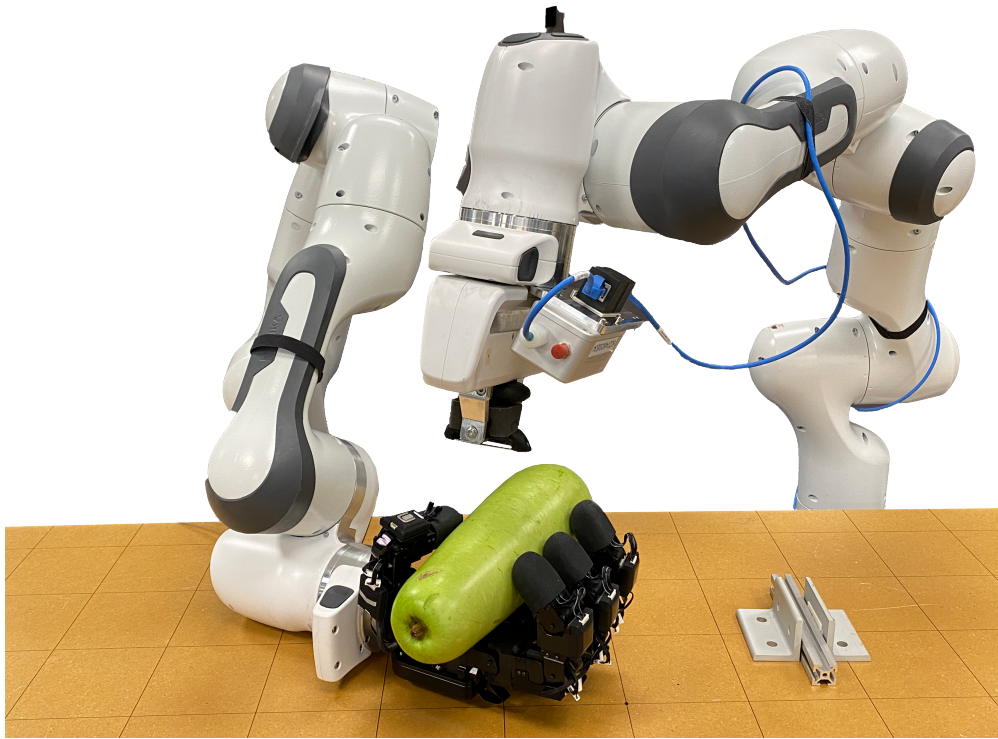


Figure 3.2: Robot setup for reorientation and peeling.

where the object is continuously rotated [15], [70], [72] or otherwise perform quasistatic re-orientation [50]. Stopping and re-starting *dynamic* re-orientation is difficult due to the challenge of dealing with the object’s inertia. **Second**, the hand needs to *hold* the object firmly enough to resist forces applied by the peeler. The closest work that attempts to hold the object at a target configuration [19] is only able to loosely hold the object which

is insufficient for resisting forces. **Third**, the hand needs to reorient the vegetable *along a specific axis in place*. Here, the specific axis refers to the rotational axis on the object that is parallel to the peeling direction. Similar to how humans reorient vegetables for peeling, it is desirable for the hand to reorient the object in place so that multiple consecutive cycles of reorientation and peeling can be performed. If the object substantially shifts its position during reorientation, the controller will struggle to reorient and hold the object at future time steps. **Fourth**, when the vegetable is held stationary the fingers should *not obstruct* the top surface of the vegetable to ensure that the peeler can peel the vegetable.

While in-hand object reorientation has been widely studied [15], [18], [19], [53], [70], [72], no prior works can satisfy the constraints mentioned above. Yet, these constraints become critical for downstream dexterous manipulation beyond object re-orientation. We use vegetable peeling as a case study to investigate the challenges and solutions for building a dexterous manipulation system that can operate under constraints. We develop a framework where we leverage reinforcement learning in simulation to train a policy that can perform object re-orientation under constraints. For the peeling task, we explored two approaches - a teleoperation-based method leveraging human guidance as well as an autonomous vision-based technique. Our contributions are as follows:

1. A framework for solving dexterous manipulation problems under the aforementioned constraints.
2. We propose a method that can make RL policy learn to stop its motion and hold objects firmly in hand – a critical behavior for many downstream dexterous manipulation problems.
3. We present a step towards a robotic system capable of peeling diverse vegetables with different shapes, masses, and material properties while holding and manipulating the vegetables in hand.

3.2 Related work

In-hand Object Reorientation: Dexterous manipulation involves the use of high degrees-of-freedom (DoF) manipulators for object manipulation [2]. Its requirement for high-dimensional real-time control and its nature of frequent contact-making and breaking present grand challenges to roboticists. Recently, there has been a growth of interest in a particular instance of dexterous manipulation problems: in-hand object reorientation. This problem is of particular interest as it is a necessary step in many tool-use scenarios. For example, to use a screwdriver for tightening a screw, one has to reorient the screwdriver to align it with the screw. We can cluster the works in in-hand object reorientation from many aspects. For example, from the perspective of sensory information, [43] studies open-loop cube reorientation without using any sensors, [6], [11], [12], [15], [73] use motion capture system or special tracking markers for object reorientation, [72] uses proprioceptive sensors such as joint encoders, [17], [46], [47], [71] use tactile sensors and [15], [19], [48], [53] utilize vision sensors. In terms of the dynamics of the system, [49], [50], [62] achieved object reorientation under the assumption of quasi-static motion where object moves slowly and

its inertia effect can be ignored, while [15], [19], [42], [46], [71] focuses on dynamic object reorientation where object is manipulated in a fast and dynamic way. To make in-hand object manipulation useful for downstream tool use tasks, one important aspect of the skill is the ability of stably and firmly holding the object in end of the policy rollout. While many prior works on dynamic manipulation such as [12], [15], [46], [71], [72] only consider endlessly rotating the object in hand and cannot stop the object stably when the object reaches the goal orientation, some works such as [19], [42] try to develop controllers that can reorient objects in hand and also hold the object in the goal orientation. Our work studies dynamic in-hand object manipulation with the capability of stopping objects stably in hand.

Reinforcement Learning for Contact-rich Tasks: Contact-rich tasks are particularly challenging due to the difficulty in modeling the system dynamics, especially when the tasks are performed in the wild, outside of a constrained and controlled setting. Examples of such tasks include quadruped robots hiking in mountains and robot hands reorienting various everyday objects. There have been many works using reinforcement learning to learn controllers for solving contact-rich tasks [15], [16], [68], [70], [74]–[76]. In the real world, robots typically only have access to a limited amount of state information of the system due to the lack of sensors or the challenges in setting up the sensors. Using reinforcement learning to learn controllers from scratch with limited sensory information tends to be data-inefficient. One way to speed up policy learning is to provide asymmetric information to the policy and value function, where the value function observes much more privileged information [15], [16], [70], [77]. Another method is to decouple policy learning into two stages: a reinforcement learning stage where agents (teacher) observe privileged fully-observable state information, and an imitation learning stage where the policy with limited sensory observation input (student) learns to imitate the policy with fully-observable state information. This approach has been successfully applied to various contact-rich problems such as locomotion [33], [54]–[56], [75] and dexterous manipulation [18], [19], [72]. Our pipeline is built upon the idea of teacher-student policy learning and has made several key improvements, which we will detail below.

3.3 Method

Peeling requires a reorientation controller that can stop its motion and firmly hold objects after reorientation. The first step in stopping is to decide when re-orientation should be stopped. One possibility is to have a perception system predict the desired rotation angle after which the next round of peeling would be performed. To accomplish the goal, the robot would need to track changes in object pose and compare it with the target rotation angle. However, accurately estimating object pose is challenging, especially when generalization to new objects is necessary [15], [70], [76], [78].

One of our insights is that instead of training a predictor for desired rotation angle and object pose estimation, it can be *easier* and *sufficient* to train a *binary vision classifier* that detects in real-time when the peeled part has been turned over. With such a classifier, the reorientation controller’s job is simply to keep reorienting the object until it receives a stop signal. In this formulation, unlike prior works [18], [19], the reorientation controller is not conditioned on target orientation but rather on a stop signal. Formally, the policy takes as

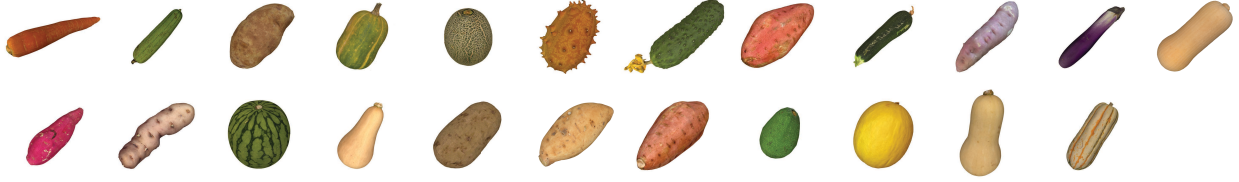


Figure 3.3: Object dataset used in this work. We collected meshes of carrot, sweet potato, potato, squash, pumpkin, etc.

input a binary variable $I_t^{stop} \in \{0, 1\}$ representing the stop signal. If $I_t^{stop} = 1$, the policy should stop immediately and ensure the fingers stably and firmly hold the object. Otherwise, the policy should continue reorienting the object. Note that in this work, we focus on learning the reorientation controller, leaving integration of a vision classifier to future work.

The next question is how to train such a policy. Using RL to train the policy from scratch can be challenging and requires extensive reward shaping because $I_t^{stop} = 1$ is a rare event in an episode, and when the I_t^{stop} is flipped to one from zero, the policy needs to quickly stop the motion posing a hard-exploration challenge.

Prior works [18], [19] show success in training a goal-conditioned object reorientation controller. Can we leverage a goal-conditioned reorientation controller to train a controller that reacts to a stop signal? It turns out we can formulate this using the teacher-student learning framework [18], [19], [32], [54], [56]. Specifically, we can use RL to train a goal-conditioned controller that reorients an object by random goal angles along its rotational axis. This acts as the teacher. Next, we can use imitation learning (specifically DAGGER [25]) to train a controller conditioned on the stop signal to imitate the teacher. The stop signal can be generated during training by checking if the orientation distance to the goal is below a threshold. Using imitation learning bypasses the hard exploration challenge.

3.3.1 Training Setup

Robot We use an Allegro Hand that is controlled via a PD controller at 300Hz. Our control policy sets joint position commands and runs at a lower frequency at 12Hz.

Simulation We trained the policies in Isaac Gym simulation [30]. To set dynamics-related robot parameters in the simulation, we followed a prior approach [19], which uses a gradient-free search method to find the dynamics parameters for each joint (joint friction, damping, maximum joint velocity, and maximum effort) in simulation that generates the motor response that is closest to the real motors.

Object Dataset We collected 23 object meshes (potatoes, squash, cucumber, etc.) from Objaverse [79]. 10 variants for each mesh were created by varying the size. The mass of the object was randomly sampled in the range of [80, 960]g. Note that we aim to reorient much heavier objects than prior works [15], [18], [19], [70].

3.3.2 Teacher Policy Learning: Reorient and Stop

We train the teacher policy to re-orient the object along a pre-defined axis and stop (see Figure 3.4a). The teacher is formulated as a goal-conditioned policy $\mathbf{a}_t^{\mathcal{E}} = \pi^{\mathcal{E}}(\mathbf{o}_t^{\mathcal{E}}, \mathbf{a}_{t-1}, g)$, where \mathcal{E} represents variables for the teacher policy, \mathbf{o}_t is the observation, \mathbf{a}_t is the action command, g is the goal representing the amount by which the object needs to be re-oriented. g is randomly and uniformly sampled from $[1.57, 4.0]$ rad during training.

While the teacher policy’s formulation is similar to that in prior works [18], [19], we propose (i) a much simpler reward function, (ii) new success criteria that effectively encourages the policy to stop the object and firmly hold it, and (iii) an interpolation scheme that enables smoother policy actions in the real world.

Observation and Action Space

$\mathbf{o}_t^{\mathcal{E}}$ includes joint positions and velocities, the fingertip poses and velocities, object pose and velocity, the distance between the current object orientation and the goal orientation, and whether any of the fingertips touch the object. \mathbf{a}_t is the delta joint position command. The neural network policy runs at 12Hz.

Reward Function

A common approach to designing the reward function is to create multiple terms that make it easier for the manipulator to discover the desired behavior (i.e., reward shaping). For instance, to facilitate exploration, we can devise a reward term that reduces the distance between the fingertips and the center of mass (CoM) of the object. To discourage excessive translational motion of the object during rotation, we can create a reward term that penalizes the displacement of the CoM. To discourage the object from rotating with undesired motion along other axes, we can add another reward term that reduces the distance between the tip of the thumb and the centerline of the palm. This ensures that the thumb applies force close to the object’s CoM, rather than to one side of the object. Additionally, we need to design a reward term that discourages the fingers from covering the top surface of the object, which affects peeling. Hence, designing multiple reward terms is necessary to regulate the behavior under specific constraints. Balancing these terms requires extensive hyper-parameter tuning.

For the task of in-hand re-orientation, we found that the reward function can be substantially simplified by using a task demonstration. However, unlike prior works that rely on trajectory-level demonstrations [80], [81], our method only requires a *one-step demonstration* (a keyframe), which is much easier to collect. Specifically, we manually move the real Allegro hand to a good pose where the constraints mentioned above are satisfied (e.g., the fingers do not cover the food item), and the fingers touch the object and are ready to reorient it. We record the joint positions as \mathbf{q}^{demo} . During training in simulation, we encourage the joint positions at any time step to be close to \mathbf{q}^{demo} .

Overall, our reward function is as follows:

$$r_t = c_1 \mathbb{1}(\text{Task successful}) + c_2 \frac{1}{|\Delta\theta_t| + \epsilon_\theta} \tag{3.1}$$

$$+ c_3 \|\mathbf{q}_t - \mathbf{q}^{demo}\|_2^2 \tag{3.2}$$

where $c_1 = 800, c_2 = 1.5, c_3 = -0.6$ are coefficients. $\mathbb{1}(\text{Task successful})$ is 1 when the task is successfully completed, and 0 otherwise. $\Delta\theta_t$ is the distance between the object’s current and goal orientation. The first two terms are task rewards for object reorientation. The last term is to regulate hand behavior.

Success Criteria

In a goal-conditioned object reorientation, a common way to claim the task successful is by checking if the distance between the object’s current and the goal orientation is smaller than a threshold value (orientation criterion $C_{ori} = \Delta\theta < \bar{\theta}$) [15], [70]. Another criterion is that all the fingertips should make contact with the object (contact criterion $C_{contact}$), a pre-requisite for firmly holding the object after reorientation. However, only checking these two criteria is insufficient to ensure the policy learns to stop the motion and hold the object firmly around the goal orientation, as discussed in [19]. The policy can oscillate around the goal state due to observation and control delay and noise.

To further encourage the policy to stop robot motion when the goal is reached and firmly hold the object, we propose adding time constraints to the success criteria: both C_{ori} and $C_{contact}$ should be continuously satisfied for \bar{T}^{succ} time steps. Adding this criterion makes the MDP partially observable since the policy’s observation lacks the knowledge of time. Therefore, to facilitate policy learning, we augment the observation space with a scalar indicator variable $I^{succ} = t^{succ}/\bar{T}^{succ} \in [0, 1]$, where t^{succ} is the number of consecutive steps satisfying C_{ori} and $C_{contact}$. The observation space becomes $\mathbf{o}^{\mathcal{E}} := \mathbf{o}^{\mathcal{E}} \oplus I^{succ}$. In this work, $\bar{\theta} = 0.2\text{rad}$, $\bar{T}^{succ} = 8$.

Reset Constraints

As mentioned earlier, a reorientation policy for peeling needs to meet several constraints, such as in-place and fixed-axis reorientation (Figure 3.4b). While one could design individual reward terms to satisfy these constraints, tuning these reward terms to achieve the desired result can be difficult. Instead, it is much simpler to formulate the constraints as reset conditions. In other words, if the constraints are violated, the episode is reset immediately. This incentivizes the policy to explore only in space where the constraints are satisfied. Similar techniques were also used in some prior works [18], [19], [71].

Domain randomization and Perturbation during training

During training, we apply domain randomization on the joint stiffness and damping, friction, and restitution. In addition, we randomly apply a perturbation force on the object’s CoM. We randomly sample the direction of the perturbation force and set its magnitude to $10m_o$, where m_o is the object mass.

Interpolation and Reference for Action Commands

Our neural network controller operates at a relatively low control frequency of 12Hz. To track the joint position command, a low-level PD controller runs at 300Hz. To ensure smoother joint motion, we interpolate the low-frequency joint position commands. While

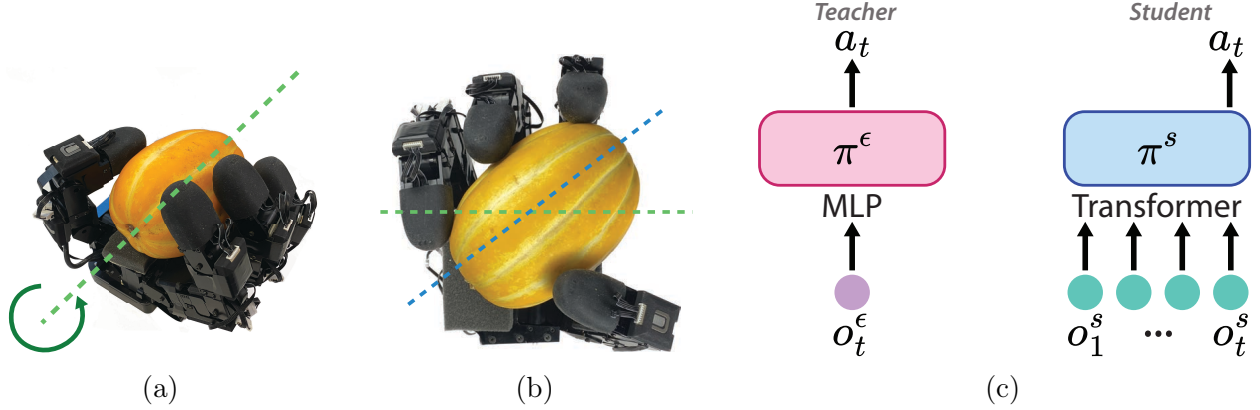


Figure 3.4: **(a)** shows an example of the rotational axis of a melon. **(b)** shows an example where the object’s orientation (the blue line) has a large deviation from the desired rotational axis (the green line). We reset the episode when this occurs. **(c)** shows the policy Architecture for the teacher and the student. In this figure, we use \mathbf{o}_t to represent all the policy input at each time step.

more complex interpolation schemes such as spline interpolation are possible, we found that simple linear interpolation is sufficient to generate smooth higher-frequency (60Hz) joint position commands. To do this, we linearly interpolate between the current reference joint positions (\mathbf{q}_t^{ref}) and the desired joint positions (\mathbf{q}_{t+1}^{cmd}) for the next policy control time step. We then send the interpolated joint position commands to the PD controllers. Mathematically, $\mathbf{q}_{t+1}^{cmd,n} = \mathbf{q}_t^{ref} + \frac{n}{N}\mathbf{a}_t$, where $n \in [1, N]$ ($N = 5$) and $\mathbf{q}_{t+1}^{cmd,n}$ represents the n^{th} interpolated joint position command for the next policy control time step.

When the action space is chosen as the change in joint position, the target joint position for the PD controller is calculated as follows: $\mathbf{q}_{t+1}^{cmd} = \mathbf{q}_t + \mathbf{a}_t$ [15], [18], [19]. Here, \mathbf{q}_t is the current joint position, and $\mathbf{a}_t = \Delta\mathbf{q}_t$ is the desired change in joint positions, as described earlier. In this case, the reference is chosen to be the current joint positions, i.e., $\mathbf{q}_t^{ref} = \mathbf{q}_t$. However, we found that this scheme results in significant jerky motion when combined with action interpolation. To illustrate this, consider a simplified example of one joint, as shown in Figure 3.5a. Since we are using a PD controller only to control the joint position, there is usually an error in tracking the joint position command, as shown by the difference between q_t^{cmd} and q_t . If we set $q_t^{ref} = q_t$, when we interpolate between q_t^{ref} and q_{t+1}^{cmd} , it tends to cause a sudden change in the PD controller’s set point, as shown in Figure 3.5a. A sudden change in the set point can cause a sudden change in the joint torque command and hence cause jerky motion. To resolve this issue, we use the previous joint position command as the reference, as shown in Figure 3.5b. In other words, $\mathbf{q}_t^{ref} = \mathbf{q}_t^{cmd}$, and $\mathbf{q}_{t+1}^{cmd} = \mathbf{q}_t^{cmd} + \mathbf{a}_t$.

3.3.3 Student Policy Learning: Imitate and Stop

After learning a goal-conditional teacher policy $\mathbf{a}_t^\mathcal{E} = \pi^\mathcal{E}(\mathbf{o}_t^\mathcal{E}, \mathbf{a}_{t-1}, g)$, the next question is how to train a real-world deployable student policy that can rotate the object in hand and hold it stably after reorientation. We propose conditioning the student policy on a stop signal $I_t^{stop} \in \{0, 1\}$: $\mathbf{a}_t^\mathcal{S} = \pi^\mathcal{S}(\mathbf{o}_t^\mathcal{S}, \mathbf{a}_{t-1}, I_t^{stop})$. In other words, the student policy should continue

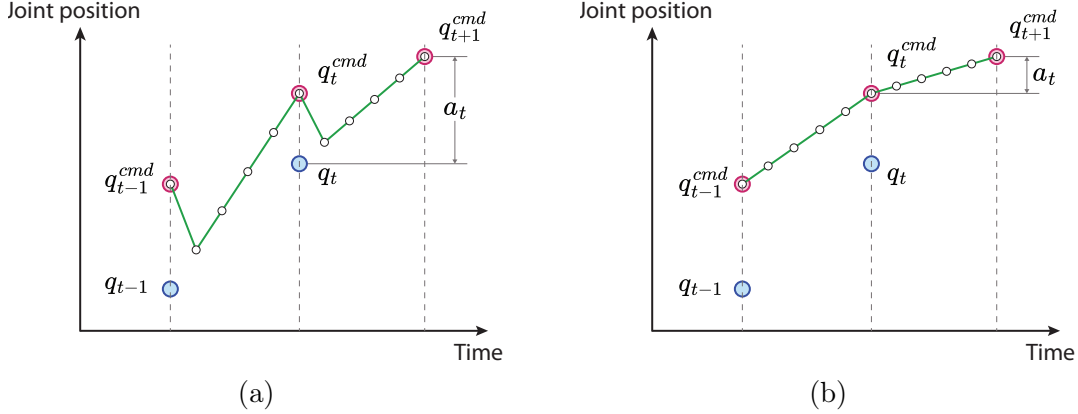


Figure 3.5: Examples of joint position commands after interpolation sent to a low-level PD controller. \bullet represents the actual joint position of the motor. \circ is the computed desired joint position. \circ on the green line shows the interpolated joint position commands that are sent to the low-level PD controller. (a) shows the case of $\mathbf{q}_{t+1}^{cmd} = \mathbf{q}_t + \mathbf{a}_t$, while (b) shows the case of $\mathbf{q}_{t+1}^{cmd} = \mathbf{q}_t^{cmd} + \mathbf{a}_t$. We can see that (b) generates much smoother joint commands.

reorienting the object when $I_t^{stop} = 0$, but stably hold the object when $I_t^{stop} = 1$. This design choice provides flexibility in how we control the policy to stop the reorientation. For example, the policy could rotate the object for a pre-specified amount of time (i.e., set $I_t^{stop} = 1$ after t seconds). Alternatively, an external perception module could detect when the peeled part has fully turned over, triggering $I_t^{stop} = 1$ and the policy to stop the motion and hold the object immediately.

How can we use the learned goal-conditioned teacher policy to train a student policy that is conditioned on the stop signal? We can set the value for I_t^{stop} automatically during policy rollout based on the orientation distance $\Delta\theta_t$.

$$I_t^{stop} = \begin{cases} 0 & \text{if } \Delta\theta_t > \bar{\theta} \\ 1 & \text{otherwise} \end{cases}$$

Observation Space

In this work, we only use proprioceptive sensory information (joint positions \mathbf{q}_t and velocities $\dot{\mathbf{q}}_t$) as the observation input (\mathbf{o}_t^S). Our findings indicate that relying solely on proprioceptive sensory information results in strong performance. Future research could investigate incorporating visual data to further enhance the system’s capabilities, such as preventing objects from slipping out of the grasp.

Policy Architecture

As the student policy only has access to a limited amount of sensory information (a POMDP setting), it is important to incorporate history information, as has been done in previous works [15], [19], [70]. While [15], [19], [70] utilized RNNs to process history information, Transformers [82] have gained significant attention due to their improved performance and

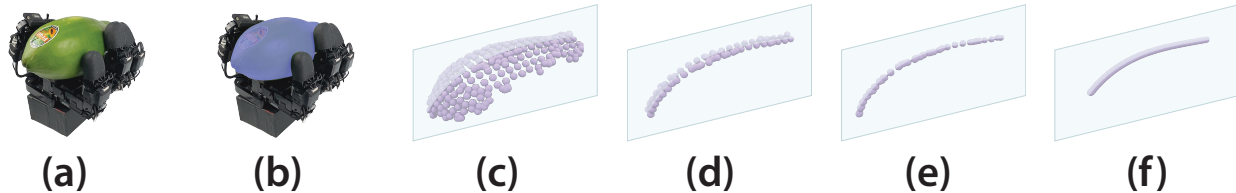


Figure 3.6: **(a)**: the Allegro hand holds a papaya to be peeled. **(b)**: we utilize Grounded SAM to segment the papaya. **(c)**: the 3D point cloud representing the segmented papaya’s exposed surface. **(d)**: we take a slice of this point cloud at the center region along the papaya’s longest axis. **(e)**: the points within this center slice are projected onto the central plane aligned with the axis. **(f)**: we fit a spline curve to the projected points to obtain the desired trajectory for the peeler tip to follow.

faster training in domains such as natural language processing. Therefore, in this work, we employ a Transformer-based policy architecture. $\mathbf{a}_t^S = \pi^S(\mathbf{o}_1^S, \mathbf{a}_0, I_1^{stop}, \dots, \mathbf{o}_t^S, \mathbf{a}_{t-1}, I_t^{stop})$. The policy is a decoder-only attention network (Figure 3.4c) with three self-attention layers. The hidden size is 256, the intermediate size is 512, and the number of attention heads is 8.

Training

The policy is trained using DAGGER [25].

3.3.4 Peeling

In this section, we demonstrate that our reorientation controller can be used for downstream peeling tasks. We use the dexterous robot hand to do the reorientation and then control another Franka Panda robot arm to do the peeling as shown in Figure 3.1. To control the robot arm, we experimented with both using a teleoperation system and an automatic vision-based peeling system.

Teleoperation-based peeling

We used a leader-follower teleoperation system in which a human operator controls a leader system, and the Franka arm follows the motion of the leader in real-time. A 200 Hz operational space impedance controller [83] runs on the Panda arm, controlling for pose via torque, and an operator interacts with a Haption Virtuose™ 6D HF TAO¹ device. Bilateral position-position haptic coupling is done between the two devices. The controllers and haptic coupling are implemented using Drake [84].

Vision-based peeling

While teleoperation provides effective peeling commands for the Franka arm and demonstrates that our reorientation controller can firmly grasp objects after reorientation, automating the peeling process would be ideal. One approach to achieve this is by computing the peeler’s

¹<https://www.haption.com/en/products-en/virtuose-6d-tao-en.html#fa-download-downloads>

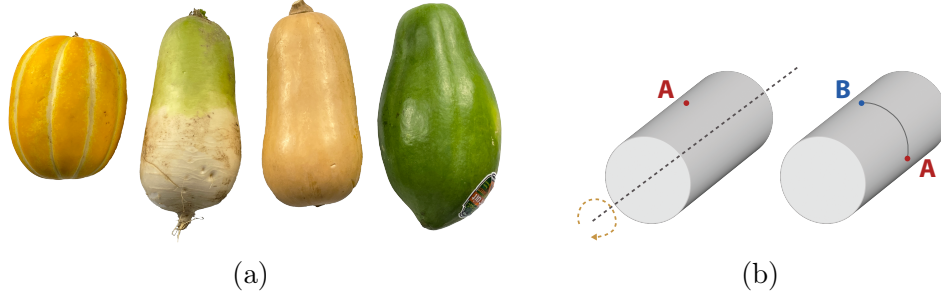


Figure 3.7: **(a)** shows the objects for evaluation: melon, radish, pumpkin, papaya. **(b)** shows the traveling distance. Before reorientation begins, we ensure a reference point (point A) is facing upward. After reorientation, we identify the point (point B) now facing upward. We then measure the distance from point A to point B along the contour.

motion trajectory based on RGB and depth vision data. The trajectory can be determined through the following steps (see Figure 3.6): (1) We utilize Grounded SAM [85] to segment the target vegetable given an image and vegetable name input. (2) Using the segmentation mask and depth data, we reconstruct the 3D point cloud representing the vegetable’s top surface. (3) We identify the vegetable’s longest axis (the peeling direction) by applying principal component analysis. (4) We slice the point cloud into a 2cm thick segment along the central plane that crosses the center point and aligns with the longest axis. We then project all the points within the slice onto the plane. (5) We fit a spline curve to the projected points to obtain a smooth trajectory for the peeler tip. Finally, cartesian-space position control moves the peeler along this trajectory while keeping the peeler orientation fixed.

3.4 Results

To quantitatively evaluate the real-world policy transfer performance, we tested the controller on four vegetables (Figure 3.7a): a pumpkin (mass: 827g), a melon(623g), a radish(727g), a papaya(848g).

3.4.1 Traveling distance for a fixed amount of commanded motion time

The first question we want to answer is whether the learned policy can successfully reorient vegetables in the real world. In peeling, the width of the peeled part depends on the peeler’s width. Thus, it is more informative to measure how much the reorientation controller rotates an object by the traveling distance of a surface point, rather than the absolute rotation angle. Specifically, we mark a reference point P^{ref} on the object surface near the mid-point of its rotational axis. At the start, we ensure P^{ref} is centered and facing upward when held. After reorientation, we record the new point P^{new} that is now centered and facing upward. We then measure the contour length from P^{new} to P^{ref} along the surface (Figure 3.7b).

To demonstrate the capability of our controller to reorient real objects, we conducted two rounds of testing. Our controller is trained to stop motion when it receives a stop signal. In

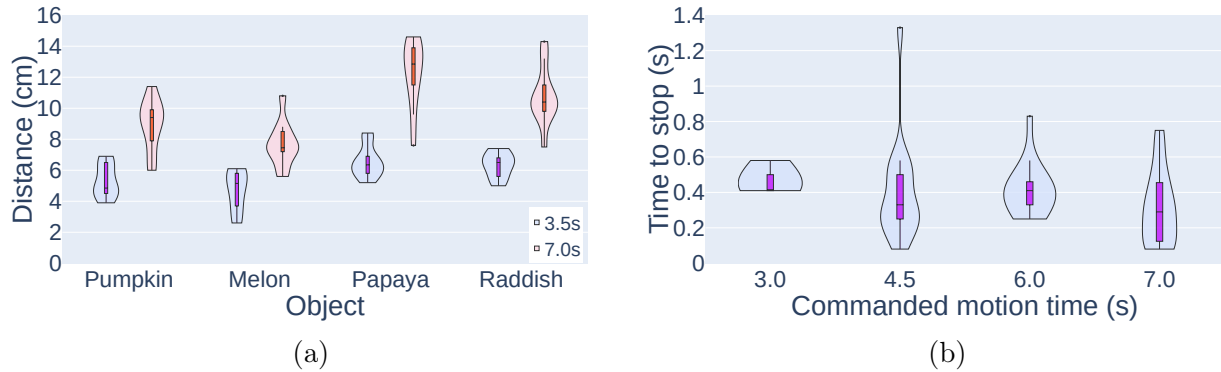


Figure 3.8: **(a)**: Violin plots showing the distribution of the traveling distance of a point on the object surface after the controller is commanded to rotate the object for 3.5 s and 7 s, respectively. **(b)**: Violin plot showing the distribution of time taken by the controller to transition from rotating the object in hand to firmly holding the object after receiving the stop signal. The x -axis represents the timing of the stop signal sent to the controller after it starts.

the first round, we sent the stop signal 3.5 seconds after the controller started rotating. In the second round, we sent the stop signal 7 seconds after start. We repeated each test 10 times. As shown in Figure 3.8a, the controller successfully reoriented all four food items by a sufficient amount for peeling. When commanded to reorient for 3.5s, 90% of tests reoriented the objects by at least 4cm. With 7s, 90% of tests reoriented objects by at least 7.3cm. Given more time, the controller reoriented objects by a larger amount.

3.4.2 How well does the controller track the commanded motion time?

As discussed in Section 3.3, if our controller can quickly respond to a stop signal at any time step, it can be combined with a perception system that tracks peeling progress. Hence, we measured how long it takes to stop the hand and object motion after receiving the stop signal. As shown in Figure 3.8b, the motion stops after 0.4s on average after the controller receives the stop signal.

3.4.3 Firm grasp after reorientation

To enable downstream peeling, the reorientation controller must learn to firmly grasp the object after stopping finger motion. We tested this by checking if the Allegro hand and object could be lifted in the air for 3s by only lifting the object with a single human hand. Table 3.1 shows that across objects and commanded times, the controller firmly grasped objects in 90% of tests. Moreover, our controller possesses the capability of performing consecutive reorientations. It can repetitively execute the sequence of peeling and reorientation multiple times in succession.

Table 3.1: Successful lifting rate (10 tests each)

Commanded motion time	Pumpkin	Melon	Papaya	Radish
3.5s	80%	90%	80%	90%
7s	100%	90%	100%	90%

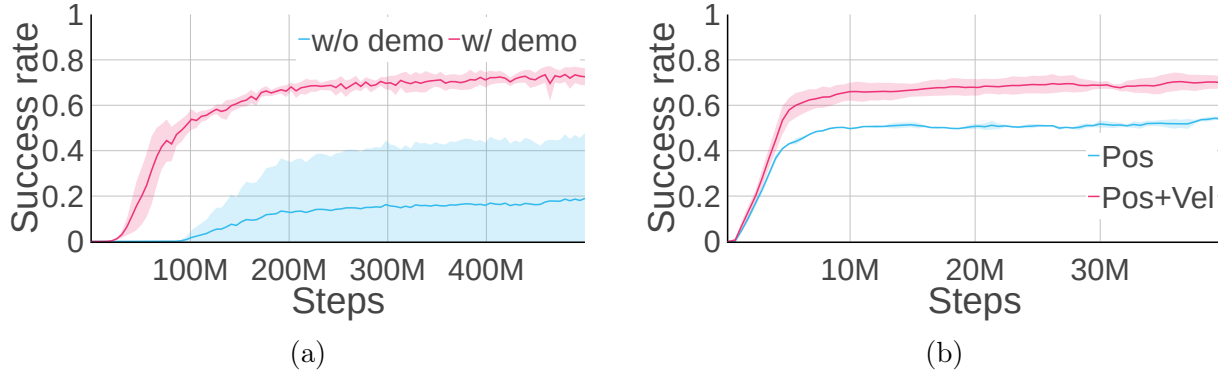


Figure 3.9: **(a)** shows learning curves of the teacher policies with or without $c_3 \|\mathbf{q}_t - \mathbf{q}^{demo}\|_2^2$ in the reward function. **(b)** shows the differences between student policies trained with different sensory information (joint positions and velocities vs. joint positions only).

3.4.4 Real-world Peeling

We evaluated whether the reorientation controller could reorient food items to facilitate peeling (Figure 3.1). We tested using an Allegro hand and a Leap hand [86]. Testing showed that peeling applied substantial pulling forces on objects. However, in most cases, both hands maintained a firm enough grasp to enable successful peeling. Failures often occur when holding small objects, as some fingertips may fail to establish secure contact with the surface.

3.4.5 Ablation study

Demo term in Reward function

We proposed using a keyframe demonstration to ease reward shaping. To evaluate its effectiveness, we compared learning curves of the teacher policies trained with and without the $c_3 \|\mathbf{q}_t - \mathbf{q}^{demo}\|_2^2$ reward term. As shown in Figure 3.9a, adding the keyframe substantially improved learning. Additionally, it demonstrates that mimicking the keyframe pose via a single reward term effectively reduces the reward-shaping burden.

Necessity of having joint velocity information in π^S

The student policy’s sensory input included joint positions and velocities. We investigated whether including joint velocity information in the input is beneficial. Figure 3.9b shows that adding joint velocities to the input improved performance.

Transformer vs RNN

Different from prior works [15], [18], [19], [70], our student policy uses a Transformer architecture instead of an RNN architecture. We compared the learning performance of a Transformer-based policy and an RNN-based policy. Figure 3.10 shows that a Transformer-based policy learns much faster and gets better performance at convergence than an RNN-based policy.

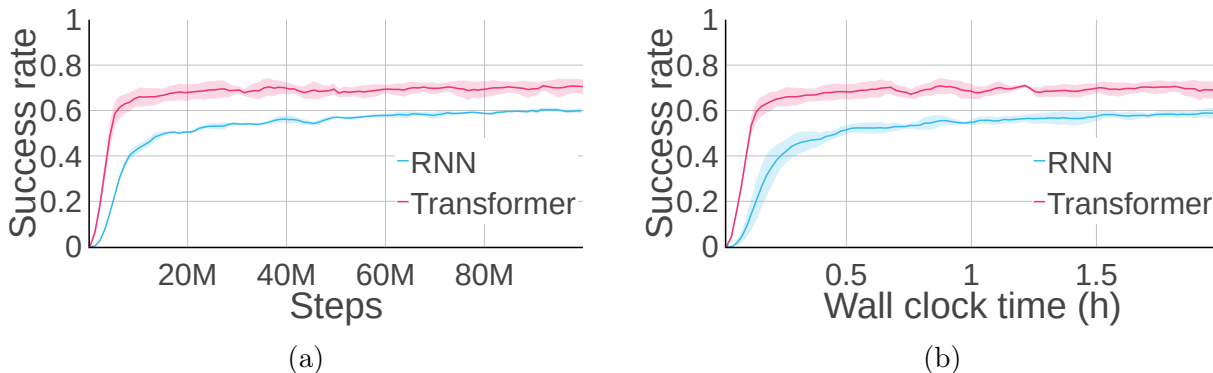


Figure 3.10: Learning curves of student policies with a Transformer or RNN architecture with respect to the number of samples and wall-clock time, respectively.

3.5 Discussions

The reorientation controller presented in this study is a blind controller that relies solely on proprioceptive sensory information. While it has demonstrated the ability to successfully reorient heavy objects and securely hold them in place, its performance could potentially be enhanced by incorporating visual and tactile feedback.

The current system has a few failure modes. Firstly, the object might slip out of the hand since the controller does not utilize any vision information. Secondly, the controller might fail if the vegetables are small, as the fingers cannot effectively make contact with the object. When using a vision-based peeling approach to peel the vegetables, the segmentation network (Grounded SAM) might fail to correctly identify and segment the target vegetable in the image. Sometimes, the segmentation mask would incorrectly include the robot hand. Some fine-tuning of the pre-trained Grounded SAM model would be necessary to mitigate such issues.

Future work could involve learning a peeling policy via behavior cloning on data collected via teleoperation to achieve better autonomy of the system. Additionally, incorporating visual and tactile feedback into the reorientation controller could potentially enhance its performance

3.6 Acknowledgements

I want to thank Eric Cousineau, Naveen Kuppaswamy, and Pulkit Agrawal for their help with this project. Eric was very helpful in debugging the Allegro hand control system, setting up the teleoperation system, and discussing and brainstorming ideas for the project. Naveen and Pulkit also provided great support and contributions that were important for completing the project successfully.

Chapter 4

Parallel Q -Learning

4.1 Overview

Reinforcement learning (RL) has achieved impressive results on many real-world problems. A primary challenge in using RL is the need for large amounts of real-world data. There are two main strategies to tackle this problem. One is to improve the sample efficiency of RL algorithms [38], [87] to make better use of available data. The other is to reduce the need for real-world data collection by training policies in simulation and deploying them in the real world [15], [19], [31], [33], [88]. In sim-to-real pipelines, the training wall-clock time matters more than the sample efficiency — faster training can speed up the experiment cycle and unlock the potential for addressing a broader range of complex problems.

The community has widely recognized the need for faster training, leading to the development of several distributed frameworks [89], [90]. However, these frameworks usually operate at a *server scale* that requires hundreds or thousands of computers in a cluster, making them impractical for many researchers and practitioners. In these frameworks, most computers run multiple simulator instances in parallel to speed up data collection. Recent advances in GPU-based simulation, such as Isaac Gym [30], mitigated the need for a large cluster by enabling the parallel simulation of *tens of thousands* of environments on a single GPU. A natural question is: what RL algorithm achieves the best wall-clock time when training uses massively parallel simulation on GPUs? Many prior works [34], [91], [92] use on-policy algorithms like PPO [37] for training in Isaac Gym due to its simplicity and easy-to-scale nature.

It is known that on-policy methods have lower data efficiency than off-policy methods. Intuitively, by virtue of requiring less data than on-policy algorithms, off-policy algorithms (Q -learning methods, in particular) should reduce the wall-clock time of training. However, better sample efficiency will not lead to shorter training time if the algorithm cannot efficiently use parallel environments. Some prior works [89], [93] have developed distributed frameworks for off-policy methods to leverage parallel environments. However, these frameworks have only shown successful scaling with hundreds of parallel environments (for example, a maximum of 256 environments in [89]). Now that GPU-based simulation enables *tens of thousands* of parallel environments on a single GPU, it remains unclear whether off-policy methods can work efficiently in this case. For instance, if there are 10,000 parallel environments and

we still use the typical replay buffer capacity (say 1M samples), the entire replay buffer is refreshed every 100 environment steps, making the data in the replay buffer more like the data collected from an on-policy method. Do off-policy methods still retain their data efficiency in this scenario? Increasing the replay buffer capacity is not always an option due to the memory size limits of the hardware.

In this work, we investigate how to scale up Q -learning to tens of thousands of environments. We present our approach, **Parallel Q-Learning (PQL)**, which can be deployed on a workstation. The learning speed in PQL is boosted by parallelizing the data collection, policy function learning, and value function learning on a single workstation. This allows for collecting more simulation data and updating value/policy functions more times in a given time window, leading to an improvement in the training wall clock time. Achieving such parallelization would be non-trivial for on-policy algorithms, as the policy update requires on-policy interaction data, which means that data collection and policy updates need to happen in sequence.

Our main contributions are summarized as follows:

- We present a scheme for time-efficient reinforcement learning, **PQL**, that can efficiently leverage tens of thousands of parallel environments on a workstation.
- We thoroughly investigate the effect of important hyperparameters such as the speed ratio on different processes that control the resource allocation and provide empirical guidelines for tuning these values to scale up Q -learning.
- We deploy different exploration strategies in parallel environments, which leads to robust exploration and mitigates the hassle of tuning the exploration noise.
- We demonstrate the effectiveness of our method on six Isaac Gym benchmark tasks [30] and show its superiority over state-of-the-art (SOTA) on-/off-policy algorithms. Our method **PQL** achieves both faster learning in wall-clock time and better sample efficiency. Empirically, we also found that DDPG performs better than SAC in a massively parallel environment setting.

4.2 Related Work

Massively Parallel Simulation Simulation has been an important tool in various research fields, such as robotics, drug discovery, and physics. In the past, researchers have used simulators like Drake [84], MuJoCo [94] and PyBullet [95] for rigid body simulation. Recently, there has been a new wave of development in GPU-based simulation, e.g., Isaac Gym [30]. GPU-based simulation has substantially improved simulation speed by allowing massive amounts of parallel simulation on a single commodity GPU. It has been used in various challenging robotics control problems, including quadruped locomotion [33], [92] and dexterous manipulation [19], [34], [91]. With fast simulation, one can obtain much more environment interaction data in the same training time as before. This poses a challenge to RL algorithms in making the best use of the massive amount of data. A straightforward way is to use on-policy algorithms such as PPO, which can be easily scaled up and is also the default

algorithm used by researchers in Isaac Gym. However, on-policy algorithms are less data-efficient. In our work, we investigate how to scale up off-policy algorithms to achieve higher sample efficiency and shorter wall-clock training time under massively parallel simulation.

Distributed Reinforcement Learning There have been numerous distributed reinforcement learning frameworks to speed up learning. One line of work focuses on Q -learning methods. Gorila [93] distributes DQN agents to many machines where each machine has its local environment, replay buffer, and value learning, and uses asynchronous SGD for a centralized Q function learning. Similarly, [96] apply asynchronous SGD to the DDPG algorithm [38]. Combining with prioritized replay [97], n -step returns [98], and double- Q learning [99], [89] (Ape-X) parallelize the actor thread (environment interactions) for data collection and use a centralized learner thread for policy and value function learning. Built upon Ape-X, [100] adapt the distributed prioritized experience replay for RNN-based DQN agents.

Another line of work improves the training speed on policy gradient methods. A3C [101] uses asynchronous SGD across many CPU cores, with each running an actor learner on a single machine. [102] develop a hybrid CPU/GPU implementation of A3C, but it can have poor convergence due to the stale off-policy data being used for the on-policy update. [90] (IMPALA) introduce an off-policy correction scheme (V-trace) to mitigate the lagging issue between the actors and learners in distributed on-policy settings. [103] further improve the IMPALA training speed by moving the policy inference from the actor to the learner. [104] parallelize the environments for synchronous advantage actor-critic. [105] propose a distributed version of PPO [37] for training various locomotion skills in a diverse set of environments. [106] develop a decentralized version of distributed PPO to mitigate the synchronization overhead between different actor processes and applies it to a point-goal navigation task.

Our scheme is most closely related to Ape-X [89] but has a number of key differences. **First**, our scheme is specifically designed for massively ($\gg 1000$) parallel GPU-based simulation. Our scheme is optimized for a single-machine setup, which can help democratize large-scale RL research. **Second**, we further decouple and parallelize the learning with two separate learners for policy function learning and Q -function learning, respectively. **Third**, we allocate a local replay buffer for each learner. This can reduce the communication cost between the replay buffer and the learners. **Fourth**, working with a single machine presents new challenges in balancing the computing resource between different parallel processes. Our scheme offers a mechanism to balance the computing resource among different processes.

4.3 Method

We developed a parallel off-policy training scheme, **Parallel Q-Learning (PQL)**, for massively parallel GPU-based simulation, where thousands of environments can be simulated simultaneously on a single GPU. In a typical actor-critic Q -learning method, three components run sequentially: a policy function, a Q -value function, and an environment. Agents roll out the policy in the environments and collect interaction data, which is added to a replay buffer; then, the value function is updated to minimize the Bellman error, after which the

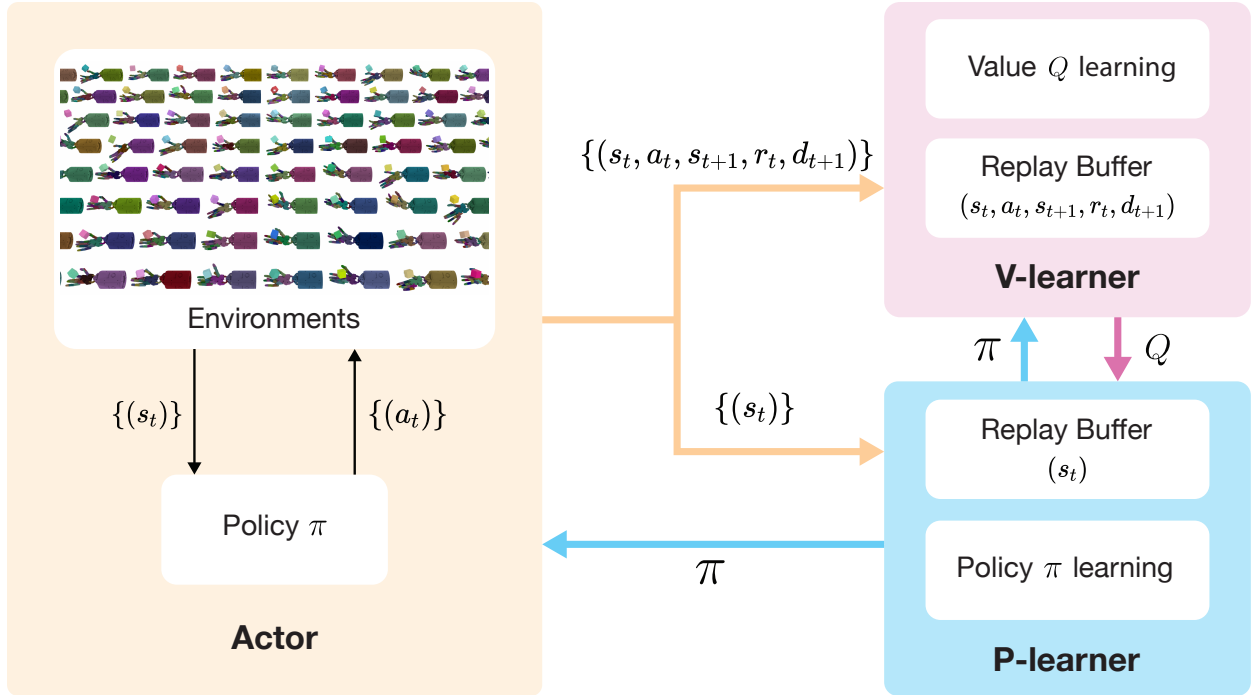


Figure 4.1: Overview of Parallel Q Learning (PQL). We have three concurrent processes running: **Actor**, **P-learner**, **V-learner**. **Actor** collects interaction data. **P-learner** updates the policy network. **V-learner** updates the Q functions.

policy function is updated to maximize the Q values. This sequential scheme slows down the training, as each component needs to wait for the other two to finish before proceeding. To maximize the learning speed and reduce the waiting time, we parallelize the computation of all three components. This allows for more network updates per data point, which can improve the utilization of the massive amount of data and lead to better training speed, as demonstrated in the experiments. Off-policy RL methods are well-suited for parallelization as the interaction data in a replay buffer does not need to come from the latest policy. In contrast, on-policy methods such as PPO require using the rollout data from the latest policy (on-policy data) to update the policy, thus making it non-trivial to parallelize the data collection and policy/value function update.

Our scheme is optimized for training speed in terms of wall-clock time and can be readily applied on a workstation. It is built upon DDPG [38], but can be easily extended to other off-policy algorithms such as SAC [107] (see Appendix B.3). Our scheme also incorporates common techniques used to improve Q learning performance, such as double Q learning [99] and n -step returns [98]. Furthermore, we experimented with adding distributional RL [108] to PQL, which we refer to as **PQL-D**. While it improves performance on challenging manipulation tasks, it leads to a slight decrease in the convergence speed of the RL agent. In this paper, we use the following notation: at time step t , s_t represents observation data, a_t represents action command, r_t represents the reward, d_t represents whether the environment terminates, $\pi(s_t)$ represents the policy network, $Q(s_t, a_t)$ represents the Q network, $Q'(s_t, a_t)$ represents the target Q network, and N represents the number of parallel environments.

4.3.1 Scheme Overview

PQL parallelizes data collection, policy learning, and value learning into three processes, as shown in Figure 4.1. We refer to them as **Actor**, **P-learner**, and **V-learner**, respectively.

- **Actor**: We collect a batch of interaction data using parallel environments. We use Isaac Gym [30] as our simulation engine, which supports massively parallel simulation. Note that we do not make any Isaac-Gym-specific assumptions, and PQL is optimized for any GPU-based simulator that supports a large number of parallel environments. In the **Actor** process, the agent interacts with tens of thousands of environments according to an exploration policy. Therefore, we maintain a local policy network $\pi^a(s_t)$, which is periodically synchronized with the policy network $\pi^p(s_t)$ in **P-learner** (which we explain below).
- **V-learner**: We create a dedicated process for training value functions, which allows for continuous updates without being interrupted by data collection or policy network updates. To compute the Bellman error, we need the policy network to estimate the optimal action and the replay buffer to sample a batch of I.I.D. training data. Since we use a dedicated process for updating value functions, **V-learner** must frequently query the policy network and sample data from the replay buffer. To reduce the communication overhead of the policy and data across processes, we maintain a local version of the policy network $\pi^v(s_t)$ and the replay buffer in **V-learner**. $\pi^v(s_t)$ is synced with $\pi^p(s_t)$ in **P-learner** periodically. When the GPU memory is sufficiently large to host the entire replay buffer, which is usually the case when observations are not images, we construct the replay buffer on the GPU to avoid the CPU-GPU data transfer bottleneck.
- **P-learner**: We use another dedicated process for updating the policy network $\pi^p(s_t)$, which is optimized to maximize the $Q^p(s_t, \pi^p(s_t))$. We also maintain a local replay buffer of $\{(s_t)\}$ and a value function $Q^p(s_t, a_t)$ in **P-learner** to reduce communication overhead across processes. $Q^p(s_t, a_t)$ is periodically updated with $Q^v(s_t, a_t)$ in **V-learner**.

We use Ray [109] for parallelization. The pseudo-code is in Algorithm 1, 2, and 3 in the Appendix B.1.

Data Transfer Suppose there are N parallel environments in the **Actor** process. At each step, the **Actor** rolls out the policy $\pi_a(s_t)$ and generates N pairs of $(s_t, a_t, s_{t+1}, r_t, d_{t+1})$. Then the **Actor** sends the entire batch of interaction data $\{(s_t, a_t, s_{t+1}, r_t, d_{t+1})\}$ to the **V-learner** (see Figure 4.1). Since policy update in **P-learner** only needs state information, **Actor** only sends $\{(s_t)\}$ to the **P-learner**.

Network Transfer The **V-learner** periodically sends the parameters of the $Q^v(s_t, a_t)$ to **P-learner**, which updates the local $Q^p(s_t, a_t)$ in **P-learner**. The **P-learner** sends the policy network $\pi^p(s_t)$ to both the **Actor** and **V-learner**.

Both the data and policy network transfer happen concurrently.

4.3.2 Balance between Actor, P-learner, and V-learner

Our scheme allows the **Actor**, **P-learner**, and **V-learner** to run concurrently. However, if each process ran as fast as possible, independent of each other, it can make training unstable. Thus, we explicitly control the update frequencies of the three processes using the following two ratios:

$$\beta_{a:v} := \frac{f_a}{f_v} \quad \text{and} \quad \beta_{p:v} := \frac{f_p}{f_v},$$

where f_a is the number of rollout steps per environment in **Actor** per unit time, f_v is the number of Q function updates in **V-learner** per unit time, f_p is the number of policy updates in **P-learner** per unit time. $\beta_{a:v}$ determines how many steps **Actor** rolls out the policy with N environments when one Q function update is performed in the **V-learner**. $\beta_{p:v}$ decides how many Q function updates are performed in **V-learner** when **P-learner** updates the policy once. Once the ratios are set, we monitor the progress of each process and dynamically adjust the speed of **Actor** and **P-learner** by letting the process wait if necessary.

Controlling the three processes via $\beta_{a:v}, \beta_{p:v}$ provides three major advantages. **First**, it allows us to balance the resource allocation of each process and reduce the variance of PQL’s performance. Given a fixed amount of computing resources, the ability to let some of the processes wait enables other processes to use the GPU resource more. This is particularly important when working with limited resources. If there is only one GPU, and all three processes run freely on it, simulation with a large number of environments can cause very high GPU utilization, which slows down the **P-learner** and **V-learner** and leads to worse performance. Note that such control was not examined in prior studies, such as Ape-X [89], where a computer cluster was used for both the simulation and network training — the phenomenon of competing for limited computing resources (all three processes on one GPU) did not occur. On the other hand, leaving each process running freely creates more variance in the training speed and learning performance as the simulation speed and network training speed are heavily dependent on the task complexity, network size, computer hardware, etc. For example, simulation for contact-rich tasks can be slower than others; some tasks might require a deeper policy network or Q networks; even the GPUs on a machine might have different running conditions at different times, leading to different speeds across processes and further leading to different learning performances.

Second, ratio control can improve convergence speed. For example, prior works [110] show that updating the policy network less frequently than the Q functions leads to better learning. **Third**, the ratio $\beta_{p:v}$ can be interpreted as the frequency of the target policy network update. One may notice that we use a lagged policy to update the Q function and synchronize it according to the above ratio. Therefore, we do not create a target policy network explicitly, but every synchronization can be considered as a hard update of the policy network.

4.3.3 Mixed Exploration

We can achieve improvement in convergence by having a good exploration strategy. Too much exploration can make agents fail to latch onto useful experience and learn a good

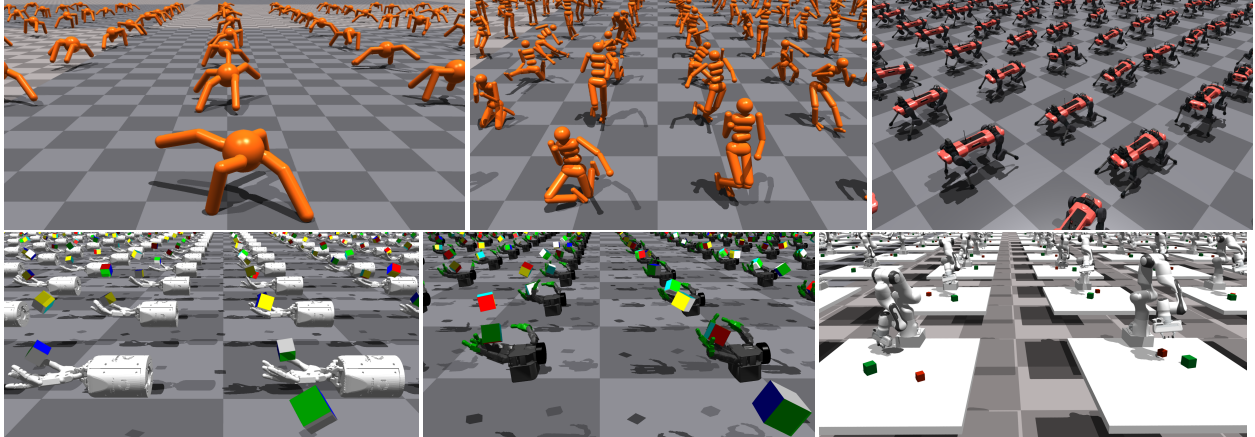


Figure 4.2: We experiment on six Isaac Gym tasks: *Ant*, *Humanoid*, *ANYmal*, *Shadow Hand*, *Allegro Hand*, *Franka Cube Stacking*.

policy quickly, while too little exploration does not give the agent enough good interaction data to improve the policy. Balancing the exploration and exploitation often requires extensive hyper-parameter tuning or complex scheduling mechanisms. In DDPG, one common practice to control exploration is to set the standard deviation σ of the uncorrelated and zero-mean Gaussian noise that is being added to the deterministic policy output ($a_t = \max(\min(\pi(s_t) + \mathcal{N}(0, \sigma), a_u), a_l)$ [111]–[113] where $a_t \in [a_l, a_u]$). Since it is difficult to predict how much exploration noise is appropriate, one typically needs to tune σ for each task. Can we mitigate the hassle of tuning σ ? Our idea is that instead of finding the best σ value, we can try out different σ values altogether, which we call **mixed exploration**. Even if some σ values lead to bad exploration at a certain training stage, others can still generate good exploration data. This strategy is easily implemented thanks to the massively parallel simulation, as we can use different σ values in different parallel environments. Similar ideas have been used in prior works [89], [101]. In our work, we uniformly generate the noise levels in the range of $[\sigma_{\min}, \sigma_{\max}]$. For the i^{th} environment out of N environments, $\sigma_i = \sigma_{\min} + \frac{i-1}{N-1}(\sigma_{\max} - \sigma_{\min})$ where $i \in \{1, 2, \dots, N\}$. We use $\sigma_{\min} = 0.05$, $\sigma_{\max} = 0.8$ for all the tasks in our experiments.

4.4 Experiments

In this section, we demonstrate the effectiveness of our method compared to SOTA baselines, analyze the effects of key hyper-parameters, and provide empirical guidelines for setting their values. All experiments are carried out on a single workstation with a few GPUs. We run each experiment with five random seeds and plot their mean and standard error.

4.4.1 Setup

Tasks We evaluate our method on six Isaac Gym benchmark tasks [30]: *Ant*, *Humanoid*, *ANYmal*, *Shadow Hand*, *Allegro Hand*, and *Franka Cube Stacking* (see Figure 4.2). For more details about these tasks, please refer to [30]. Additionally, we provide two more tasks in Section 4.4.5: (1) a vision-based *Ball Balancing* task and (2) a contact-rich dexterous

manipulation task that requires learning to reorient hundreds of different objects using a *DClaw Hand* with a single policy [19]. Note that we use the four-finger hand version and do not include any domain randomization.

Baselines We consider the following baselines: (1) **PPO** [37], which is the default algorithm used by many prior works [30], [34], [91] that use Isaac Gym for simulation, (2) **DDPG(n)**: DDPG [38] implementation with double Q learning and n -step returns, (3) **SAC(n)**: SAC [107] implementation with n -step returns. Hyper-parameters are available in Appendix B.2.1.

Hardware We use NVIDIA GeForce RTX 3090 GPUs as our default GPUs for the experiments unless otherwise specified. More details are shown in Table B.3 in the appendix.

4.4.2 PQL learns faster than baselines

The first and most important question to answer is whether PQL leads to faster learning than SOTA baselines. To answer this, we compared the learning curves of PQL and PQL-D (PQL with distributional RL) with baselines on six benchmark tasks. As shown in Figure 4.3, our method (PQL, PQL-D) achieves the fastest policy learning in five out of six tasks compared to all baselines. Moreover, we observed that adding distributional RL to PQL can further boost learning speed. Figure 4.3 shows that in five out of six tasks, PQL-D achieves wall-clock time faster than, or at least on par with, PQL. The improvements are most salient on the two challenging contact-rich manipulation tasks (*Shadow Hand* and *Allegro Hand*). Additionally, the faster learning of PQL than DDPG(n) demonstrates the advantage of using a parallel scheme for data collection and network updates. We also found that DDPG(n) outperforms SAC(n) in all tasks. This could be due to the fact that the exploration scheme in DDPG can scale up better than the one in SAC. In DDPG, we apply the same mixed exploration as in PQL, while the exploration of SAC solely comes from sampling in the stochastic policy distribution, which can be heavily affected by the quality of the policy distribution.

4.4.3 How well does mixed exploration perform?

As discussed in Section 4.3.3, massively parallel simulation enables us to deploy different exploration strategies in different environments to generate more diverse exploration trajectories. We use a simple mixed exploration strategy, as described in Section 4.3.3, and compare its effectiveness to cases where all the environments use the same exploration capacity (the same σ values). We experimented with $\sigma \in \{0.2, 0.4, 0.6, 0.8\}$. As shown in Figure 4.4, the learning performance is significantly affected by the choice of σ value. If we use the same σ value for all parallel environments, then we need to tune σ for each task. In contrast, the mixed exploration strategy, where each environment uses a different σ value, outperforms (learns faster or at least as fast as) all other fixed σ values. This implies that using the mixed exploration strategy can reduce the tuning effort needed for σ values per task.

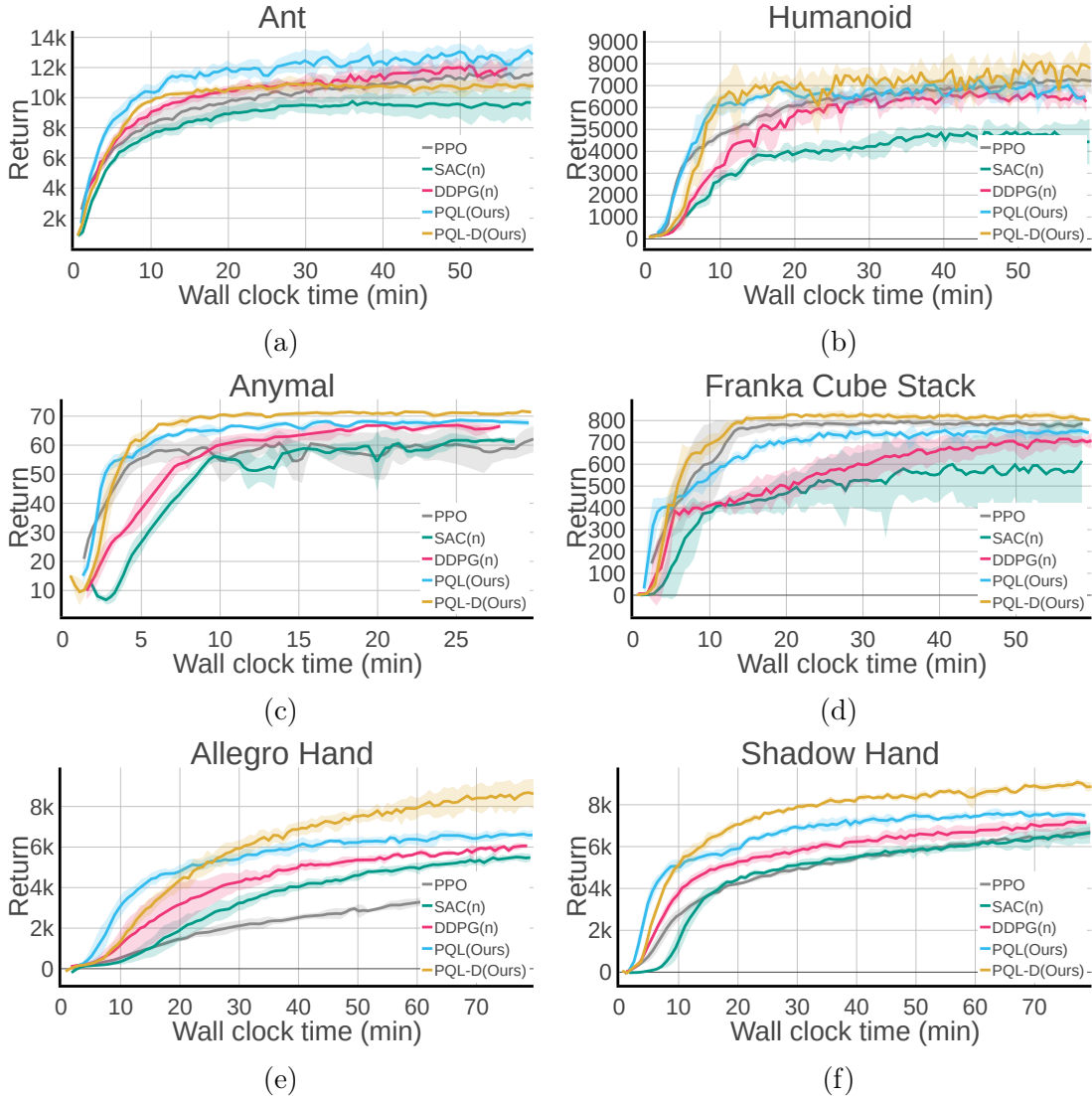


Figure 4.3: We compare our methods to the SOTA RL algorithms (PPO, SAC with n -step returns, DDPG with n -step returns). We use 4096 environments for training in all tasks except the PPO baseline on *Shadow Hand* and *Allegro Hand* tasks, where we use 16384 as it gives the best performance for PPO on these two tasks as shown in Figure 4.5c. Our methods achieve the fastest learning speed in almost all tasks.

4.4.4 Effects of different hyper-parameters

In this section, we investigate the effects of the number of environments, $\beta_{p:v}$, $\beta_{a:v}$, batch size, replay buffer size, and the number of GPUs. These hyper-parameters are of particular interest given the massively parallel simulation ($N \gg 1000$) and our parallel scheme. Lastly, GPU hardware can also impact learning speed. To explore this, we conduct experiments using four different GPU models and analyze the effect of GPU hardware on performance in Appendix B.3. Overall, PQL works robustly across different GPU models.

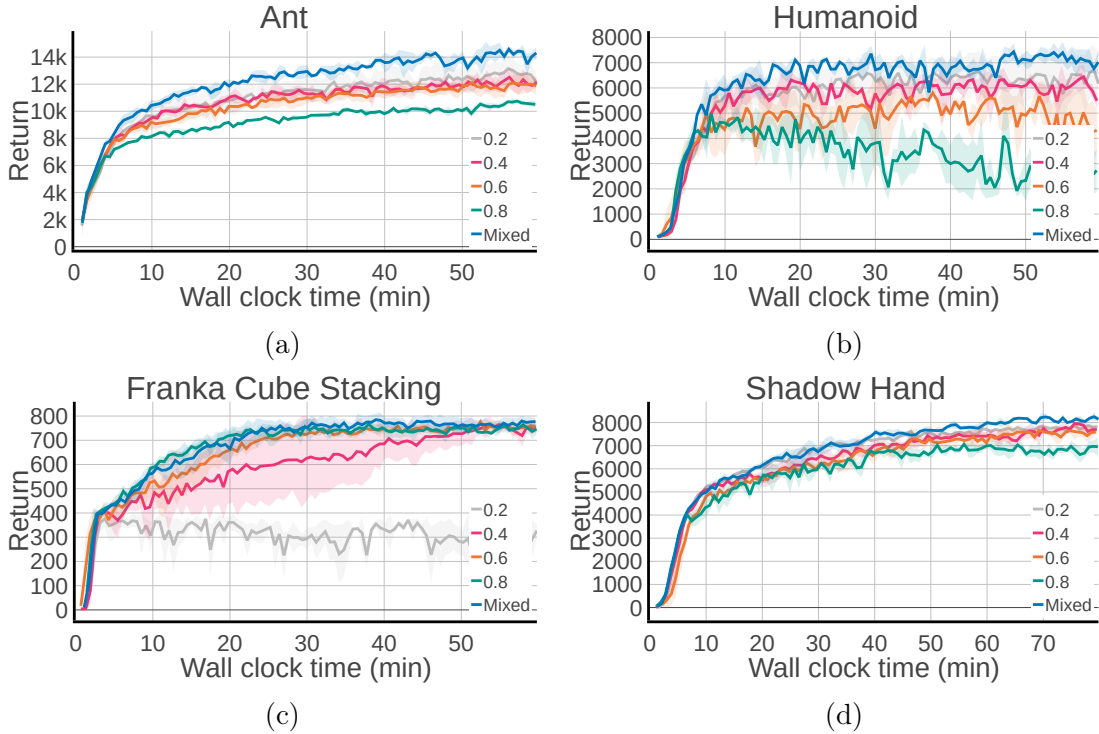


Figure 4.4: We compared our proposed mixed exploration scheme by applying different constant maximum noise values. We can see that the mixed exploration scheme either outperforms or is on par with other schemes, which can save the tuning effort on the noise level.

How does the number of environments N affect policy learning?

Previous works on distributed frameworks for RL [89], [90] have shown how the learning performance is affected by the number of parallel environments N , with N in the order of hundreds. GPU simulation enables running thousands of environments in parallel on a single workstation, and we anticipate that this will only improve with time. However, more parallel environments will only be beneficial if RL algorithms can exploit such data, i.e. if performance scales with more data. We, therefore, investigated how different algorithms scale with the number of environments ($N \gg 1000$, the biggest N we experimented with is 16,384). As shown in Figure 4.5, both PQL and PPO benefit from using more environments in parallel. Moreover, the learning performance of PQL is relatively less sensitive to N on the simple task (*Ant*), while on the hard task (*Shadow Hand*), PPO’s learning performance substantially drops as we decrease the number of environments. In contrast, our method (PQL) demonstrates stable and similar learning with all the different numbers of environments except when N is very small ($N = 256$) on *Shadow Hand*, suggesting that PQL is more robust to changes in N .

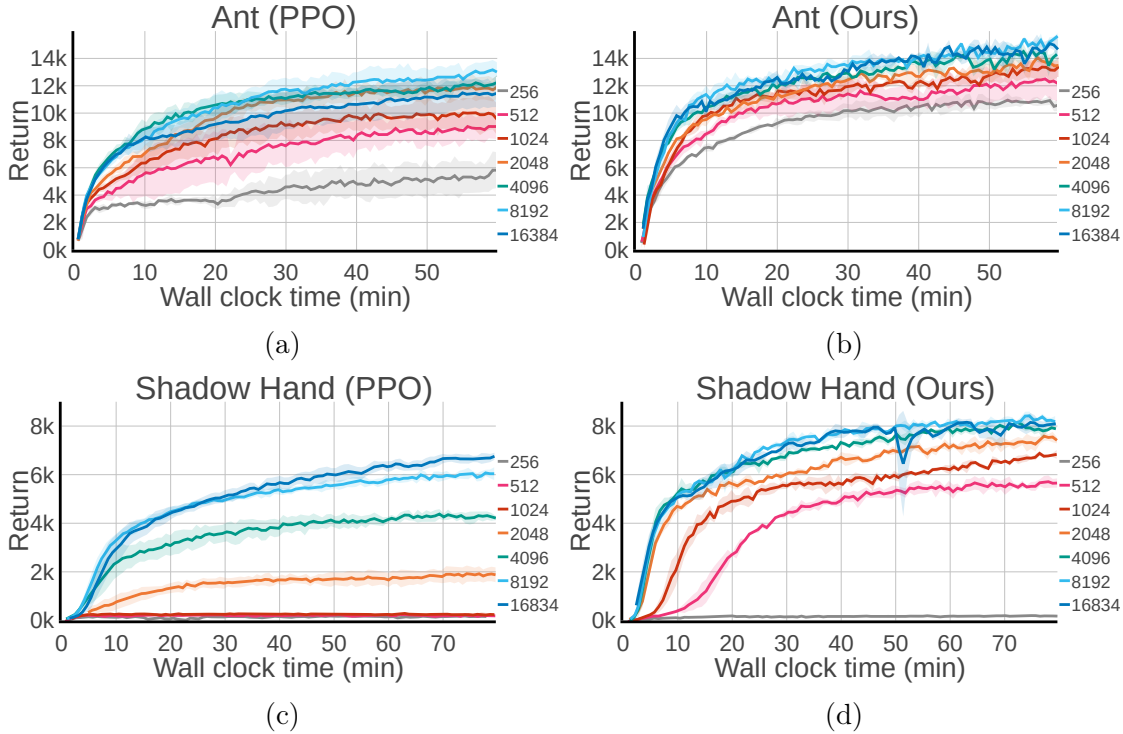


Figure 4.5: We sweep over different numbers of environments (N) on both PPO and PQL (our method). Overall, PQL is less sensitive to the number of environments than PPO on both tasks.

Effect of $\beta_{p:v}$ and $\beta_{a:v}$

As discussed in Section 4.3.2, explicitly controlling the $\beta_{a:v}$ and $\beta_{p:v}$ can help improve the learning performance and reduce the variance under different training conditions, such as fluctuated hardware utilization. If $\beta_{p:v}$ is larger, the policy updates more frequently than the value functions, potentially leading to policy overfitting to the stale value function, which in turn leads to poor exploration. If the policy updates much slower than the value function, the policy might lag behind the value function a lot, which hurts the learning speed. Similarly, if $\beta_{a:v}$ is larger, the **V-learner** might need to wait for **Actor** to collect enough data, since the simulation speed cannot be changed, leading to slower learning. If $\beta_{a:v}$ is smaller, the value function updates more given the generated rollout data.

To qualitatively assess the effects of different $\beta_{a:v}$ and $\beta_{p:v}$, we sweep over a range of values for these two hyper-parameters and compare them in Figure 4.6 and Figure 4.7. Figure 4.6 shows that PQL is relatively robust to a wide range of $\beta_{p:v}$ values, which means this hyper-parameter would require little tuning. We use $\beta_{p:v} = 1 : 2$ as the default value in our experiments shown in the paper. This ratio value is consistent with prior works [110], [113], but our findings are in the context of parallel training of policy and value function. Figure 4.7 shows that $\beta_{a:v}$ has a greater impact on the learning performance. An overall trend is that if we increase the number of environments, then we need to have **V-learner** update the Q functions more times. For example, on *Shadow Hand*, $\beta_{a:v} = 1 : 4$ performs the best when $N = 2048$ and $N = 4096$. But when $N = 8192$ and $N = 16384$, $\beta_{a:v} = 1 : 12$ performs the

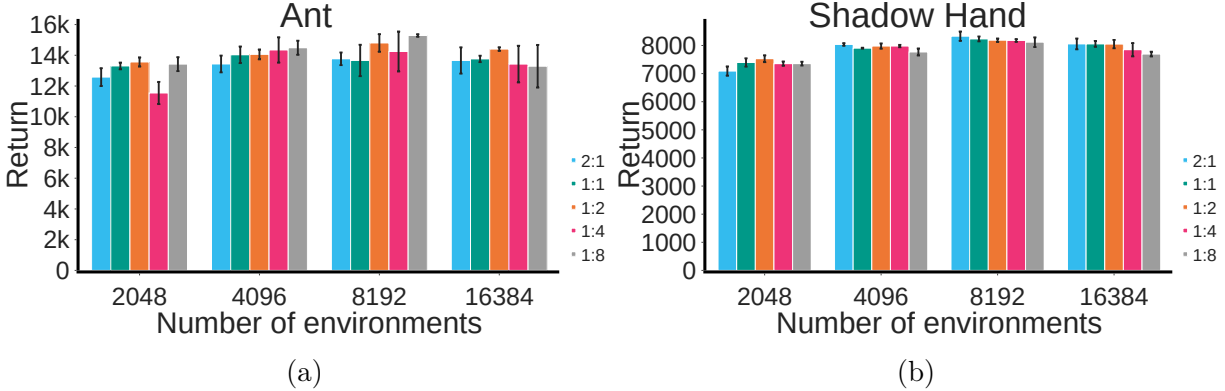


Figure 4.6: We show the averaged returns in evaluation after a fixed amount of training time ΔT . Across the set of different numbers of environments we experimented with (2048, 4096, 8192, 16384), we found that setting $\beta_{p:v} = 1 : 2$ generally works well. $\Delta T = 60$ mins for *Ant*, and $\Delta T = 80$ mins for *Shadow Hand*. The complete learning curves are in Figure B.6.

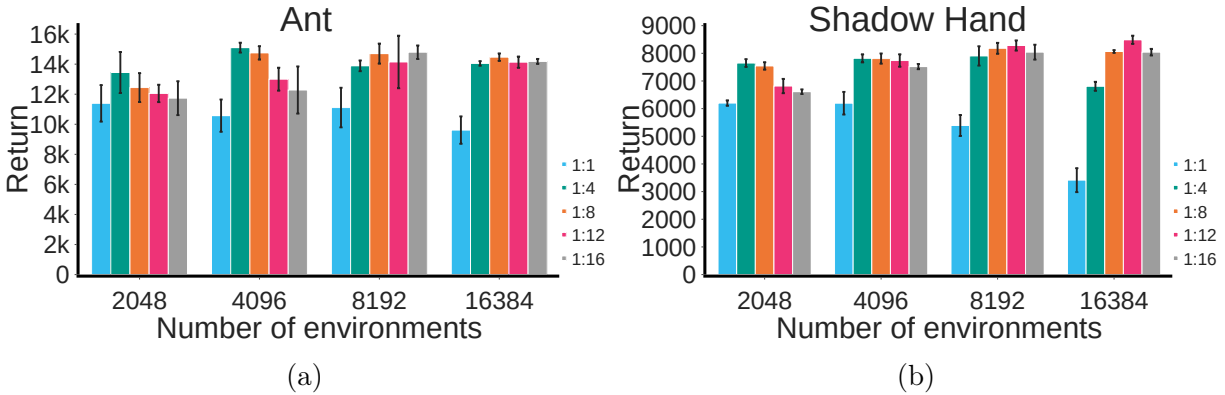


Figure 4.7: Given different values of N , we show the effect of different $\beta_{a:v}$. An overall trend we observe is that as N gets bigger, it’s more beneficial to update the critic more frequently. We also found that $\beta_{a:v} = 1 : 8$ generally works well given different N values. So one can set $\beta_{a:v} = 1 : 8$ as a good initial value, and tune it if necessary on new tasks.

best. We use $\beta_{a:v} = 1 : 8$ by default as it achieves a good performance across different N values. In summary, Figure 4.6 and Figure 4.7 show that $\beta_{p:v}$ and $\beta_{a:v}$ do affect the performance with a varied number of environments. We suggest setting $\beta_{p:v} = 1 : 2, \beta_{a:v} = 1 : 8$ as a good starting point for new tasks and tune them if necessary, as these are the values we found work well on six different tasks with different numbers of environments. In addition, in Section B.3, we show that adding the speed ratio control ($\beta_{a:v}$ and $\beta_{p:v}$) is beneficial for balancing the computing resources used by each process when resources are limited.

Effect of batch size

With many parallel environments (N), a significant amount of data is generated quickly. While it is easy to increase N from hundreds to tens of thousands in Isaac Gym on a single

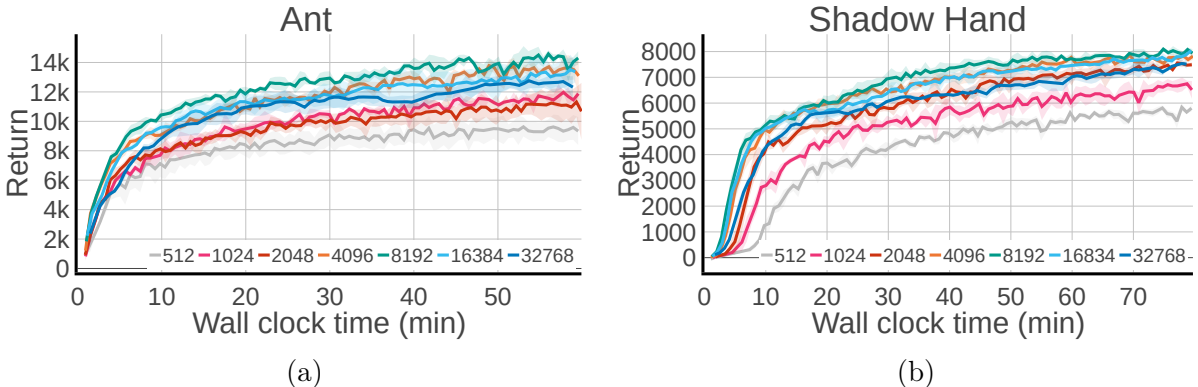


Figure 4.8: Effect of different batch sizes. Small batch size usually leads to slower learning. If the batch size is too big, the policy learning can slow down because GPUs have limited cores and it takes more time to process a very big batch of data.

GPU, it is infeasible to increase the replay buffer size by 100 times due to limited GPU memory or CPU RAM (if the data is stored on the CPU). Consequently, the replay buffer is frequently overwritten, meaning that each collected sample may not be used efficiently. One way to efficiently utilize large amounts of changing data is to increase the batch size. To determine how much increase in batch size is necessary for Q-learning with a limited-capacity replay buffer to take advantage of the large amounts of incoming data, we investigated the relationship between performance and batch size. Many prior works have shown that using a large batch size can improve network performance, such as in contrastive learning settings [114], [115]. In our work, we found that large-batch training can notably improve the learning speed in off-policy RL for massively parallel simulations, as shown in Figure 4.8. However, if the batch size is too big, the learning speed can be slowed down. This is because GPUs have a limited number of CUDA cores, and it takes more time to process a very big batch of data once the batch size is above some threshold value, which is another underlying trade-off.

Effect of Replay buffer size

As discussed in Section 4.1, when dealing with tens of thousands of parallel environments, the replay buffer with normal capacity (e.g. 1M) gets a full refresh for every several hundreds of environment steps. This means that the replay buffer will not contain too much historical data. Apriori, one might think off-policy methods will fail in this case and that we need to proportionally increase the replay buffer size to store more experience data. However, surprisingly, we empirically found that even with thousands of parallel environments, having a “small” replay buffer (1M or 5M) can still lead to good performance. In Figure 4.9, we show how the learning curves change as we vary the buffer capacity. We can see that, in all cases, the policies learn well. We hypothesize that PQL still works well in this case because a large number of parallel environments can generate diverse enough data in a few environment steps. Moreover, $|B| \in \{1, 5\}M$ leads to faster policy learning at the beginning of the training than $|B| = \{10, 20\}M$. We hypothesize that this is because a smaller replay buffer allows the old and less informative samples to be replaced much faster, which is more important in the

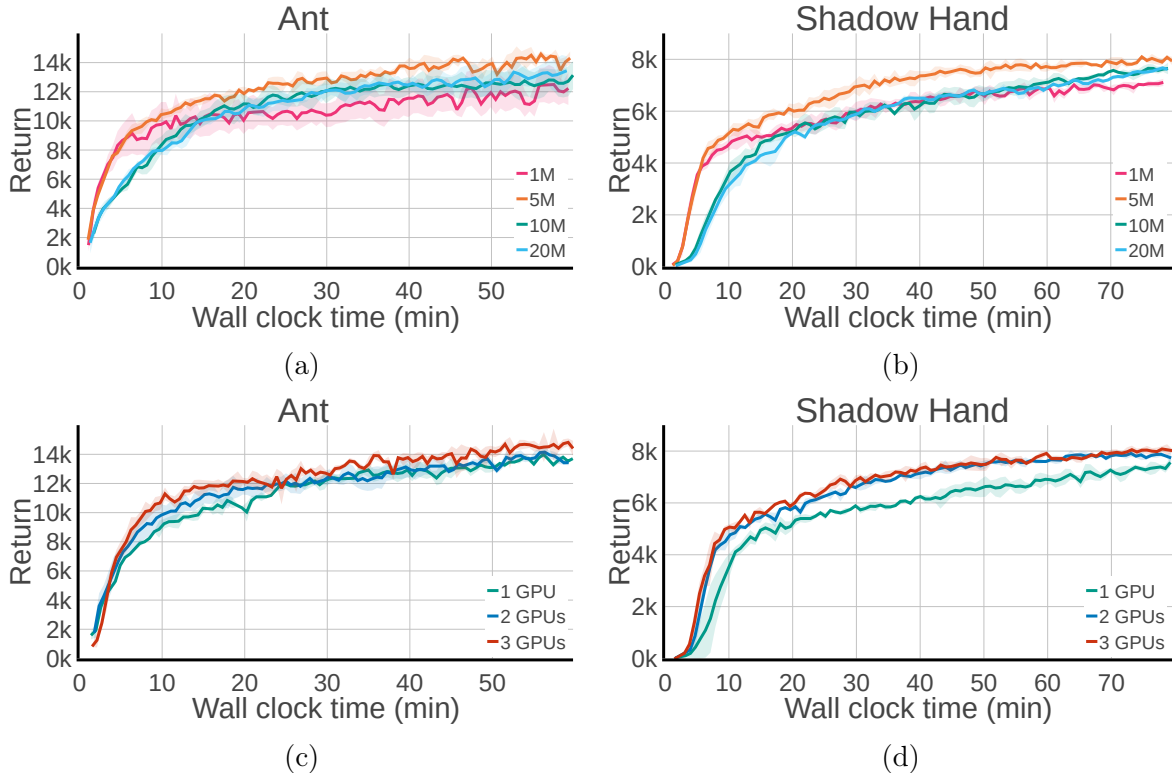


Figure 4.9: (a) and (b): effect of different replay buffer size. (c) and (d): effect of number of GPUs used for running PQL. PQL can be deployed on a flexible number of GPUs. In complex tasks such as *Shadow Hand*, it is beneficial to have at least 2 GPUs where the **Actor** runs on a separate GPU as the simulation itself consumes more GPU compute as the task complexity increases.

early stages of training. The converged performances are similar when $|B| = \{5, 10, 20\}M$. However, the converged performance is slightly worse when $|B| = 1M$. This may be because a small replay buffer, which discards new data too quickly, can have a negative impact on learning in the end.

Number of GPUs

Nowadays, it is common to have workstations with multiple GPUs. Running different processes on separate GPUs can potentially speed up learning. Our PQL scheme can adapt to different numbers of GPUs available on a workstation. Specifically, **Actor**, **P-learner**, and **V-learner** can be placed on any GPU. To investigate the performance variation when we distribute the three components across different numbers of GPUs, we conducted experiments with three scenarios on Tesla A100 GPUs: (1) place all three processes on the same GPU, (2) place the **Actor** on one GPU, **P-learner** and **V-learner** on another GPU, (3) place the **Actor**, **P-learner**, **V-learner** on a different GPU respectively. In the two-GPU case, we allocate **Actor** to a dedicated GPU because simulating many tasks with a large number of environments can cause high GPU utilization. As shown in Figure 4.9, our PQL scheme works well in all three scenarios with one, two, or three GPUs. When the task becomes more

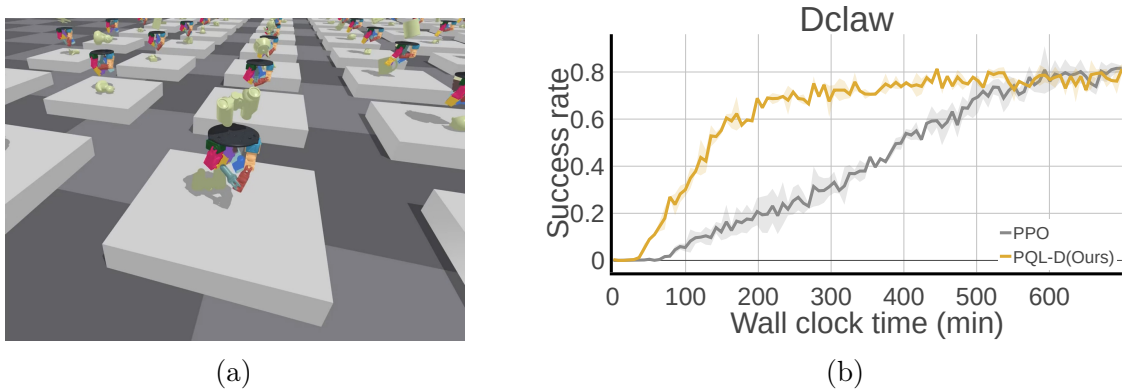


Figure 4.10: (a): *DClaw Hand* task. (b): We compared our proposed PQL-D method with PPO.

complex like *Shadow Hand*, the simulation takes much more computation and time. Putting all three processes will slow down each one of them due to full GPU utilization, which is why we see a bigger gap between the 2-GPU or 3-GPU training and 1-GPU training on *Shadow Hand*. Therefore, it is beneficial to place the **Actor** on one GPU and **P-learner** and **V-learner** on other GPUs.

4.4.5 Additional Tasks

Vision-based *Ball Balancing* task Simulating vision-based tasks is much slower and more demanding on the GPU as each simulation step involves both the physics simulation and image rendering. To demonstrate the generality of our scheme in operating in this practical setting of vision-based training, we consider a vision-based *Ball Balancing* task [30].

Since directly learning a vision-based policy with RL is time-consuming, we use the idea of asymmetric actor-critic learning [77] to speed up vision policy learning. Image data is compressed using the *lz4* library to reduce the bandwidth requirement and communication overhead. More setup details are in Appendix B.2.3. As shown in Figure B.1, PQL achieves better sample efficiency and higher final performance than PPO with $N = 1024$ parallel environments.

Reorient hundreds of objects with a *DClaw Hand* We conducted further experiments on a contact-rich dexterous manipulation task [19], *DClaw Hand*, as shown in Figure 4.10a. This task is much more challenging than the *Shadow Hand* task and *Allegro Hand* task because it needs to learn to reorient hundreds of different objects with a single policy. Furthermore, the control frequency (12Hz) is much lower than the default control frequency (60Hz) used in all six proposed benchmark tasks. This means that the simulation takes much longer to run between each policy command step, and the **Actor** process will be much slower. In Figure 4.10b, we observe that our method reaches 70% success rate around 200 minutes, which is approximately 3 times faster than PPO.

4.5 Discussion and Future Work

We present a scheme **PQL** for scaling up off-policy methods with tens of thousands of parallel environments on a single workstation. Our method achieves state-of-the-art results on the Isaac Gym benchmark tasks in terms of the training wall clock time. The driving force behind this success is the parallelization of data collection, policy function learning, and value function learning. We provide a mechanism to balance and control the speed in different processes, which leads to better and more stable performance across different hardware conditions or when the GPU resource is limited. Although PPO requires a large number of environments to work on complex tasks such as *Shadow Hand*, PQL is more lenient on the number of environments and works well on a wide range of different numbers of environments. With a large number of parallel environments, it is beneficial to use a big batch size for training agents, with the caveat that if the batch size is too big, it might take the GPU more time to process the batch data and lead to a slowdown in policy learning. We also found using different exploration scales in different environments achieves better or similar performance compared to a carefully-tuned exploration scale in all parallel environments, which means we need less hyper-parameter tuning. Even though the number of environments is $1000\times$ more, we did not find it necessary to use a replay buffer that is $1000\times$ bigger. In fact, a replay buffer with a capacity of 5M transitions is sufficient for our experiments even with 16843 parallel environments. Our scheme’s hardware requirements are flexible and work well with different numbers of GPUs and various GPU models.

In this paper, we experimented with default task configurations for the Isaac Gym benchmark tasks. Therefore, the reported results are for tasks without extensive domain randomization. The investigation of how well PQL performs in the presence of extensive domain randomization is left for future work. Another interesting direction is to explore better sampling strategies for the replay buffer. PQL does not use techniques such as prioritized experience replay [97], which could improve sample efficiency but significantly hurt wall-clock time efficiency due to massive amount of collected data. Therefore, new strategies should be considered, such as rejecting samples given the massive amount of data. It would also be practical to study different exploration strategies that can take advantage of parallel environments. When using PQL in other tasks, we have observed that if the agent is in joint position control mode, mixed exploration strategy tends to work better when the action space is defined on relative joint position control rather than on absolute joint position control. We hypothesize that exploring the full action space ($[-1, 1]$) in absolute joint position control leads to many useless explorations, and thus data from a significant portion of the parallel environments may not provide much value. In such cases, it would be interesting to investigate how to adaptively change the maximum range of exploration noise throughout the training process. Lastly, our scheme can be easily extended to a system with multiple parallel learners or value learners given the decoupling of policy and value learning. In this case, it would be interesting to apply ensemble methods or evolutionary strategies to further exploit the massive amount of data.

4.6 Acknowledgements

I would like to thank Zechu Li, Zhang-Wei Hong, Anurag Ajay, and Pulkit Agrawal for their contributions to this project. In particular, I want to highlight Zechu Li's role as he co-led the project and played a pivotal part in helping to complete it successfully.

Chapter 5

Tactile sensing with a dexterous hand

5.1 Overview

Consider the setup in Fig 5.1 wherein multiple objects are placed in a box with unknown positions and orientations. A multi-fingered robot is tasked to fetch a particular object using only tactile observations from sensors placed on fingertips, without access to visual observations. Such problems are commonly encountered in daily life — retrieving a desired object from the backpack, inside of a drawer, or from a tall cabinet where the topmost shelves are not visually observable. In these situations, the sense of touch can compensate for the lack of visual observations. To the best of our knowledge, the capability of a robotic system successfully retrieving a desired object based only on tactile observations has not been demonstrated in scenes with multiple freely moving and unknown objects.

To understand what makes vision-free object retrieval challenging, first consider a scenario where the robot can visually observe the scene. In such a case, it is (typically) possible to identify the location and identity of all objects from just a single image. Furthermore, the same image communicates enough information about object geometry to plan a grasp. In contrast, tactile observations made by the fingertips are local and sparse. Fingertips only make infrequent contact with a local area of the object. Therefore the robot first needs to explore to localize objects. Then it needs to plan a sequence of touches on each object to gather enough information to identify and grasp it. Unlike visual data, where a single observation is usually sufficient, tactile-based object retrieval requires careful planning of a long sequence of actions for information gathering and mechanisms for temporal integration of this information.

Usually, one would expect that as the robot interacts more with the environment, it gathers more “information”. However, this is not always true: generating touch readings requires applying force on an object that might move it. Such motion creates difficulty both in localization and identification. During localization, a moving object might contact another object previously encountered by the robot and potentially invalidate existing state estimates such as the object’s location and pose. Similarly, during classification, if the object moves while being touched, the shape estimation is noisier which can make object identification difficult.

To summarize, the two key problems that make object retrieval in the absence of vision

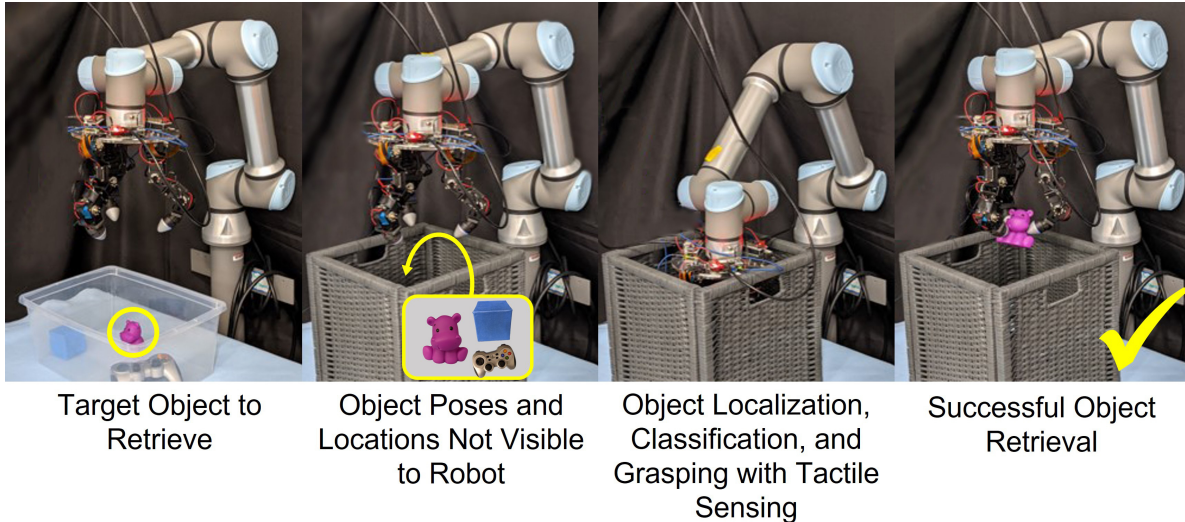


Figure 5.1: System setup showing object retrieval from an occluded bin using only tactile sensing in the real world. The agent has to localize and identify the object from fingertip tactile sensors alone

challenging are: **Firstly**, there is significant partial observability resulting from local and sparse touch observations. **Secondly**, obtaining touch readings requires force application on objects which may, in turn, lead to a change in object pose. While prior work has studied object classification and localization using tactile observations [116], [117], this has primarily been limited to single object scenes with a few classes of stationary objects.

We present a system that overcomes these challenges and is capable of exploring and retrieving novel objects only from local fingertip tactile observations in the presence of several movable objects, albeit in a simplified setup where individual objects are separated. Our solution employs a localization scheme that minimizes object motion while clustering to find object positions. Once objects are localized, the robot performs directed exploration to infer object shape by making careful touches around the determined object location. The shape information gathered from the sequence of contacts is represented as a point cloud. We leverage self-supervised contrastive learning [118], [119] to embed tactile point clouds into a feature space used for object identification. To retrieve localized and identified objects, a simple grasping system is deployed. We present results on a 3-fingered robotic hand-arm system in both simulation and the real world. Our system achieves 76% success in simulation and 60% in the real world at retrieving novel objects without visual feedback.

5.2 Related Work

In recent years, much progress has been made in the development and usage of tactile sensors to improve a robot’s ability to complete dexterous manipulation tasks [120]. In particular, sensors utilizing a variety of transduction methods, such as resistance [121], capacitance [122], piezo-electrical [123], magnetic [124], and optical [125]–[127], have been shown to help robots more intelligently interact with the physical world [127]–[130]. In our work, we leverage

tactile sensing to perform object retrieval completely in the absence of visual input.

5.2.1 Object Classification with Tactile Sensors

While there has been much work in object classification using a combination of image and tactile information [131]–[133], our problem requires us to rely only on tactile data for classification. [134], [135], and [126] classified different fabrics, textures, and basic geometries (respectively) using only camera-based tactile sensors. Others, such as [116], [136], [137], used force-based tactile sensors to accomplish the classification task but often assumed the object was static. [138] aims to do tactile-based classification by leveraging multi-modal information across vision and touch. A class of methods also aim to make full shape inference from tactile feedback [139]–[143] but typically assume the object is static and there is only a single object in the scene. In contrast, in this work, we also operate in the tactile-only regime, but we interact with multiple non-static objects when collecting the surface touches used in classification and just look to make object identification, not ideal shape reconstruction. Moreover, we are able to operate in a regime where we do not require any explicit object models or knowledge of object dynamics at test time, making the method easy to apply broadly to a variety of novel objects.

5.2.2 Object Localization with Tactile Sensing

Localization and manipulation of objects using only tactile sensing have been explored in prior work. While some works [144], [145] focused more on the dynamics and kinematics of the robot to perform localization, Li et al. [146] used a GelSight sensor to localize the sensed portion of single small objects like USB cables relative to the gripper. Bauza et al. [117] adopt a similar approach, combining heightmap information obtained from a tactile sensor with the robot’s kinematics and the iterated closest point method to localize and even identify small objects. Other works [147] approach the localization problem by also building a visual map, but do so assuming objects are fixed. [148] perform contact-based localization to perform accurate manipulation when the object is in hand. In contrast to these works, we are less focused on fine-grained, precise, and in-hand localization than localization strategies that can deal with multiple, much larger movable objects while also performing identification and grasping. We compromise significantly on accuracy to obtain a general strategy that can approximately localize objects without requiring known object shapes and dynamics by aiming to keep the scene static.

[149] goes beyond static objects and uses a SLAM approach to do planar object reconstruction and localization of non-static objects in the plane in single object scenes, leveraging a Gaussian process implicit surface method and particle filtering. Methods like [150], [151] also attempted to use only tactile sensing to retrieve a non-static object in a single object scene using particle filtering methods. Additionally, [152] uses a pre-touch sensor to localize and then senses object properties like stiffness and sliding for object discrimination. In contrast, in our work, we interact with multiple movable objects with a dexterous 3-fingered manipulator and perform the whole pipeline of localization, identification *and* grasping for object retrieval only using fingertip tactile sensors.

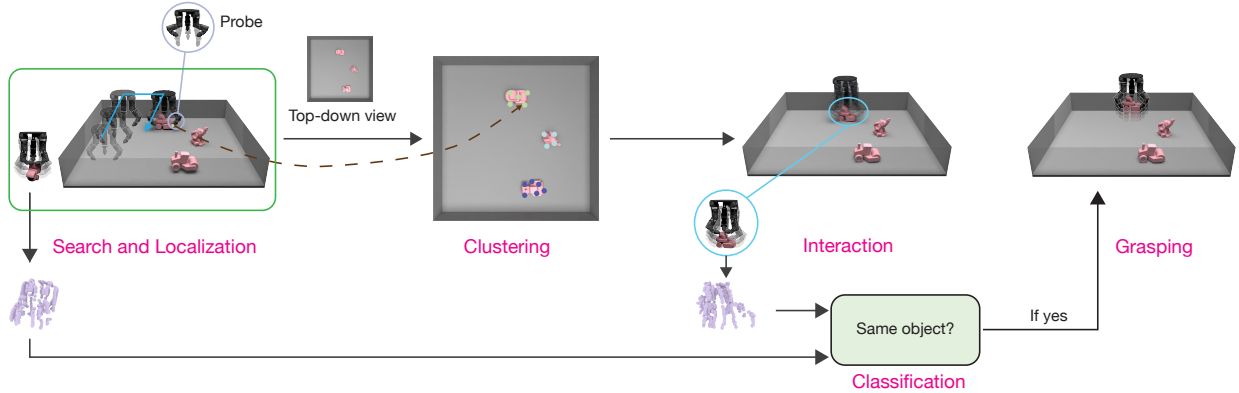


Figure 5.2: Depiction of the full pipeline for tactile-only object retrieval. The agent first localizes several objects in the scene by vertically probing in a discrete grid around the environment, while applying minimal force. The object is then interacted with by radially tapping it gently for identification. The collected data is then used to identify if the object matches the object it is tasked to retrieve and then grasping is performed to actually retrieve the object.

5.3 System Description

Fig 5.1 illustrates our system consisting of a three-fingered robotic hand mounted on a UR5e robotic arm (controlled by [153]), equipped with fingertip GelSight 360 [154] for touch sensing. **Robotic Hand:** as shown in Fig 5.1, we use the three-fingered D’Claw robotic hand system [58]. Each finger has three Dynamixel servo motors connected in series for a total of nine degrees of freedom. The fingers are position controlled. **Tactile Sensor:** we use omnidirectional GelSight sensors [154] on the tip of each finger of the D’Claw hand. These sensors contain fisheye camera lens surrounded by a soft elastomer gel that deforms when the sensor makes contact. The deformations are recorded as a RGB image by the camera. This GelSight sensor has a significantly larger area of contact (approximately 15 cm²) with objects than most sensors since it wraps completely around the finger. This is important to actually help identify objects with fewer interactions. To obtain binary contact readings from the sensor’s raw data (RGB images), we train a force recognition model (as described in [155]) in the real world to map images of contact from the GelSight sensor to binary labels of whether contact occurred or not.

5.4 Problem Setup

We first task the agent by allowing it to interact with a given *target object* at a known location and pose, as shown in Fig 5.2. The target object is then placed in a planar bin area along with other *distractor objects*, each in a random unknown pose. The agent’s goal is to leverage only tactile sensing to retrieve the target object. All objects are rigid and movable, and the agent has no prior knowledge of distractor object shapes. Furthermore, we assume no objects are stacked on each other for tractability. The robot observes fingertip positions

$\{p_i \in \mathbb{R}^3, i = 1, 2, 3\}$ estimated through joint encoders, as well as a reading of binary contact $\{c_i \in \{0, 1\}, i = 1, 2, 3\}$ for each finger.

5.5 TactoFind: Identifying and Retrieving Objects in the absence of Visual Feedback

The problem of blind identification and retrieval of novel and movable objects is an instance of *partially observable Markov decision process*, which is known to be intractable in general. To make this problem more approachable, instead of solving end-to-end, we can take a step-wise approach illustrated in Fig 5.2 and make assumptions detailed below. The robot first performs object localization, then instance identification, followed by grasping. In the *localization phase*, the location of all objects is estimated agnostic of their identity. During *instance identification*, an exploration algorithm is deployed to interact with each object. The contact data resulting from the interaction with each object is matched against the contact data of the target object for identification. Lastly, the position of the identified object is used to *grasp* and retrieve it. In each step, both interaction and inference algorithms are designed to minimize object motion and to integrate sparse tactile information for dealing with the challenges mentioned in Section 5.1. We train the identification model in simulation and transfer it to the real world, whereas the localization and interaction strategies are directly implemented in the real world.

5.5.1 Object Localization with Tactile Sensing

For the task of interest, an object localization The main challenge in object localization is that object motion while localizing the N -th object may disrupt the estimates for any of the $N - 1$ objects localized previously. Minimizing object motion is further complicated when object masses, shapes, and friction properties are unknown. Assuming that no two objects are vertically stacked and that objects do not touch each other, our approach to object localization consists of two steps: first, the workspace is discretized into a square grid (H) with fixed side length $\delta = 5cm$ (roughly half the average width of the target objects) and every location on the grid is probed to estimate if it is occupied by the object. The result is a binary *object occupancy map* (i.e., $H[i, j] \leftarrow \{0, 1\}$). Next, H is clustered to obtain approximate locations for each object.

To obtain the occupancy map, $H[i, j]$ we move the gripper to the center of the grid location (i, j) and at a pre-specified height above the bottom of the bin/table. The gripper is commanded to move down along the normal to the plane (Fig 5.2). To minimize object motion, as soon as any contact is encountered or if the tip of the gripper reaches the surface of the table, the arm’s motion is stopped. If contact was encountered, the grid position is deemed to be occupied (i.e., $H[i, j] = 1$). The procedure is repeated for all grid locations. The occupancy map is clustered using K-means algorithm [156], with a known value of K corresponding to the number of objects in the bin. This procedure yields a list of approximate object center positions $[o_c^1, o_c^2, \dots, o_c^N]$ (Fig 5.5), which can then be utilized for fine-grained object identification.

5.5.2 Dynamic Object Identification

The localization process above provides an occupancy map and object locations, which is insufficient to estimate the 3D shape of objects required to perform identification. Prior works have found that random One naive strategy to get more interaction data is to randomly move the fingers, and once in a while, the fingers will touch the object. However, such a naive strategy makes it hard to identify objects. The reason is that the object might move a lot due to random finger motions, causing noise in the data. When the object does not move, the fingertip locations when fingers make contact with the object represent the points on the object’s surface. However, if the object moves a lot, the touch points are effects of both the object’s geometry as well as the object’s motion, which makes it more challenging to identify what object it is. Therefore, it is beneficial to have an interaction policy that explores the object’s surface but also tries to minimize the object’s motion.

After collecting such data, the next step is identifying whether the object is the target object we are searching for. A straightforward way is to train a classifier that outputs the object ID and see if it matches the target object ID. However, such a classifier cannot generalize to novel objects. Therefore, instead, we use contrastive learning to learn an embedding space in which we can determine whether the object is the same as the target object based on the distance between their embedding vectors.

Collecting Interaction Data Instead of having the hand doing unstructured exploration around the object, which obtains very few useful interaction data and can cause the object to have a big motion, we devised a radial sliding strategy. It tries to move the fingers to follow the contours of the objects in the vertical direction. We hypothesize that such slices of object contours along the vertical axis would be sufficient for identifying an object (as described in the next section) since they give a notion of object shape. However, in our experiments, we found that having the fingertips move along the object contours when object geometry is complex tends to cause the object to move. Instead, we command the fingers to *tap* along the object contours (*radial tapping strategy*). Tapping significantly reduces the object motion and allows us to identify free objects much more effectively than sliding. This is because the number of points of contact is reduced and the force inward can be controlled more carefully. As shown in Fig 5.3, we first reset the fingers to their initial positions, and then close the fingers (move the fingertips inwards to touch the objects), when touch happens on any of the fingertips ($c_j^t = 1$), we record the fingertip position p_j^t and stop moving the finger. Since each touch only generates one data point (the fingertip position p_j^t), we get up to three points at each time we close the fingers (potentially one from each finger). We move the fingers upward by a small distance and repeat such a process until the fingers touch the top of the object. By doing so, we get a sequence of contact positions $\mathcal{P}_o^x = \{p_0^0, p_1^0, p_2^0, p_0^1, p_1^1, p_2^1, \dots, p_0^N, p_1^N, p_2^N\}$ for a series of N taps for object o at an unknown pose x , where each contact point p is a point in \mathbb{R}^3 .

Some things to note — Firstly, the actual contact forces do not need to be used since we are just detecting binary contact on the surface. Secondly, the object itself may move since the fingers are not *guaranteed* to perfectly cage the object while the contact is being made. To account for potential object motion, we perform object relocalization by estimating the direction of motion from the contact points at each time step. Specifically, at every point

in time, we maintain a current estimate \hat{c} for the current object center. Each radial tap gives us a sequence of contact points $\{p_i\}$, which we average to get a single point p . We then perform an exponential smoothing update $\hat{c}_{new} = \gamma\hat{c}_{old} + (1 - \gamma)p$ to get a smoothed new estimate, and move the hand in the xy -plane to \hat{c}_{new} . This allows us to correct for local object motions that may occur when interacting with the object without knowing the object model beforehand. Note that our goal is not to accurately estimate the center of the object. A rough estimate is sufficient for the hand to track its location. As long as the three fingers can surround the object, we can collect meaningful interaction data.

Representation Learning for Object Identification The contact information we get from the interaction policy is a sequence of contact positions \mathcal{P}_o^x , which can be viewed as a very sparse point cloud. Our goal is to identify whether the obtained point cloud $\mathcal{P}_o^{x_1}$ and the point cloud obtained by interacting with the target object $\mathcal{P}_{o_{tgt}}^{x_2}$ belong to the same object. Note that even if they are the same object, the object can be in different poses. Therefore, we need a shape representation that is agnostic to the object orientation but still allows distinguishing between different objects. We can learn such representation using contrastive learning [119], [157]–[159] — represent touches on the same object at different poses similarly and touches across different objects differently. One potential representation learning objective is the InfoNCE loss (as also discussed in [119]). Assuming we have some parametric encoder f with parameters θ , this loss can be expressed as:

$$\mathcal{L}_{\text{NCE}}(\theta) = \mathbb{E}_{(x, x^+) \sim \mathcal{D}_o, \{x_k\} \sim \mathcal{D}} \left[\log \frac{\exp f_{\theta}(x) \cdot f_{\theta}(x^+)}{\sum \exp f_{\theta}(x) \cdot f_{\theta}(x_k)} \right]$$

where the positive distribution \mathcal{D}_o consists of touch sequences sampled from different poses of the same object o and the negative distribution \mathcal{D} is touch sequences sampled from arbitrary poses of different objects. We treat the touch data (sequences of contact positions \mathcal{P}_o^x) as a sequential time serie, and parameterize the encoder f_{θ} with a transformer architecture.

Once the representation is learned, the object of interest can be identified by comparing the cosine similarity of touch sequences in the representation space $z = f_{\theta}(\mathcal{P}_o^x)$ of the touches made blindly across various objects in the bin with the touches made on the target object outside of the bin. This identified object can be used for object retrieval using a grasping controller.

While more complex schemes could be developed for grasping [34], [160], we found that the hand lends itself very naturally to grasping with a simple caging policy. Concretely, we simply close the fingers until the fingertips touch the object with some amount of force. The touch sensors on the fingertips are used to tell whether the fingertips touch the object with a sufficient amount of contact area. After all three fingers touch the object with a sufficient amount of contact, we lift the arm to grasp the object. We depict this policy in Fig 5.2, and deploy it in an object-agnostic manner to grasp objects which have been localized and identified.

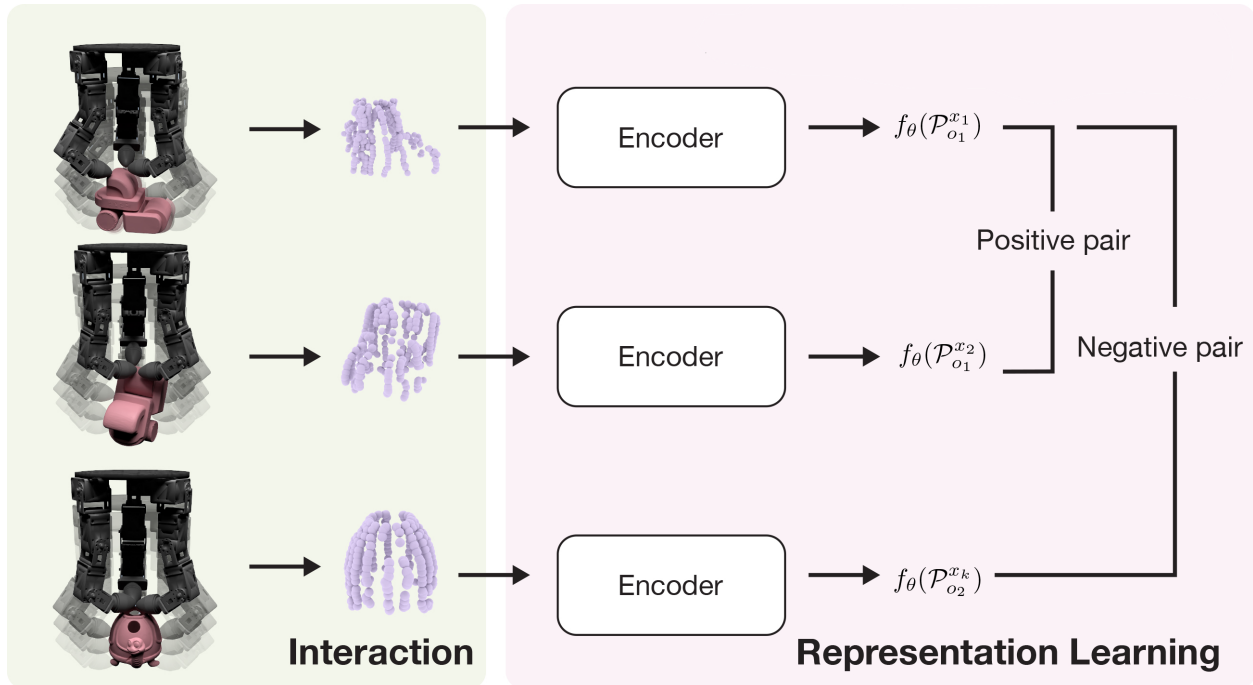


Figure 5.3: Depiction of the contrastive representation learning architecture for object identification. The sequence of contact points goes through a transformer based encoder to provide embeddings that are trained with the InfoNCE loss. Affinity in this representation space can then be used for object identification

5.6 Experimental Evaluation

In this section, we aim to answer the following questions: (1) How effective is the overall pipeline at identifying and grasping particular object instances from tactile sensing alone? (2) Can a dexterous hand with tactile sensing localize movable objects without visual input? (3) Is the data obtained by interacting with objects using the radial tapping strategy described in Section 5.5.2 effective for classification? (4) Is the radial tapping interaction strategy for object identification effective for movable objects? (5) Is representation learning using the scheme in Section 5.5.2 effective for identifying novel objects?

We used the 150 object meshes collected in [19]. The meshes are from various datasets including Google Scanned Objects [161] and ShapeNet [162]. We built the simulation environment in PyBullet. The 3-D printed versions of the objects were used for testing in the real world. For evaluation, we create test scenes by randomly choosing K objects from this set and placing them in random poses in a bin.

5.6.1 Baselines and Evaluation Metrics

While no entire system exists that completes the entire task described in this work, we compare it with several prior works that perform localization, identification, or grasping. For localization, we compare with [150] which uses particle filters for localization. To understand the importance of our particular interaction strategy, sensing modality and other design

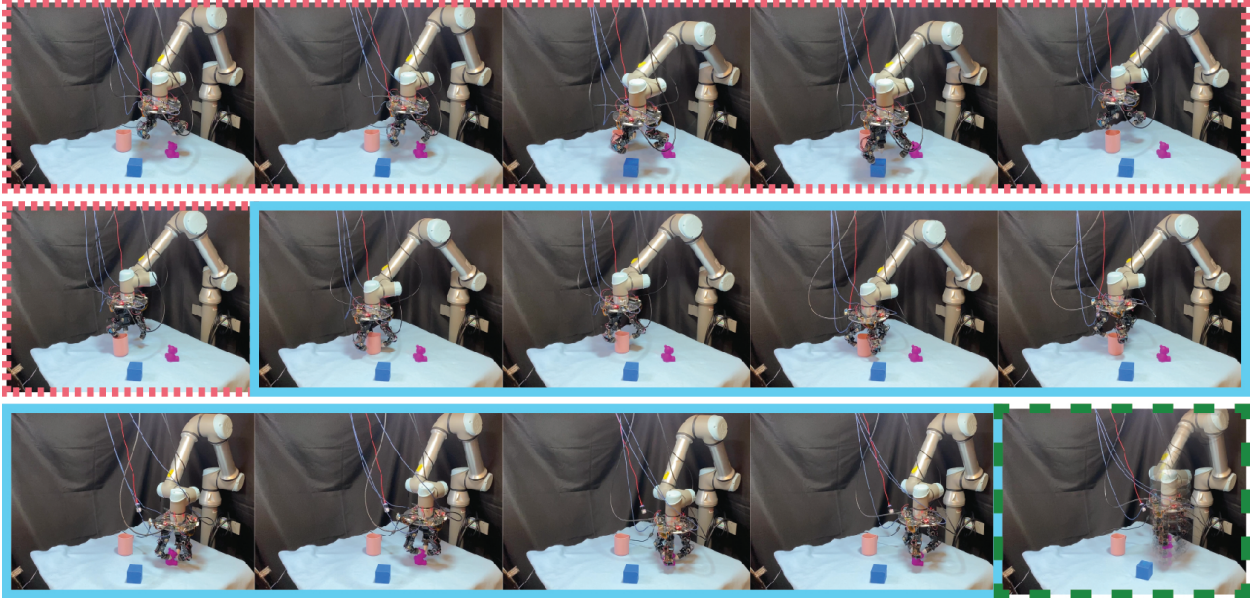


Figure 5.4: Film strip depicting phases of real world localization and identification with our pipeline. We see that objects move minimally, while shapes are successfully identified and grasped. Red box shows the localization, blue box shows the interaction, and the green box shows the grasping.

choices in terms of being able to collect information without moving the object significantly during object identification, we compared with two baselines: 1) radial tapping with noisy contact detection rather than accurate binary contact direction (noisy contact) to test the importance of accurately binarizing contact, 2) radial tapping without hand relocalization in cases where the object slips as discussed in Section 5.5.2 (no relocalization) to show the importance of the relocalization strategy. To understand the impact of specific model architecture, we compared with two baselines: 1) an LSTM [116] rather than a transformer with the InfoNCE objective, 2) using Triplet loss [163], [164] instead of InfoNCE loss function, to understand the importance of the particular choice of transformer and infoNCE objectives.

Evaluation Metrics: For localization, we measure the error (in cm) between the identified object center and the ground truth object center of mass (clustering error). Additionally, we measure displacement from the original object positions to determine how disruptive exploration is (perturbation error). We evaluated this for scenes with 3 objects. For identification, we measure the success percentage in identifying the correct object out of 5 unique object instances that are randomly placed in the bin, as well as the number of successful taps made with the object.

5.6.2 Simulation Results

We report results on localizing objects in different scenes in simulation with various different objects in Table 5.1. We see that our proposed clustering scheme gets within 3.6 cm on average across various objects while ensuring minimal displacement of only 1.7 cm during exploration when evaluated on test objects approximately 10 to 15 cm in diameter. In

comparison, the particle filtering scheme used in [150] performs competitively with ours on scenes with single objects, achieving an average of $4.5cm$ of error. However, when increasing the number of objects to three, the particle filtering struggles and the error increases to $6.4cm$ on average. This is likely due to the fact that increasing the number of objects also increases the dimensionality of the particles, meaning exponentially more particles are needed to achieve more accuracy.

Table 5.1: Results on object localization in simulation. For both strategies, we measure the success rate (defined as when each predicted center is within $\delta = 7.5cm$ of the object center), the average prediction error, and the average displacement of the objects during the localization. Our clustering strategy scales to multiple objects much better than particle filtering, likely because multiple objects introduce a higher dimensional search space.

Method	Success Rate	Center Error	Object Displacement
Ours (1 object)	$99.2\% \pm 0.2\%$	$3.2 \pm 0.1cm$	$1.8 \pm 0.1cm$
Particle Filter (1 object) [150]	$94.6\% \pm 0.6\%$	$4.5 \pm 0.3cm$	$1.5 \pm 0.1cm$
Ours (3 objects)	$91.3\% \pm 1.7\%$	$3.6 \pm 0.2cm$	$1.7 \pm 0.1cm$
Particle Filter (3 objects) [150]	$52.8\% \pm 1.4\%$	$6.4 \pm 0.1cm$	$2.4 \pm 0.1cm$

Next, we show in Table 5.2, that our proposed identification technique is able to select the correct object out of five novel objects with 69.8% accuracy.

Interaction: To understand why the interaction strategy is better than alternatives, we study the impact of removing individual elements of the interaction strategy. Table 5.2 shows the results of these ablations. First, we add random noise into the sensors, compromising the system’s ability to quickly detect contact. This results in identification success dropping from 69.8% to 54.4%, likely due to greatly increased object motion. We also investigate the effects of removing the relocalization policy described in Section 5.5.2. Without the ability to adapt to object motion, identification accuracy drops to 58.5%, a drop of over 10%. Therefore, we can conclude that both our relocalization policy, as well as accurate contact detection, are necessary to achieve our system’s accuracy.

Identification: From Table 5.3, we see that the transformer with the InfoNCE objective achieves a higher success rate than the alternatives. In particular, training with InfoNCE loss improves performance over using Triplet loss

Method	Accuracy (Validation Set)	Accuracy (Training Set)	Successful Taps
Ours	$69.8\% \pm 1.2\%$	$88.4\% \pm 0.9\%$	228.4 ± 0.7
No Relocalization	$58.5\% \pm 1.4\%$	$61.8\% \pm 1.3\%$	157.1 ± 0.8
Noisy Sensors	$54.4\% \pm 1.4\%$	$59.3\% \pm 1.4\%$	183.8 ± 0.7

Table 5.2: Effectiveness of interaction strategies on object identification in simulation. We measure for each policy the overall accuracy when selecting out of five (previously unseen) objects on both the training and validation object sets, as well as the average number of successful taps the policy gets from the object. These experiments show that both our object position estimation as well as accurate contact detection are essential for successful object identification.

Method	Accuracy (5 objects)	Accuracy (3 objects)
Transformer + InfoNCE loss (Ours)	69.8% \pm 1.2%	80.3% \pm 1.0%
LSTM + InfoNCE loss	65.7% \pm 1.3%	77.0% \pm 1.2%
Transformer + Triplet loss	52.3% \pm 1.4%	64.8% \pm 1.3%

Table 5.3: Effects of architecture and objective on object identification in simulation. Using a more expressive architecture like a transformer helps with classification, while the InfoNCE loss outperforms using a triplet loss.

Finally, in simulation, we evaluate the combined pipeline of object localization and identification (leaving grasping for real-world evaluation in Section 5.6.3) in terms of the percentage of trials where the successful object is identified and localized. We find in Table 5.4 that our pipeline is able to accomplish a success rate of 76.8% in simulation.

5.6.3 Real World Results

Success Rate	Overall	Localization	Identification
Sim	76.8% \pm 2.3%	91.3% \pm 1.7%	80.3% \pm 1.0%
Real	59.4% \pm 11.3%	81.0% \pm 9.0%	75.7% \pm 9.8%

Table 5.4: Results on full pipeline object retrieval in simulation and the real world. We find that there are expected drops in performance from simulation to the real system, but the identification is still able to significantly outperform random chance

The pipeline transfers relatively well to the real world. Due to the lack of ground truth object positions running on the real system, we qualitatively evaluate pipeline stage success. Specifically, we deem localization to be successful if the robot is able to get its three fingers around each object, and grasping to be successful if the robot is able to lift the chosen object. When evaluated on three objects at a time, we find that localization has a success rate of 78%. The object identification model trained in simulation has a success rate of 73% in the real world. And similarly, grasping is able to accomplish a success rate of 71%. When this entire system is executed sequentially, it accomplishes a success rate of 59.4%, as shown in Fig 5.4. While this leaves room for improvement, in each part of the pipeline, it is significantly better than chance and much better than a random policy.

5.6.4 Ablations

To understand how much moving objects affect classification performance, we perform an ablation study in Table 5.5 repeating the comparisons in Section 5.6.2 with a fixed stationary object. We find that classification is significantly easier, showing the difficulty of scenarios with moving objects.

To understand just how much these methods help with moving objects, we also ran an ablation study where we performed comparisons in Table 5.6 as we change object friction and mass. The localization performance degrades as we use small friction and mass values, especially low friction. This is expected as on a slippery surface, an object moves more easily

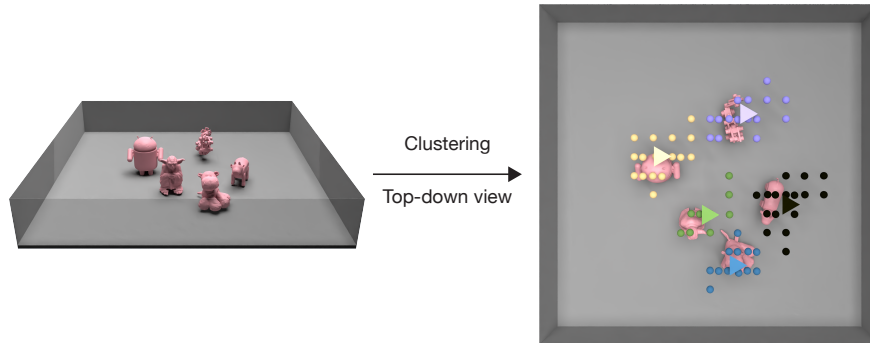


Figure 5.5: Visualization of planar occupancy and resulting clusters (the dots) during localization compared with true object centers (the triangles). While showing non-zero error, the relative object centers are well predicted enough for subsequent object identification.

Method	Accuracy (5 objects)	No. of Successful Taps
Ours (Moving Objects)	$69.8\% \pm 1.2\%$	228.4 ± 0.7
Ours (Static Objects)	$86.2\% \pm 0.9\%$	235.4 ± 1.2

Table 5.5: Effectiveness of interaction strategies on object identification in simulation for *static* objects. We run our object interaction strategy on both moving and static objects and find that static objects are easier to classify than moving ones, showing how challenging our problem setting is

Friction Coefficient	$\mu = 0.5$	$\mu = 0.25$	$\mu = 0.1$
Localization Accuracy	$91.3\% \pm 1.7\%$	$82.6\% \pm 1.8\%$	$17.3\% \pm 1.7\%$
Identification	$69.8\% \pm 1.2\%$	$65.3\% \pm 1.3\%$	$53.8\% \pm 1.4\%$

Table 5.6: Effect of the coefficient of friction (μ) on localization and classification performance in simulation. As friction reduce, localization becomes much more difficult and identification accuracy also decreases due to increased object movement.

and suffers from large motion even if the robot hand applies a small force on it. However, our identification pipeline suffers a much smaller performance drop, indicating the capability of our method in handling non-static objects.

5.7 Discussion

The pipeline proposed in this work is only a starting point for tactile-only object localization and retrieval. While we have designed a strategy using tapping to minimize object movement, an interesting future research direction would be to explore how to localize, identify and grasp objects that can move substantially.

5.8 Acknowledgements

I would like to express my sincere gratitude to Sameer Pai, Megha Tippur, Edward Adelson, Abhishek Gupta, and Pulkit Agrawal for their invaluable contributions to this project. Sameer, Megha, and I co-led the project, collaborating closely to oversee its progress and ensure its successful completion.

Chapter 6

Discussion

This thesis explored techniques for training dynamic dexterous manipulation policies, accelerating policy learning for these challenging contact-rich tasks, and incorporating constraints for training dexterous manipulation skills for downstream applications.

It may seem counterintuitive that we can train a controller to reorient thousands of objects with different geometries without explicit knowledge of their shapes. One hypothesis is that the controller can implicitly infer the object shape information using a history of state information. However, even if the teacher policy is just a simple multi-layer perceptron, it can still perform well. This indicates the existence of a strong baseline that can reorient different objects with high precision without knowing any object shape information, either explicitly or implicitly.

Nevertheless, this does not imply that knowing object shape information is not useful. The overall precision and success rate of the reorientation controller can be further improved if it can leverage the object shape information. For instance, we found that the failure rates are much higher for objects with extreme aspect ratios, such as long and thin objects like a spoon. We discovered that if we train a controller specifically to reorient this type of object using the robot hand, we can achieve a very high success rate. However, reorienting these kinds of objects becomes challenging when we train the controller with a mixed set of different objects, even if we provide the policy with information about object shape. One hypothesis is that in the early stages of training, the policy finds an easy solution that does not utilize any object shape information and still achieves a good overall success rate across objects, leading the policy to ignore shape information. Once the policy performance plateaus after ignoring the shape information, it becomes difficult for the policy to further improve performance by learning to utilize shape information (i.e., a local minima). On the other hand, the ability to reorient objects within the hand is a valuable skill for many tasks, such as vegetable peeling, tightening screws, opening bottles, and more. When developing in-hand manipulation capabilities, it is crucial to consider the additional constraints imposed by the intended downstream application tasks. For instance, the task of vegetable peeling introduces specific requirements, like maintaining a firm grip after rotating the vegetable for stable peeling. Accounting for these task-specific constraints during system development can lead to more practical and effective solutions tailored to real-world applications.

The teacher-student framework proves to be a very useful tool in simulation for training dynamic dexterous manipulation tasks. Having access to the full state information of rigid

bodies in simulated tasks is a significant advantage. We can leverage this privilege to train policies much more efficiently. Typically, the teacher and the student policy share the same task objective. However, we also demonstrate the flexibility of using different task objectives to train each policy. In the vegetable peeling project, the teacher policy was trained to reorient the object in hand by a random orientation angle, while the student policy aimed to reorient the object for a random duration. Such reformulation expands the application scenarios of the teacher-student framework.

When training vision policies in simulation, a substantial portion of compute time is dedicated to image rendering. If the vision policy operates on point clouds, it can be beneficial to first train the policy with synthetic point clouds and then fine-tune it with realistic point clouds. Since all state information is available in simulation, we can create synthetic point clouds using the state data of rigid bodies and their CAD models. This process is highly efficient and avoids image rendering in simulation. As generating synthetic point clouds is fast and they contain more information than realistically rendered point clouds, training with synthetic data can substantially accelerate policy learning, as demonstrated in our experiments.

Future work

While our work advances the field of dexterous manipulation, it merely scratches the surface. Many research opportunities lie ahead for further exploration and discovery.

Precise and Robust Dynamic Dexterous Manipulation: Achieving highly precise manipulation with high success rates is challenging for dynamic dexterous manipulation tasks involving fast-moving objects with significant inertia. For instance, getting less than 0.1 rad orientation error in the system presented in Chapter 2 was challenging. Depth images in real-world settings exhibited substantial noise. Future work could investigate leveraging RGB images for control and incorporating tactile sensors on fingertips for contact detection to improve success rates.

Dexterous Manipulation of Challenging Objects: Manipulating objects with complex characteristics, such as small size or extreme dynamics like slippery or deformable surfaces, poses significant challenges. The issue of manipulating small objects is exacerbated by the use of large robot hands, such as the Allegro or Leap hands, commonly employed in research. Manipulating small objects like pens or forks is particularly challenging with these hands. Future studies could explore more compact hand designs to address this issue. Furthermore, manipulating objects with extreme dynamics, such as slippery or deformable objects, presents another layer of difficulty. Simulating the dynamics of deformable objects can be time-consuming, which limits the speed of data collection. More research is needed to investigate how to dexterously manipulate such objects.

Dexterous Manipulation for Real-World Tasks: Recent years have witnessed significant progress in in-hand object reorientation [15], [18], [19], [50], [62], [71], [72], [165]. However, grounding these dexterous manipulation skills in downstream tasks remains under-explored. For example, training robots to grasp a knife from a block, reorient it in hand, and chop vegetables would impose additional constraints and spur new research problems.

Scalable Learning Systems for Dexterous Manipulation: In this thesis, we looked at different manipulation tasks like reorienting objects in hand, peeling vegetables, and picking

up objects blindly with tactile sensors. We proposed many techniques to teach robots these skills. However, an open question remains: how can we scale up the learning process so that robots can master many different dexterous manipulation tasks? Two common approaches include learning from simulation data and learning from real-world data. In simulation, we can use reinforcement learning algorithms to train a simulated robot agent to learn skills. However, designing a good reward function for each task is not straightforward and can be very time-consuming and difficult. One interesting research direction is to investigate how to combine planning or optimization approaches [166] with model-free policy learning during the training time. This hybrid approach could potentially make finding effective policies easier in simulation environments. In the real world, one option is to set up a teleoperation system where humans control the robot and demonstrate the desired behaviors. We can then use imitation learning methods to train the policy. However, teleoperation with dexterous robot hands on dynamic, fast-moving tasks is challenging. Future work could explore more scalable solutions that allow robots to learn and perform many different types of dexterous manipulation tasks more efficiently.

Closing remarks: Robotics is a system engineering discipline that requires a holistic understanding of both the software and hardware components. Gaining this integrated perspective on the AI/software and physical/mechanical aspects has been crucial, affording me a more comprehensive view of the multifaceted challenges in robotics. Ultimately, it is this holistic mindset that has enabled me to make progress and achieve the results laid out in this thesis.

Appendix A

Visual Dexterity

A.1 Supplementary Methods

A.1.1 Nomenclature

\mathbb{B}	the Big dataset
\mathbb{S}	the Small dataset
\mathbf{o}	observation
\mathbf{a}	action command
$\bar{\mathbf{a}}$	smoothed action command
π	policy
r	reward
\mathcal{E}	expert
\mathcal{S}	student
γ	discount factor
α	smoothing factor for action
\mathbf{q}	joint positions
$\Delta\theta$	the distance between the object's current orientation and goal orientation
$\dot{\mathbf{q}}$	joint velocities
\mathbf{v}^o	object's linear velocity
$\boldsymbol{\omega}^o$	object's angular velocity
\mathbf{z}	embedding vector
$\mathbf{A}(\cdot)$	a sequence of action commands
Λ	the entire space of possible dynamics parameters
λ	dynamics parameters of a robot
f	frequency
F_d^o	disturbance force on the object
m^o	object's mass
$\bar{\theta}$	threshold value for orientation distance
\bar{q}	threshold value for joint velocity norm
\bar{a}	threshold value for action norm
\bar{v}	threshold value for linear velocity norm
$\bar{\omega}$	threshold value for angular velocity norm

\bar{p}	threshold value for object’s distance from the hand
ϵ_θ	a constant in reward function
\mathbf{P}	point cloud
l_j	j^{th} link on the hand
G	number of fingers
\mathbf{p}^{f_i}	fingertip position of i^{th} finger
$\boldsymbol{\tau}_t$	joint torques
\mathbf{p}^o	object center position
\mathbf{R}	rotation matrix
\mathbf{p}	position
M	number of links on a hand
C	robot
h	score function for trajectory similarity
$\mathcal{N}()$	Gaussian distribution
$\mathcal{U}()$	uniform distribution

A.1.2 Experiment details

Object datasets

We use the following object sets: **Big dataset** (\mathbb{B}) and **Small dataset** (\mathbb{S}). \mathbb{B} is used for training policies. It is a collection of 150 objects from internet sources such as Google Scanned Objects [161]. The chosen objects cover a wide range of complex and non-convex shapes, such as cars, shoes, and animals (see Figure A.1). We only choose objects that are asymmetric or only reflective symmetric, which mitigates the multi-modality issue in defining object poses [18]. However, such a choice for training does not restrict our system from reorienting symmetric objects, a claim we empirically evaluate. \mathbb{S} is used for evaluating generalization performance. It contains 12 objects from the ContactDB [64] dataset with no overlapping object shape with dataset \mathbb{B} . We use a subset of 5 objects from \mathbb{S} to evaluate out-of-distribution (OOD) performance in the real world.

Object dataset preprocessing In order to make the objects manipulable by the robot hands, we need to scale their meshes to proper sizes. In simulation, we center each mesh and manually scale each object mesh to a proper size compared to the robot hand size. Overall, the longest side of the objects’ bounding boxes lies in the range of [0.095, 0.165]m. The mass of each object is randomly sampled from [0.03, 0.18]kg. The mass of the objects used in the real-world tests is shown in Table A.2.

Convex Decomposition We use approximate convex decomposition (V-HACD [167]) to perform an approximate convex decomposition on the object and the robot hand meshes for fast collision detection in the simulator. The decomposition resolution is 100,000.

Training setup

Observation space for the teacher policy The inputs to the teacher policy, $\mathbf{o}_t^{\mathcal{E}} \in \mathbb{R}^{19G+21}$, include joint positions (\mathbb{R}^{3G}) and velocities (\mathbb{R}^{3G}), fingertip pose (\mathbb{R}^{7G}) and velocities (\mathbb{R}^{6G}), object pose (\mathbb{R}^7) and velocity (\mathbb{R}^6), target orientation expressed as a quaternion (\mathbb{R}^4), and the rotation difference between current and target object orientation expressed as a quaternion (\mathbb{R}^4), where G represents the number of fingers. The pose of each finger is represented by a position (\mathbb{R}^3) and an orientation component (quaternion; \mathbb{R}^4).

Teacher policy architecture Our teacher policy is an MLP (Multilayer Perceptron) network consisting of three hidden layers (512, 256, 256 neurons) and ELU (Exponential Linear Unit) activation functions [168]. We use Adam [169] to optimize the networks.

GPU hardware In our experiments, we use one NVIDIA GeForce RTX 3090 for training the teacher policies, and one NVIDIA Tesla V100 for training the student (vision) policies.

Hyper-parameters Table A.1 lists the hyper-parameters used in the experiments.

Point cloud voxelization The point cloud input to the policy network has no color information and is voxelized in a resolution of 0.005m.

Success criteria The success criterion defines when the agent has accurately reoriented the object in the target configuration. Its purpose is two-fold: a reward signal during training and a criterion that signals success to stop the reorientation policy and thereby end the episode during training. A straightforward success criterion is judging whether an object’s orientation is close to the target orientation (orientation criterion). The controller learned using the criterion when evaluated in simulation results in a behavior wherein the robotic fingers stabilize the object when its orientation is close to the target. However, the same controller, when evaluated in the real world, often does not result in fingers stopping when the object orientation is close to the target leading to overshooting. Consequently, instead of stopping, the object oscillates around the target orientation. We believe this is a result of sim-to-real gaps, including the control latency, observation noise, and the difference in dynamics. We ameliorate this issue by expanding the definition of success criterion to penalize finger and object motions explicitly. The task is considered completed successfully in the simulation if all the following three criteria are satisfied. First, the orientation criterion is satisfied when $|\Delta\theta_t| < \bar{\theta}$ where $\Delta\theta_t$ is the distance between the object’s current and target orientation. Second, the finger motion criterion requires the joint motion of the robot to be small and is satisfied when $\|\dot{\mathbf{q}}_t\| < \bar{q}$ and $\|\mathbf{a}_t\| < \bar{a}$ where $\dot{\mathbf{q}}_t$ is the joint velocities at time step t , \mathbf{a}_t is the policy output. Third, the object motion criterion requires the object’s velocity to be small. It is satisfied when $\|\mathbf{v}_t^o\| < \bar{v}$ and $\|\boldsymbol{\omega}_t^o\| < \bar{\omega}$ where $\mathbf{v}_t^o, \boldsymbol{\omega}_t^o$ denote object’s linear and angular velocities respectively.

$\bar{\theta}, \bar{q}, \bar{a}, \bar{v}, \bar{\omega}$ are manually defined thresholds. The finger and object motion criteria act as regularizers to explicitly encourage the policies to slow down the motion near the end.

Compute cost Energy Cost: We used a single NVIDIA V100 (32GB memory) GPU to train the policy. The total training time, including the teacher and two student stages, was less than 400 hours. The GPU has a maximum power consumption of 250W when running at full capacity, but our learning system did not always use the GPU to its fullest extent. Therefore, the maximum power consumption was not always reached, resulting in a GPU power consumption of no more than 100kWh. Based on the average electricity cost in the United States of 15.64 cents/kWh in May 2022, the total cost of GPU computing is less than \$15.6. Additionally, if we consider the energy cost of other workstation components, such as the CPU and fans, the power rate remains under 1000W. Running the entire system at full utilization for 400 hours would cost around \$62.4.

GPU Cost: The V100 GPU model designed for HPC (High-performance Computing) is relatively expensive, costing about \$4K on Amazon. We used it only because it is the default GPU model in our servers. However, our training is not limited to V100 and can also be performed on other GPUs. For example, the training can be done using a 3090 RTX GPU, which costs about \$1.4K. During deployment, we have also successfully run the policy on a workstation with a 2080Ti GPU, which costs only \$700. Therefore, the cost of computing hardware can vary greatly depending on the available GPUs. Our policy network is not very large, so it can be trained or deployed on cheaper GPUs, such as the 3090 RTX.

Real-world setup

We used two robot hands in our experiments: a three-fingered hand (D’Claw) and a four-fingered hand. The three and four-fingered hands consist of nine and twelve Dynamixel motors, respectively. The hands are fixed on an 80/20 aluminum frame. Since our goal is to construct a real-world ready reorientation system that can be used on a mobile manipulator in the future, we only use one RealSense D415 camera to observe the robotic hand manipulate the object. The robot and camera are calibrated using the dual quaternions method [170]. We only perform camera calibration on one of the robot fingers, and the ArUco marker is attached to the fingertip. Due to the limits on the finger’s motion (3 DoFs), it is impossible to span a broad range of positions for the ArUco marker, resulting in noisy calibration with noticeable errors. Empirically we found such errors didn’t influence the performance of our system. We use ROS [171] for communication with the Dynamixel motors and use a threading lock to prevent simultaneous reading and writing on the motors as Dynamixel motors use a half-duplex UART (Universal Asynchronous Receiver Transmitter) for communication. Each Dynamixel motor is controlled via position control in 12Hz. The Dynamixel-specific parameters noted in Dynamixel’s control table are set as follows: P is 200, and the D gain is 10. The observations available to the controller are the joint positions of each motor and the depth image from the Realsense camera.

Real-world observation The observation consists of the joint positions of each motor in the robotic hand and the depth image from the RealSense camera. We convert both these sensory inputs into a point cloud. The joint angles are converted into a point cloud as follows: Using the robot’s CAD model, we uniformly sample points on each link and cache it. Given a sequence of joint positions, we use forward kinematics to compute the pose of each link and accordingly transform each of the associated pre-cached point clouds. We call the

concatenated point cloud of all links as proprioceptive point cloud. The depth image is also converted into a 3D point cloud (exteroceptive point cloud). Both point clouds are merged and used as inputs to our policy. We do not assume access to any other sensory information.

Stopping criteria The robot stops when it is deemed successful as per the success criteria described above. Evaluating the success criteria requires knowledge of object motion (object motion criterion), orientation distance (orientation criterion), and fingertip motion (finger motion criterion). Although we can measure the fingertip motion ($\dot{\mathbf{q}}_t$ and \mathbf{a}_t) in the real world, we cannot directly measure the object’s motion and pose. To obtain the orientation error, we train a predictor that re-uses features from the policy network to predict $|\Delta\theta_t|$ (see Figure 7A). Predicting object motion from point clouds is harder, and we found it unnecessary to estimate. We found that satisfying only the orientation criterion and finger motion criterion is sufficient to stop the object at the target orientation successfully. We also found it sufficient only to check $\|\mathbf{a}_t\| < \bar{a}$ to detect finger motion during real-world deployment.

Real-world quantitative evaluation setup We set up a motion capture system using six OptiTrack cameras to evaluate the policy performance in the real world quantitatively. We add markers on the surface of evaluation objects. Even though the added markers add little bumps on the object’s surface and therefore change the dynamics, we found our policies to be robust enough to deal with these changes. The output of the motion capture system is the object pose. We use the tracked object pose when the stopping criteria are satisfied to compute the error from the target orientation. We only use the motion capture system for quantitative evaluation, and it is not required by our system to reorient objects. Note that the motion capture system can occasionally fail to track the objects when the fingers heavily occluded the markers. When it happens, we discard this test as we cannot get a quantitative error in this case.

Computing the orientation error for symmetric objects

With symmetric objects, it is hard to determine whether the controller completes the task or not unless we know in what way the object is symmetric. We can still use a motion capture system to get the object’s pose. However, we cannot directly compute the distance between the orientation from the motion capture system, and the goal orientation as there exist many different orientations that make objects look similar. Therefore, we need to find out all such possible goal orientations. Although scaling this up to a wide variety of objects is challenging, in our experiments, we tested on two symmetric objects (a rectangular cuboid and a cube) whose symmetric axes can be easily enumerated. In other words, on the rectangular cuboid and cube, we can easily identify all possible rotational axes upon which the objects can be rotated by some angle and end up with the same visual appearance. Then we compute the distances between the actual object orientation and all possible goal orientations and find the minimum distance.

A.1.3 Fabrication

Fingertip Fabrication We experimented with both rigid and soft fingertips. The rigid fingertips were fabricated using 3D printing. For the soft fingertips, we designed a rigid inner skeleton coated in a soft outer elastomer. The elastomer allows the robot’s fingertips to have increased compliance and friction when interacting with the plastic objects, as the plastic internal skeleton helps maintain the shape of the fingertips without too much deformation, similar to a human finger. The design for the internal skeleton used is shown in Figure A.2.

The fingertip skeletons and molds for the elastomer are 3D printed on the Markforge Onyx One using the Onyx filament. To help improve the adhesion of the silicone to the Onyx material, the skeletons are sanded with 400-grit sandpaper, corona treated, and primed using the Dow DOWSIL P5200 Adhesion Promoter.

The gel coating for both fingertip designs is made from Smooth-On’s Ecoflex platinum-catalyzed silicone. The elastomer has a shore hardness of 00-10 and exhibits a tacky finish when fully cured. A ratio of 1:0.008:0.005 by weight of the Ecoflex mixture (Parts A and B combined), Smooth-On Silc Pig White, and Smooth-On Silc Pig Black are combined to provide the gray color of the fingertips. To ensure the surface of the gel elastomer is smooth, XTC-3D is applied to the inside of the 3D-printed molds to smooth out any texture. The uncured Ecoflex mixture is poured into the molds and degassed to eliminate air bubbles from forming on the elastomer’s surface. The skeletons are pushed into the top-down molds and left to cure at room temperature for 4 hours.

Object Fabrication We use 3D printing to fabricate objects that are used for quantitative evaluation. Each object was printed with a 0.25mm layer height on the Lulzbot TAZ Pro Dual Extruder using PolyTerra PLA filament in different colors. Table A.2 lists the mass of each object used in real-world experiments. Note that to test the transfer of our results, we also included real household objects in our test set.

A.1.4 Overcoming sim-to-real gap

Dynamics Identification

Action commands In this work, we send two types of action commands to the robot and collect the joint movement trajectories. The first type of action command is a step command ($\mathbf{A}(t) = c$ where c is a constant joint position command). We collect the step responses from the motors. The second type of action command is sin-wave action commands in different frequencies. The sin-wave command is $\mathbf{A}(t) = \sin(2\pi ft)$ where $f \in [0.05, 1.5]$ Hz. For both types of action commands, we scale the amplitude proportionally to the joint limit of each joint.

Dynamics parameters to be identified We perform dynamics identification on the joint stiffness, damping, and velocity limit. Only one finger on the real robot hand is used to collect the response trajectories. Each joint on the finger is identified individually, and the same group of dynamics parameters of the finger is applied to the remaining fingers in the

simulator. Figure A.3, Figure A.4, and Figure A.5 show that the simulated joints behave similarly to the real joints given the same control commands.

Response curves In Figure A.3, Figure A.4, and Figure A.5, we show the response curves of the three joints on a finger given a sequence of action commands both in simulation and in the real world. We can see that after the dynamics identification, the simulated joints can give a similar response as the real joints. We also observe that the real joints (the orange lines) usually have a slightly slower response than the simulated joints (the green lines). This is due to the latency of a real robot hand system. We did not model the latency in simulation and found that our controller still works on the real robot hands. It is possible that including the latency in simulation might further improve the controller’s real-world performance, for which we leave the investigation to future work.

Robust policy learning

Observation and action noise We add Gaussian noise to the action commands (Table A.3) for training all policies. The teacher policy is trained with state noise as detailed in Table A.3. The vision policies are trained with data augmentation on the point cloud observation. With a probability of $p = 0.4$, we add Gaussian noise $\mathcal{N}(0, 0.004)$ to the point positions. Independently, with a probability of $p = 0.4$, we randomly drop out $q \in [0, 20]$ percent of the points.

Dynamics randomization We train policies with small randomization in the joint dynamic parameters: link mass, joint friction, and joint damping. We add large randomization to the object dynamics parameters such as mass friction, and restitution. Table A.3 lists the amount of randomization we add to the dynamics parameters and the observation/action noise.

Disturbance force on the object With a probability $p = 0.2$ at each time step, we apply a disturbance force with a magnitude of $F_d^o = c_d m^o$ where m^o is the object mass, c_d is a coefficient and a random force direction sampled in the $SO(3)$ space.

A.2 Supplementary Discussion

A.2.1 Student policy closely tracks the performance of teacher policy

We evaluated our learned policies on the training object dataset (\mathbb{B}) and the testing object dataset (\mathbb{S}), respectively, in simulation. To characterize how well a policy behaves in the testing time, we use the empirical cumulative distribution function (ECDF) as the metric to measure the distribution of the errors ($\Delta\theta$). Figure 7D shows the ECDF curves for the teacher policy and student policies at stages 1 and 2, respectively. Using fully-observable low-dimensional state information, the teacher policy achieves the highest success rate at any error threshold. The student policies are able to track the teacher policy’s performance closely.

A.2.2 Symmetric object reorientation

Learning visual policies to reorient symmetric objects is challenging because objects in different but symmetrical poses appear similar leading to multimodality[18]: Given a target orientation, many different poses of a symmetric object match the goal configuration visually, leading to different but equally good action sequences. It is challenging to learn a stochastic vision policy that explicitly accounts for multi-modality. An alternative is to use implicit models [172]. However, these models are computationally inefficient, which prevents their use in a real-time controller.

The key intuition behind how we overcome this problem is that we need to account for multi-modality only at training time to ensure that a correct action sequence is not incorrectly penalized. However, at deployment, it is not necessary to distinguish between modes, and reorienting the object into any of the equivalent symmetric configurations would suffice. We bypass the problem of accounting for multi-modality at training time by using only unimodal objects – asymmetric or reflective-symmetric objects. Our hypothesis was that if we are able to successfully learn a controller that operates over a diverse range of asymmetric shapes, it may also generalize to symmetric objects.

To verify if this hypothesis was true, we tested our controller on two symmetric objects (a rectangular cuboid and a cube) with table support. Figure 4F shows that our controller still works reasonably well. Nonetheless, in this case, the orientation error tends to be higher than non-symmetric objects (Figure 4E). Although it is hard to pinpoint whether this is due to the performance drop in predicting the actions or predicting when the goal orientation is reached, we believe the latter plays a bigger role.

A.2.3 Ablation on the reward terms

To investigate how reward terms and their coefficients affect policy performance, we conducted an ablation study where we varied the values of c_1 , c_2 , and c_3 in Equation 2.1. For the sake of brevity, we trained policies on a single object (object #10) without domain randomization.

As illustrated in Figure A.6, increasing the value of c_1 led to an improvement in policy learning, but too large a value of c_1 resulted in a decline in performance. Similarly, increasing c_2 improved policy learning, but after $c_2 = 1.0$, the performance began to deteriorate. In contrast, policy learning was found to be less sensitive to c_3 , which encourages fingers to remain near objects, and to c_5 , which discourages fingers from pushing objects away. However, we did observe that having $c_3 < 0$ and $c_5 < 0$ was advantageous, as the learning curves exhibited substantially higher variance when $c_3 = 0$ and $c_5 = 0$. The fourth term (c_4) imposes an energy penalty to prevent excessive energy usage on the motors. As shown in Figure A.6, a c_4 value of 0 allows the policy to learn the fastest, as there are no energy constraints. However, as c_4 increases, the learning speed decreases. When c_4 becomes too large, the learning process begins to fail.

A contact penalty term (c_6 , Equation 2.7) promotes in-air object reorientation by reducing table dependency. Policies trained with the penalty achieved 87% in-air success, as those without achieved only 4.1%. This clearly shows that the contact penalty term benefits the in-air reorientation.

A.2.4 Using a different encoder for goal

As shown in Figure A.7b, we also found that stacking the goal object point cloud onto the scene point cloud and feeding it as a whole into the 3D CNN encoder (Figure 7B) leads to faster policy learning than using two separate 3D CNN encoders (Figure A.7a) to process the scene point cloud and the goal object point cloud.

A.2.5 Stage 0: speeding up vision policy training with visual pre-training

Can we further speed up the vision policy learning? As shown in Figure 7A, our policy network is a recurrent network. Training a sequence model can take longer training time. We investigated whether we can first pre-train the vision network component (sparse 3D CNN) in the policy network without involving the RNN component (Stage 0), and then fine-tune the policy with the pre-trained vision network (Stage 1). To pre-train the vision network, we explored various representation learning techniques, such as learning a forward/inverse dynamics model and reconstructing the input point cloud to pre-train the vision network. To our surprise, although most pre-training techniques lead to mild, if any, improvement in the policy learning speed, it’s beneficial to first train the vision network to predict some low-dimensional state information of the system. More specifically, in the pre-training stage, the vision network is trained to predict the object category, the distance between the object’s orientation and the goal orientation, and the joint positions of the robot hand (\mathbf{q}_t). Note that we do not predict object pose, which would require a definition of reference frame on the objects. To further speed up the training, we do not use simulation at all to generate the training data at this stage. Instead, we generate completely random synthetic data for training (Figure A.8). First, we convert meshes of the robot links and objects into their corresponding canonical point clouds. Next, we randomly sample joint angles and use forward kinematics to get the pose of each robot link and randomly sample object poses and goal poses. Finally, we transform the canonical point clouds of each part according to their poses and get the point cloud of the entire scene.

For rotational distance prediction, we experimented with two representations. The first one is the 6D representation [173] of the relative rotation matrix $\mathbf{R}_t^o(\mathbf{R}^g)^{-1}$ between the object’s current orientation \mathbf{R}_t^o and goal orientation \mathbf{R}^g . The second one is the scalar distance between the two rotation matrices ($\Delta\theta_t$). We found that pre-training the vision network to predict the scalar distance leads to faster convergence during pre-training than predicting the 6D representation of the relative rotation matrix. In addition, we experimented with two ways of using the output of the vision network for the state prediction tasks: (1) three prediction tasks use the same embedding (Figure A.9a), (2) we split the vision network output into three parts (object embedding \mathbf{z}^o , goal embedding \mathbf{z}^g , robot embedding \mathbf{z}^r), and each prediction task only uses the relevant embedding (Figure A.9b). When training to predict the scalar rotational distance, we don’t find these two ways of using the vision network output to make a difference. However, when training to predict the 6D representation of the rotational distance, splitting the embedding leads to a notably faster pre-training (Figure A.10a). After the pre-training converges, we proceed to Stage 1 using the pre-trained vision network backbones. We found that all four pre-training schemes lead to a substantial and similar speedup for

policy learning (Figure A.10b). We also did an ablation study on the importance of different pre-training tasks in Section A.2.6.

A.2.6 Ablation on the prediction tasks for vision network pre-training

We perform ablation study on different prediction tasks for pre-training the vision networks in Stage 0. In Figure A.11, Full represents the case of training to predict all three tasks (the scalar rotational distance, the joint positions, and the object category) with the single embedding architecture. And our ablation studies remove each prediction task individually. As shown in Figure A.11, removing the rotational distance prediction task makes the pretraining much easier (loss goes down much faster), but it also negatively affects the benefit of pre-training the most. It implies that the task of predicting the rotational distance is most useful for pre-training the vision network. Removing the task of predicting the joint positions slightly reduces the benefit of pre-training. Removing the task of predicting the object category has almost no effect on the policy learning.

A.2.7 Analysis for object reorientation in the air in simulation

We can get three outcomes when using the four-fingered hand to reorient objects in the air: the episode succeeds (Success), the controller stops the object with an orientation error bigger than 0.4 radians (Orientation error), the object falls (Object falls), or the controller runs out of time and fails to reach the goal orientation (Time out). To see the ratio of each case, we tested the vision controller, which takes as input the realistically rendered point cloud, on the twelve objects in Figure A.12a in simulation. We set the testing episode length to 180 time steps, which is equivalent to 15 seconds. For the first 24 time steps (2 seconds), we keep the table below the hand so that the hand can first grasp the objects. After the 24th step, we remove the table and check if the controller can reorient the objects in the air. Note that this is a rough approximation of the real-world testing scenario where we hand over the object to the robot hand and release the object after the hand grasps the object, because when we remove the table, there is no guarantee that the hand happens to grasp the object stably at the 24th time step. Nonetheless, empirically, we found only 10.3% of the 1200 testing episodes have the issue of object falling immediately (we check if the object falls from 24 steps to 30 steps) after the table is removed, suggesting that this is still a reasonable approximation of the real-world testing scenarios. To emulate how we stop the policy in the real world, we also stop the policy in the simulation if $|\Delta\theta_t^{pred}| < \bar{\theta}$ (we check the predicted orientation distance from the policy network) and $\|\mathbf{a}_t\| < \bar{a}$.

Each object is tested 100 times with a random initial pose and goal orientation. We plot the percentage of each case (Success, Orientation error, Time out, Object falls) in Figure A.12c. The figure shows that the majority of failures occur because objects fall out of the hand. For object #12, the percentage of failures due to large orientation errors is particularly high because this object is nearly symmetric in the point cloud representation. Figure A.13a and Figure A.13b show the comparison of two objects between the orientation error and episode time in simulation and in the real world, respectively. The results suggest that there are still gaps in the policy performance between simulation and the real world. Nonetheless, even in

the real world, the median time for successful reorientation in the full $SO(3)$ space is less than 7 seconds, demonstrating the fast and dynamic manipulation capability of the system. Figure A.13c and Figure A.13d show the distribution of the orientation error and episode time of the non-dropping episodes for all twelve objects in the simulation.

A.2.8 Discussion on precise manipulation

In this study, we adopted a success threshold of 0.4 radians, consistent with the definition used in a previous study [15]. It is natural to wonder if our controller can accurately reorient objects with a smaller reorientation error. To provide more insight into the system performance at a stricter success criterion of 0.1 radians, we did more analysis in simulation. We find that at 0.4 radians, the success rate is 72.3% (from 1200 tests on the 12 objects shown in Figure A.12a), but drops to 25.9% at 0.1 radians. However, this drop is not due to the inability of our controller to perform precise manipulation. It is, in fact, largely attributed to the failure of the module that predicts the rotational distance between the object’s current and target orientation, which in turn is used to stop the hand. For instance, if we use the ground-truth distance to stop the controller, success rates of 67.4% and 80.9% are achieved at 0.1 radians and 0.4 radians thresholds, respectively.

To provide readers with a better understanding of the accuracy of the rotation distance predictor, we have also included scatter plots comparing ground-truth and predicted distance in Figure A.14. More specifically, we conducted simulation tests on each of the twelve objects 100 times, with a success threshold $\bar{\theta}$ set to 0.1 radians. For each trial, we recorded the trajectory and predicted rotational distance at each time step. We then plotted the actual and predicted rotational distances between the object and the goal orientation in Figure A.14. The figure demonstrates that although the prediction model performs reasonably well overall, it suffers from providing sufficient accuracy in the region where $\Delta\theta \leq 0.4$ radians, which is of particular interest for precise manipulation. For example, among the data points for which the actual $\Delta\theta \leq 0.1$ radians, only 29.4% of the predictions correctly estimated distances within 0.1 radians. When the actual $\Delta\theta \leq 0.4$ radians, 85.9% of the predictions estimated the distance to be less than 0.4 radians. Inaccurate predictions of rotational distance, coupled with observation and command delays in the real system, make precise manipulation with $\Delta\theta \leq 0.1$ radians challenging. This indicates one research direction for improving orientation accuracy is to train a better predictor for orientation distance or a better classifier for identifying whether the goal orientation has been reached.

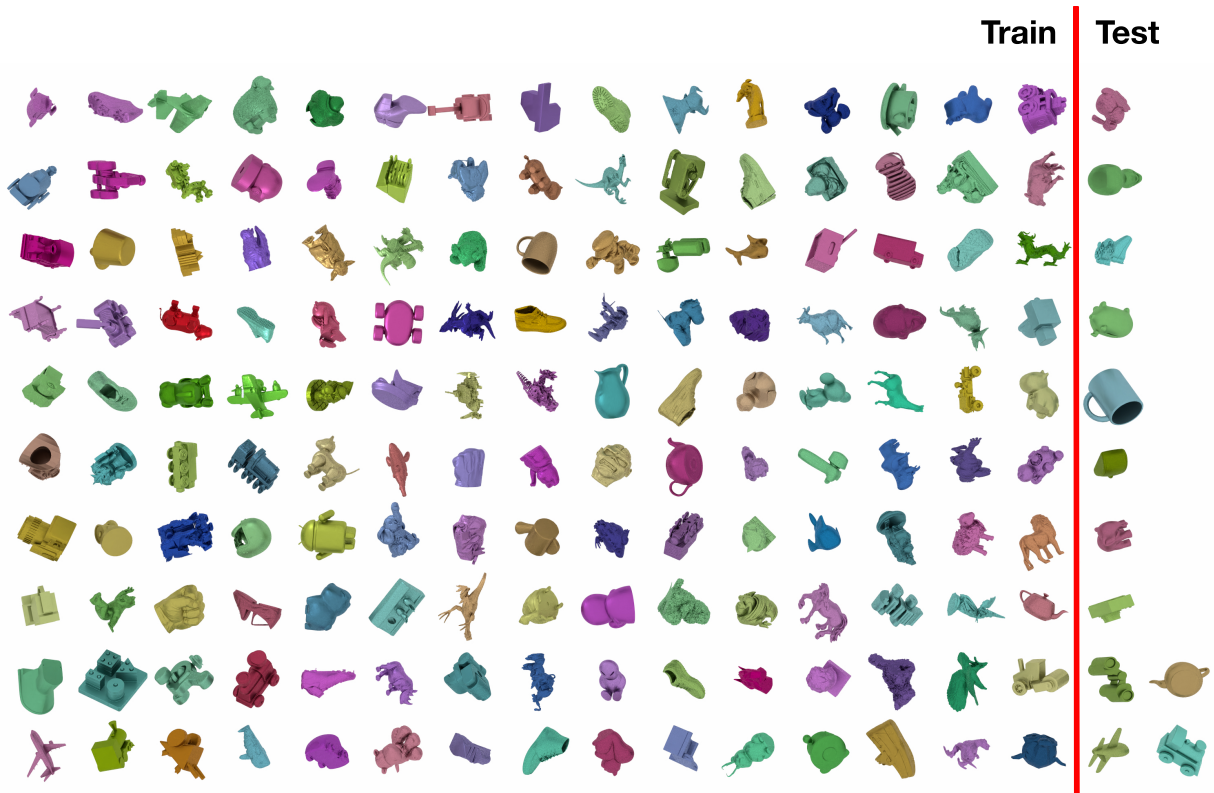


Figure A.1: **Object dataset.** On the left of the red line, we show the dataset \mathbb{B} (the training dataset). And on the right of the red line, we show the dataset \mathbb{S} (the testing dataset in simulation).

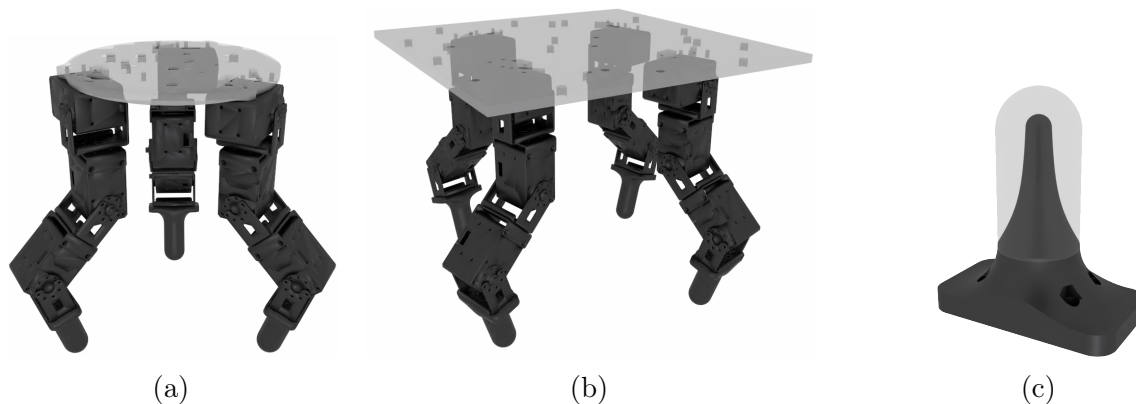


Figure A.2: **3D models for the robot hands.** (A): three-fingered robot hand. (B): four-fingered robot hand. (C): fingertips with a rounded skeleton and the grey shell represents soft elastomer.

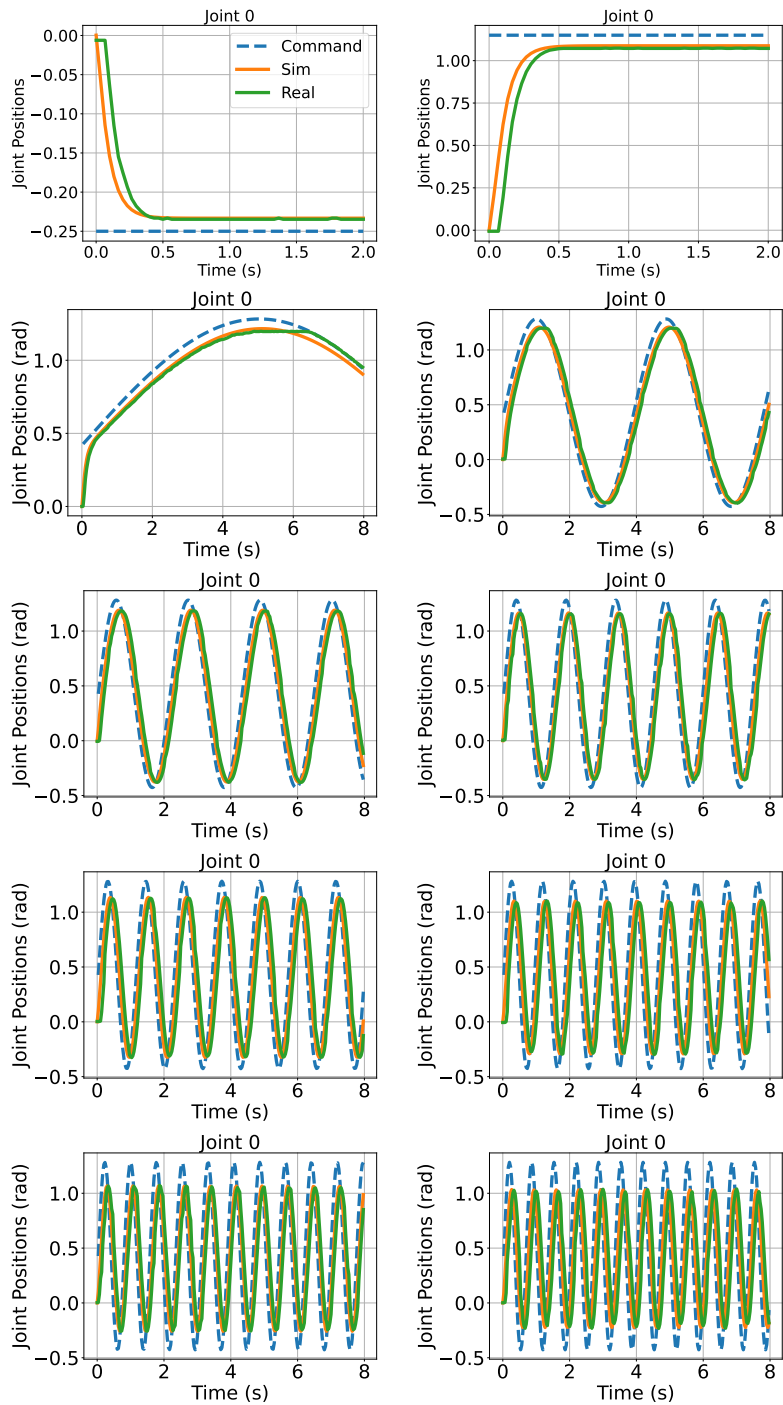


Figure A.3: **Joint response curves.** We identified the dynamics of a finger joint (the top one) and show the results. Three curves are plotted: (1) the command sent to the joint, (2) the joint's simulated response using the identified dynamics parameters, (3) the joint's real-world response. The identified dynamics parameters allow the simulated joint to move similarly to the real joint.

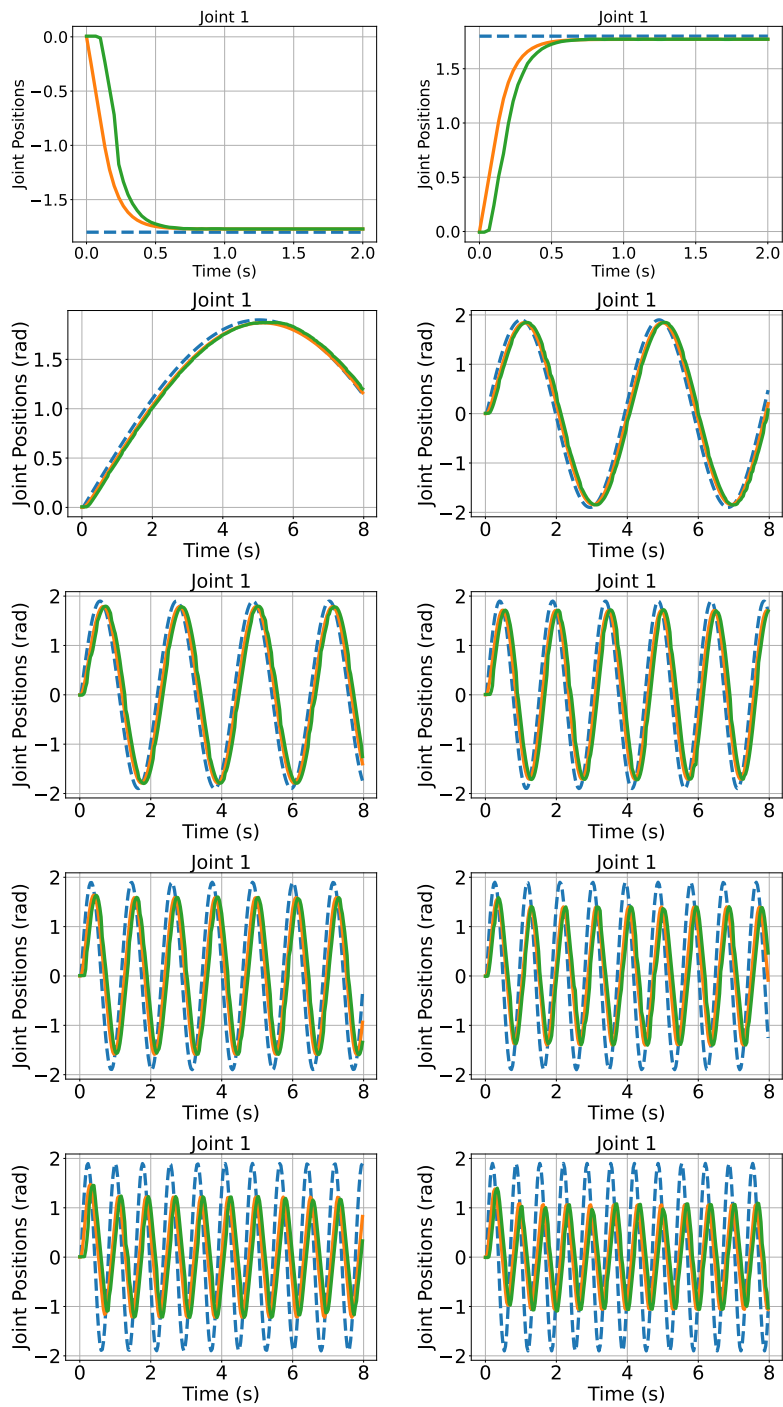


Figure A.4: **Joint response curves.** Dynamics identification on a middle joint of one finger.

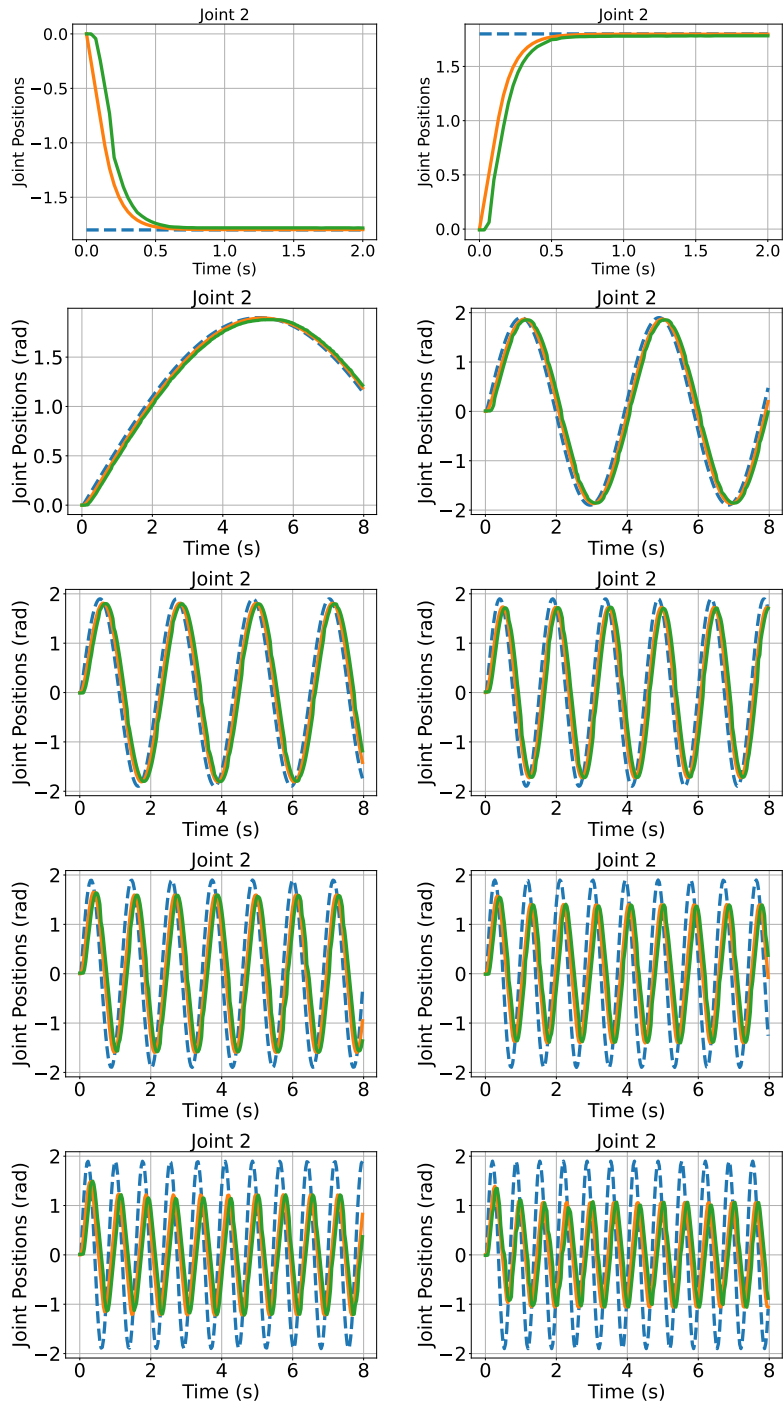
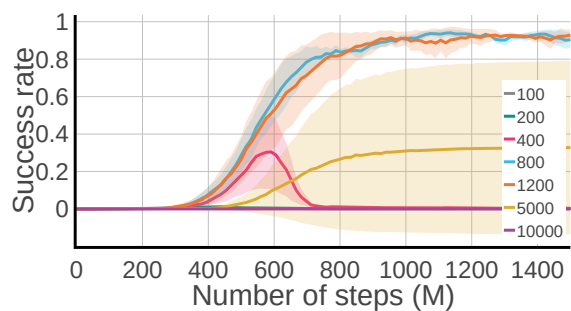
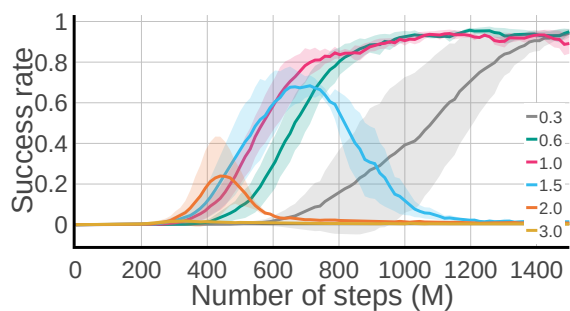


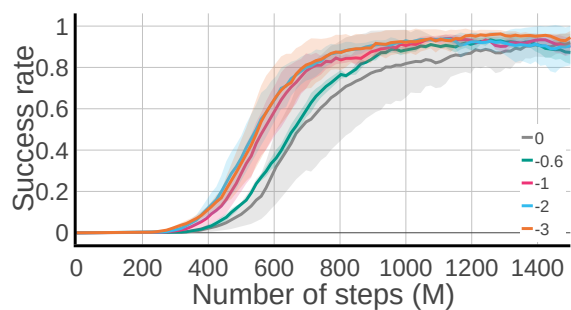
Figure A.5: **Joint response curves.** Dynamics identification on a bottom joint of one finger.



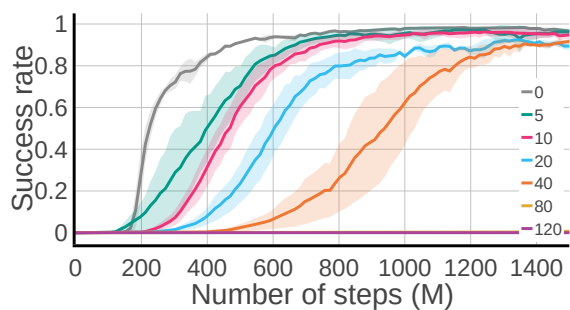
(a) c_1



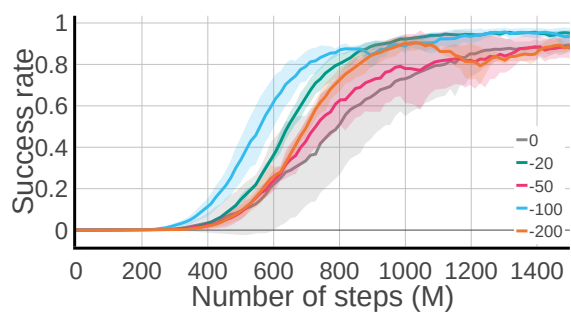
(b) c_2



(c) c_3



(d) c_4



(e) c_5

Figure A.6: **Reward function ablation.** Different learning curves as we vary the values of c_1 (a), c_2 (b), c_3 (c), c_4 (d), c_5 (e).

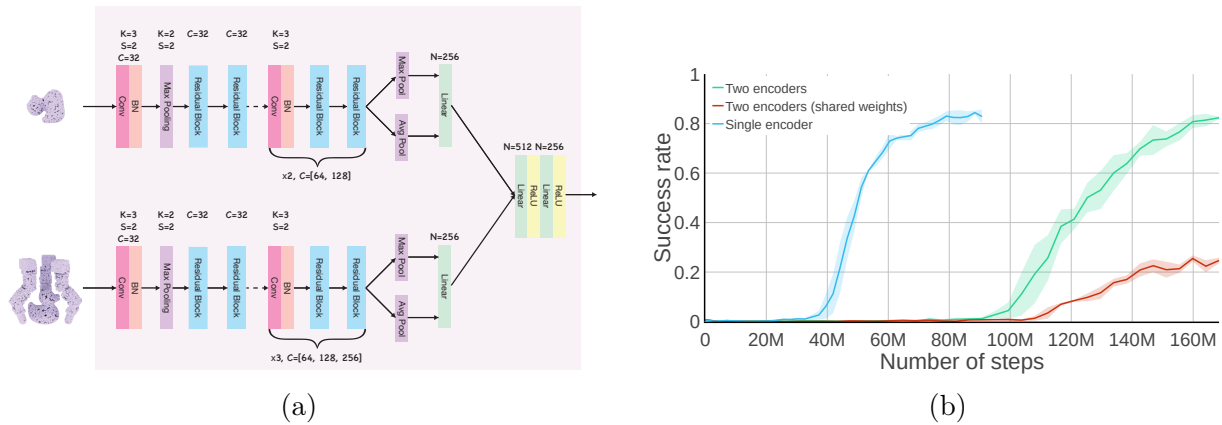


Figure A.7: **Encoder architectures.** (a): we tried using separate encoders for the goal point cloud and the scene point cloud. (b) shows that using separate encoders leads to considerably slower policy learning than using a single encoder on merged goal and scene point clouds.

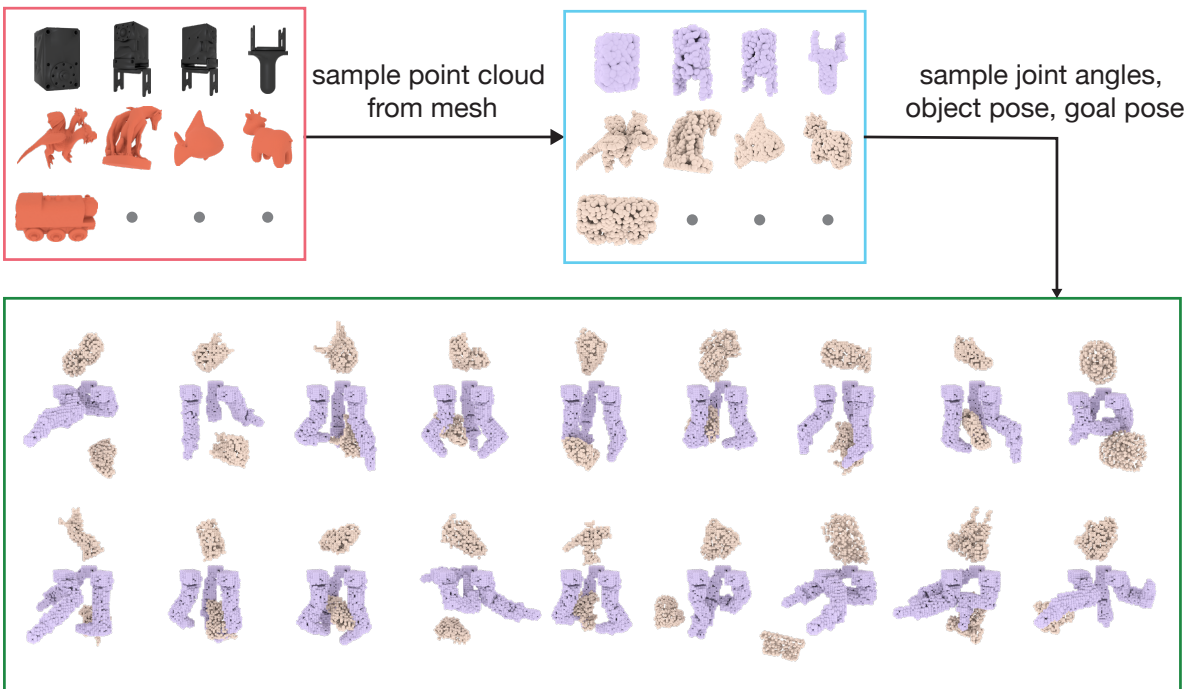
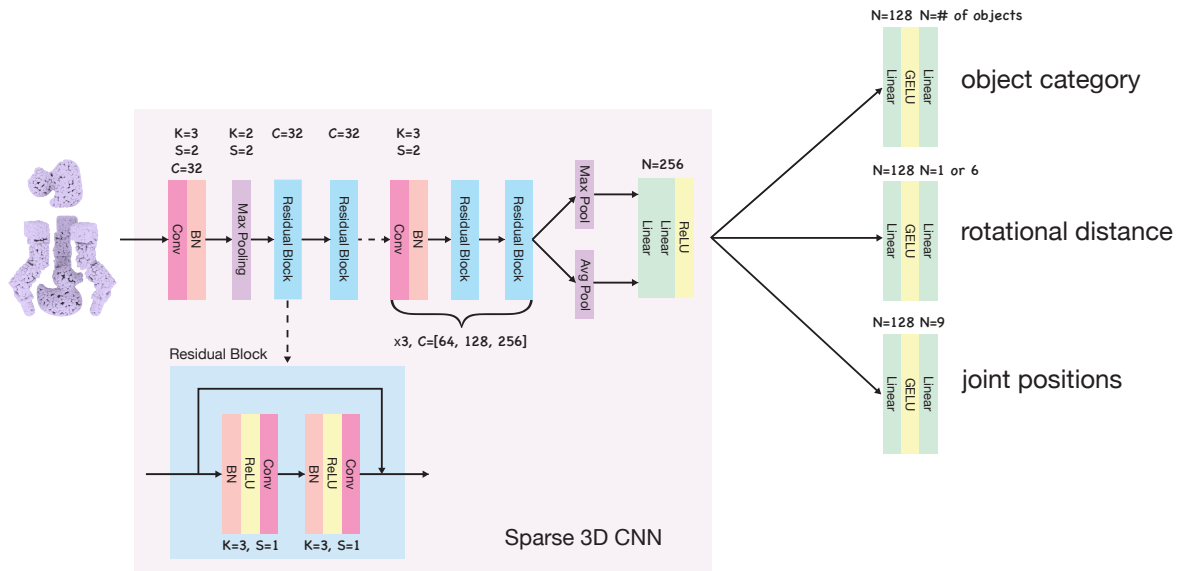
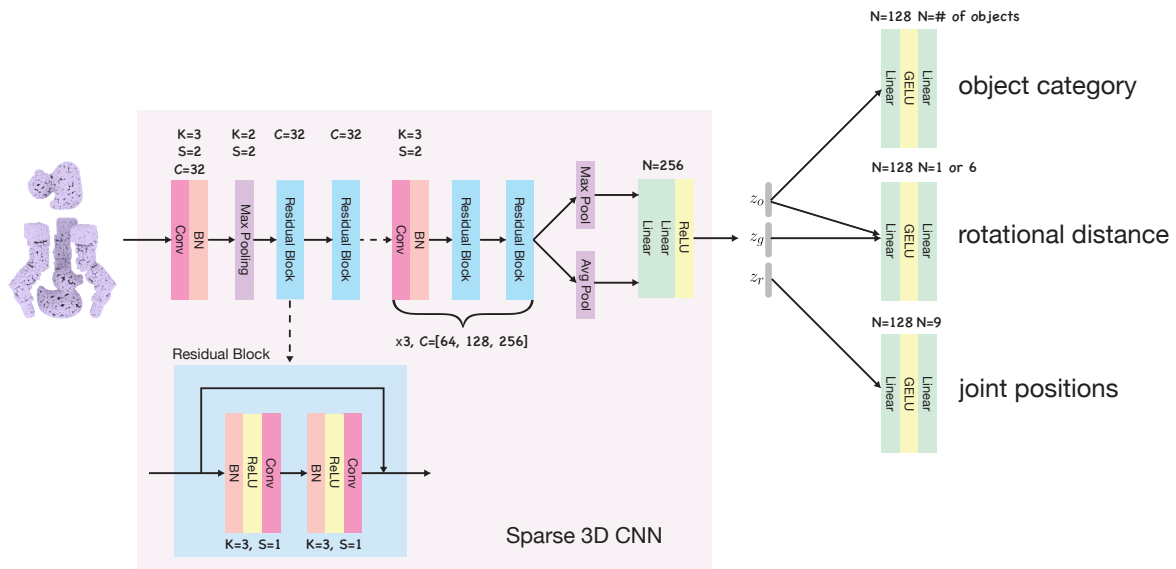


Figure A.8: **Synthetic data.** The synthetic data generation in Stage 0.



(a)



(b)

Figure A.9: **Different architectures for prediction.** In (a) and (b), we designed two architectures, with the difference being whether the output of the vision network is split into entity-specific embeddings or not.

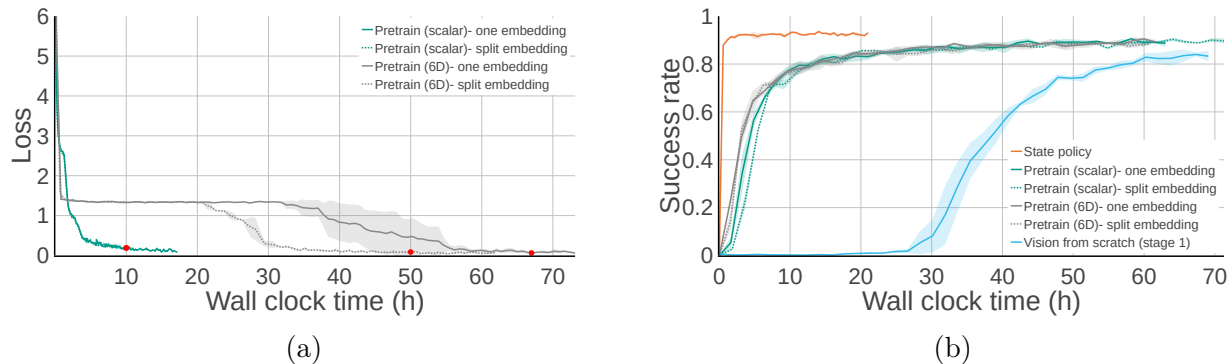


Figure A.10: **Learning curves for pretraining.** (a) shows the learning curves under different training conditions. The red dots represent the checkpoints we took for policy learning. (b): After the pre-training, we use the vision network as the policy backbone and train the policy with BC. The **State policy** is a student policy that takes as input the joint positions, rotation matrix of the relative orientation, and object position, which can be seen as an upper bound for the vision policy. **Vision from scratch (stage 1)** means the vision policy learning in stage 1 only without stage 0.

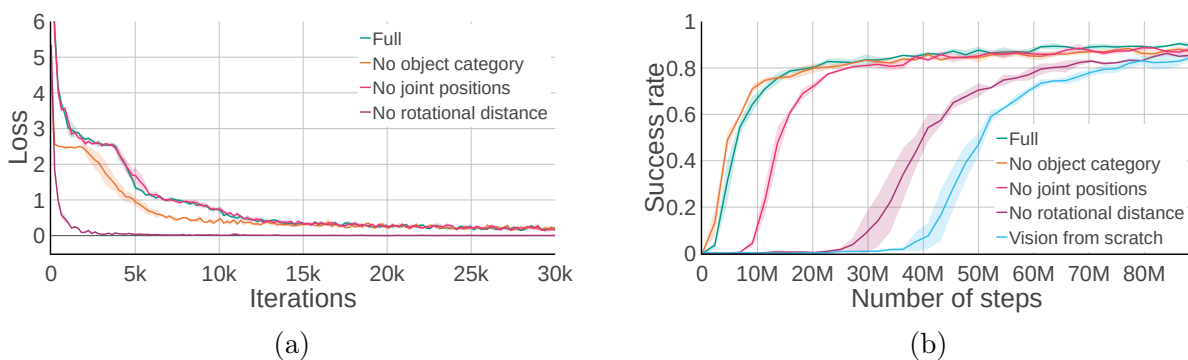


Figure A.11: **Effects of different prediction tasks.** (a): loss curves for the pre-training in Stage 0. (b): learning curves for training the vision policies with the pre-trained vision networks in Stage 1.

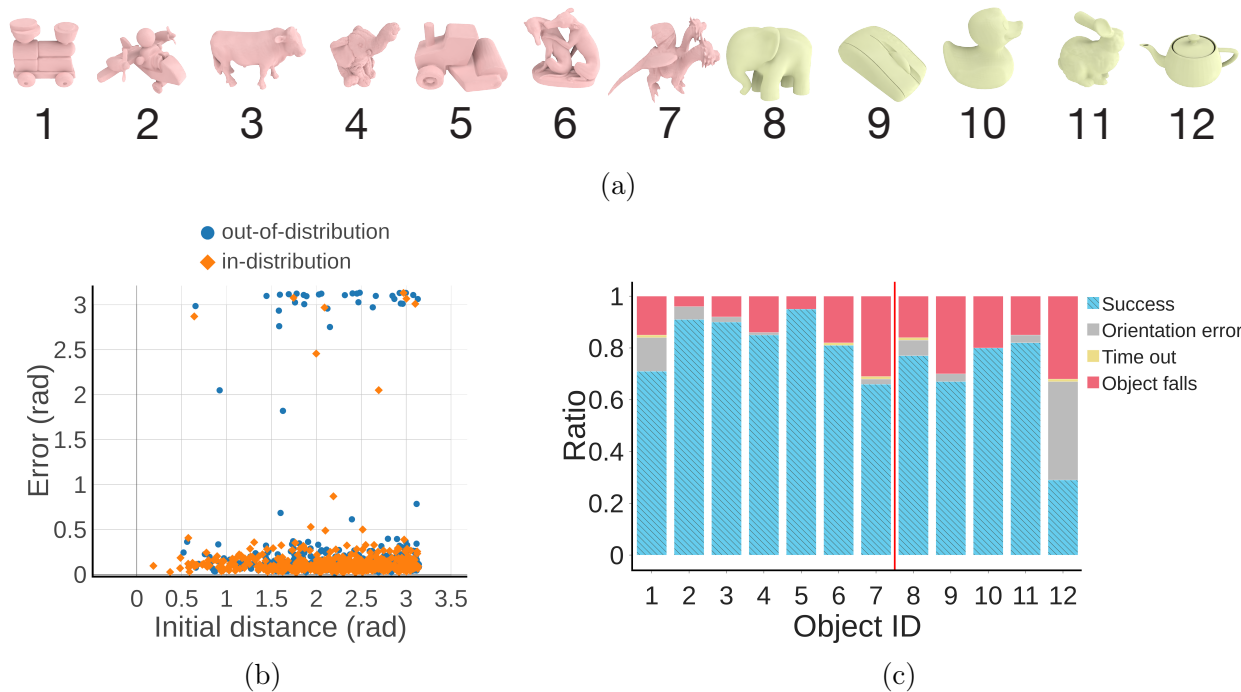


Figure A.12: **Precise manipulation analysis.** Tests for object reorientation in the air in simulation. We tested each object in Figure A.12a 100 times with random initial pose and goal orientation. (a): the objects we used for testing (same as Figure 3A). (b): We show the relationship between the reorientation error and the distance ($\Delta\theta_0$) between the object's initial and target orientation on non-dropping tests (around 90%). We randomly sub-sample the tests on in-distribution objects to make sure the total numbers of points are the same for in-distribution objects and out-of-distribution objects in this plot. (c): We categorize the testing results for each object into four cases: Success, Orientation error (where the controller stops the object with an orientation error greater than 0.4 radians), Time out, Object falls.

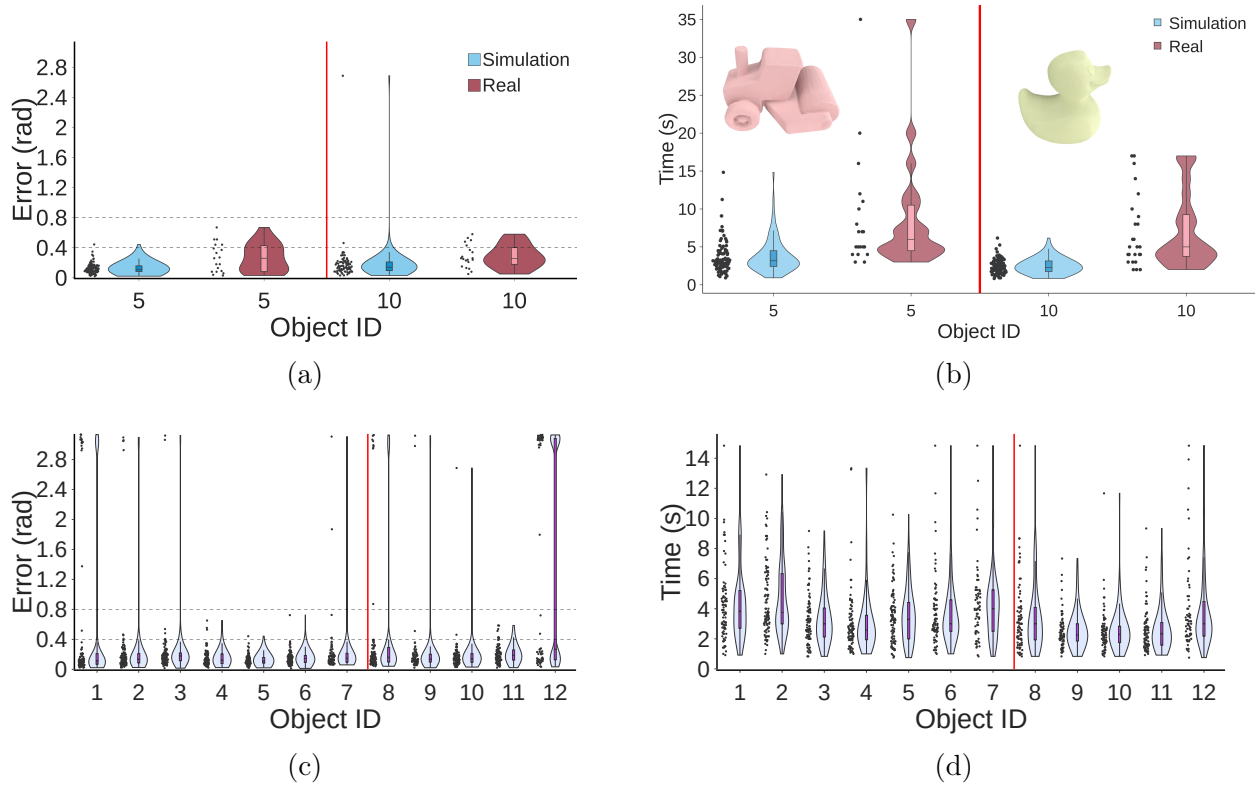


Figure A.13: **Precise manipulation analysis.** Following Figure A.12, (a) and (b): With object #5 and #10, we compare the distribution of the orientation error and the elapsed time of the episodes in simulation and in the real world. The controller achieves lower error and uses shorter time in simulation. We can see there is still a gap between the simulation and real-world performance. (c) and (d): we show the distribution of the reorientation error and episode time of the non-dropping testing episodes on all twelve objects in simulation.

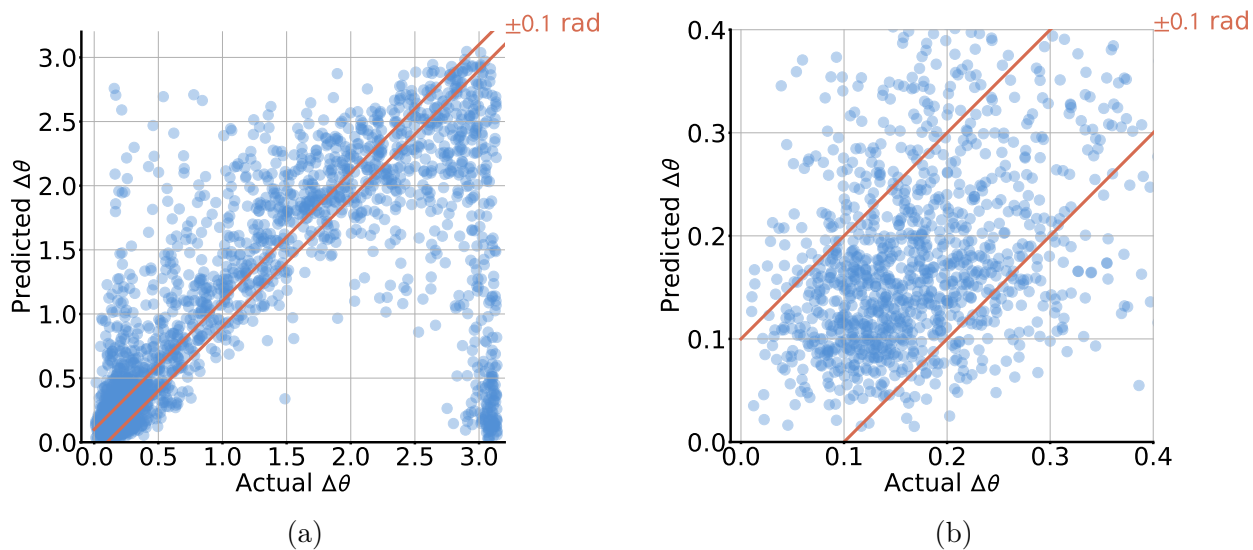


Figure A.14: **Reorientation error analysis.** We plotted the actual and predicted rotational distance between the object and goal orientation from 1200 testing episodes on twelve objects in Figure A.12a. (b) is a zoomed-in version of (a) for $x \in [0, 0.4]$ radians and $y \in [0, 0.4]$ radians. The region between the two red lines indicates an error of less than 0.1 radians. Overall, our rotational distance predictor performs reasonably well, but it has limited accuracy when the actual distance is $\Delta\theta \leq 0.4$ radians, indicating the difficulty of precisely predicting the rotational distance.

Table A.1: Hyper-parameter Setup

Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
Teacher policy					
# of envs	32000	batch size	64000	# of rollout steps per policy update	8
GAE lambda	0.95	Reward discount	0.99	# of policy update epochs after each rollout	12
Actor learning rate	0.0003	Critic learning rate	0.001	PPO clip range	0.1
ϵ_θ	0.4 radians	c_1	800	c_2	1
c_3	-1	c_4	-20	c_5	-100
c_6	-1	c_7	-2	\bar{p}	0.15
\bar{p}_z	0.16	\bar{q}	0.25	\bar{v}	0.04
$\bar{\omega}$	0.5	c_d	15		
Student policy					
# of envs (Stage 1/Stage 2)	400/260	batch size (Stage 1/Stage 2)	40/20	# of pts sampled from each CAD model	500
# of pts sampled from realistically rendered point cloud	6000	learning rate	0.0003	# of rollout steps per policy update	80

Table A.2: Object mass.

























Object	Mass (g)	Object	Mass (g)	Object	Mass (g)	Object	Mass (g)
	158.0		95.0		111.1		116.9
	151.8		70.3		104.3		92.1
	106.3		140.9		86.8		117.6
	148.1		162.1		106.1		50.1
	60.5		104.1		85.7		180.9
	244.0		127.9		137.1		67.1

Table A.3: Dynamics Randomization and Noise

Parameter	Range	Parameter	Range	Parameter	Range
state observation	$+\mathcal{N}(-0.002, 0.002)$	action	$+\mathcal{N}(0, 0.05)$	joint stiffness	$\times\mathcal{U}(0.8, 1.2)$
joint damping	$\times\mathcal{U}(0.8, 1.2)$	link mass	$\times\mathcal{U}(0.8, 1.2)$	friction for robot, objects	$\mathcal{U}(0.24, 1.6)$
friction for table	$\mathcal{U}(0.05, 1.0)$	restitution	$\mathcal{U}(0.0, 1.0)$	object size scale	$\mathcal{U}(0.95, 1.05)$
object mass	$\mathcal{U}(0.009, 0.324)$ kg				

$\mathcal{N}(\mu, \sigma)$: Gaussian distribution with mean μ and standard deviation σ .
 $\mathcal{U}(a, b)$: uniform distribution between a and b .
 +: the sampled value is added to the original value of the variable. \times : the original value is scaled by the sampled value.

Appendix B

Parallel Q-Learning

B.1 Pseudo Code

Algorithm 1 Actor Process (main process)

```
for  $n = 1 : W_a$  do  
   $\pi \leftarrow$  policy network from P-learner process  
  Initialize an empty buffer  $B = \phi$   
  for  $t = 1 : H$  do  
     $\mathbf{a}_t \leftarrow \pi(\mathbf{s}_t)$  with mixed exploration noise  
     $(\mathbf{r}_t, \mathbf{s}_{t+1}) \leftarrow \mathbf{envs.step}(\mathbf{a}_t)$   
     $B = B \cup \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$   
  end for  
   $Q_1, Q_2 \leftarrow$  Q functions from V-learner process  
  send  $B, \pi$  to V-learner, send  $\{\mathbf{s}_t\}$  in  $B, Q_1, Q_2$  to P-learner  
  sleep for  $t_a$  seconds to satisfy  $\beta_{a:v}$   
end for
```

Algorithm 2 P-learner Process

```
Initialize an empty buffer  $B_p = \phi$   
for  $n = 1 : W_p$  do  
  if new data received then  
     $\{\mathbf{s}_t\} \leftarrow$  from Actor process  
     $Q_1, Q_2 \leftarrow$  from Actor process  
     $B = B \cup \{\mathbf{s}_t\}$   
  end if  
  sample a batch of  $\{\mathbf{s}_t\}$   
  update  $\pi$  by maximizing the  $\min_{i=1,2} Q_i(\mathbf{s}_t, \pi(\mathbf{s}_t))$   
  sleep for  $t_p$  seconds to satisfy  $\beta_{p:v}$   
end for
```

Algorithm 3 V-learner Process

Initialize an empty buffer $B_v = \phi$
for $n = 1 : W_v$ **do**
 if new data received **then**
 $\{s_t, a_t, r_t, s_{t+1}\} \leftarrow$ from **Actor** process
 $\pi \leftarrow$ from **Actor** process
 $Q_1, Q_2 \leftarrow$ from **Actor** process
 $B = B \cup \{s_t\}$
 end if
 sample a batch of $\{s_t, a_t, r_t, s_{t+1}\}$
 update Q_1, Q_2 by minimizing the mean-squared Bellman error (with Double Q-learning)
 sleep for t_v seconds to satisfy $\beta_{p:v}, \beta_{a:v}$
end for

B.2 Training setups

B.2.1 Hyper-parameters

We use the hyper-parameter values shown in Table B.1 and the reward scaling shown in Table B.2 for all the experiments unless otherwise specified. As for PPO, we use the same hyperparameter setup in [30].

Table B.1: Hyper-parameter setup for six Isaac Gym benchmark tasks

Hyper-parameter	PQL(ours)	DDPG	SAC
Num. Environments	4,096	4,096	4,096
Critic Learning Rate	5×10^{-4}	5×10^{-4}	5×10^{-4}
Actor Learning Rate	5×10^{-4}	5×10^{-4}	5×10^{-4}
Learnable Entropy Coefficient	-	-	True
Optimizer	Adam	Adam	Adam
Target Update Rate (τ)	5×10^{-2}	5×10^{-2}	5×10^{-2}
Batch Size	8,192	8,192	8,192
Num. Epochs ($\beta_{a:v}$)	8	8	8
Discount Factor(γ)	0.99	0.99	0.99
Normalized Observations	True	True	True
Gradient Clipping	0.5	0.5	0.5
Exploration Policy	Mix	Mix	-
N -step target	3	3	3
Warm-up Steps	32	32	32
Replay Buffer Size	5×10^6	5×10^6	5×10^6

Table B.2: Reward scale

	Reward scale
Ant	0.01
Humanoid	0.01
ANYmal	1.0
Franka Cube Stacking	0.1
Allegro Hand	0.01
Shadow Hand	0.01
Ball Balance	0.1
DClaw Hand	0.01

B.2.2 Hardware Configurations

Table B.3 lists the hardware configurations of the workstations we used for the experiments. We use the machines with GeForce RTX 3090 for experiments by default. We also measure how much time it takes for the simulator to generate 1M interaction data with 4096 parallel environments on *Ant* and *Shadow Hand*. We generate 1M data via the following command.

```
for i in range(244):
    action = torch.randn((4096,
                          envs.action_space.shape[0]),
                          device='cuda')
    out = envs.step(action)
```

Table B.3: Hardware configurations on different workstations

		Workstation 1	Workstation 2	Workstation 3	Workstation 4
CPU		AMD Threadripper 3990X	Intel Xeon Gold 6248	AMD Rome 7742	Intel Xeon W-2195
GPU		GeForce RTX 3090	Tesla V100	Tesla A100	GeForce RTX 2080 Ti
GPU CUDA Cores		10496	5120	6912	4352
GPU FP32 TFLOPs		35.58	16.4	19.5	13.45
Time for generating	Ant	1.678 ± 0.006	2.117 ± 0.038	1.999 ± 0.004	3.397 ± 0.014
1M data ($N = 4096$) (s)	Shadow Hand	6.706 ± 0.028	9.051 ± 0.035	8.653 ± 0.101	10.885 ± 0.025

B.2.3 Vision experiment setup

We render the RGB camera image in a resolution of 48×48 . The CNN part of our vision network $g(o_t)$ is as follows:

Conv(3, 32, 3, 2) -BN(32) -ReLU -3x(Conv(32, 32, 3, 2) -BN(32) -ReLU)

where Conv(a, b, k, s) is a Convolutional layer with input channels a , output channels b , kernel size k , stride s .

Since our policy input contains a history of observations (o_{t-2}, o_{t-1}, o_t) , we use the same CNN to extract the feature of each observation and then concatenate all the embeddings. Then, the concatenated embedding goes through an MLP network h :

FC(256)-ReLU-FC(63)-ReLU-FC(3)

In summary, at each time step t , the policy output is $h[\text{cat}(g(o_{t-2}), g(o_{t-1}), g(o_t))]$. Storing images in a replay buffer can take up a lot of memory. Therefore, we experiment with different placements of the replay buffer: (1) put the replay buffer on a GPU with a big memory, (2) put the replay buffer on CPU RAM. We use the same A100 GPUs for all these image-based experiments. Figure B.1 shows that our method (PQL) works with either the replay buffer on the GPU or CPU, and it achieves much faster learning and better performance than PPO.

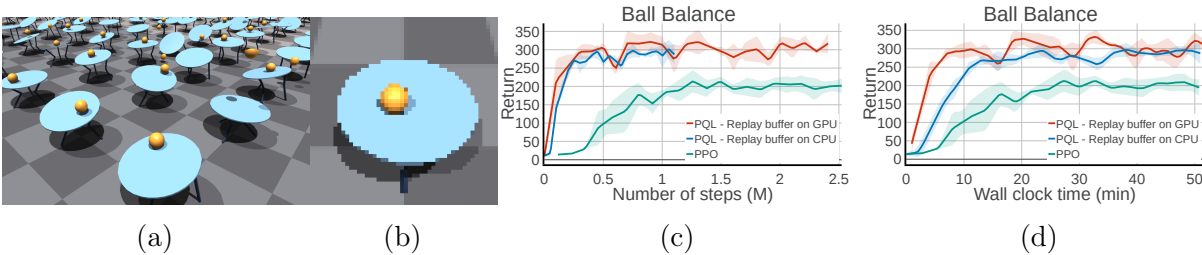


Figure B.1: (a): the *Ball Balancing* task in Isaac Gym. (b): the rendered RGB image from the simulated camera. (c) shows the learning curves regarding the number of environment steps. (d) shows the training wall-clock time. We can see that PQL achieves both better sample efficiency and higher final performance than PPO.

B.3 Additional Experiments

n -step returns We investigate how much does n -step returns help for PQL. As shown in Figure B.3, adding n -step return leads to faster learning than not using n -step return ($n = 1$). However, using a big n value hurt the learning. Empirically we found that $n = 3$ gives us the best performance.

Benefit of adding speed control $(\beta_{p:v}, \beta_{a:v})$ on different processes As we mentioned in Section 4.3.2, adding speed control using $\beta_{p:v}, \beta_{a:v}$ can help reduce the variance of training when the amount of computation resources changes. To provide more insights, we ran experiments without speed control, i.e., each process could run as fast as possible without any waiting. As shown in Figure B.2, when there are sufficient compute resources available (with two GPUs), the benefit of having the speed ratio control is not significant. However, when only one GPU is available for running all three processes (**Actor**, **P-learner**, **V-learner**), we can see that without the ratio control, the learning curves on all six benchmark tasks slow down. We believe this is because all three processes are trying to run as fast as possible, resulting in competition for GPU utilization, which slows overall learning. Adding the ratio control helps balance GPU resource utilization among the three processes. Thus, even with one GPU, the learning performance with ratio control is quite similar to that with two GPUs.

Table B.4: Hyper-parameter setup for the *Ball Balancing* task.

Hyper-parameter	PQL(ours)	PPO
Num. Environments	1,024	1,024
Critic Learning Rate	5×10^{-4}	5×10^{-4}
Actor Learning Rate	5×10^{-4}	5×10^{-4}
Optimizer	Adam	Adam
Target Update Rate (τ)	5×10^{-2}	-
Batch Size	4,096	4,096
Horizon length	1	16
Num. Epochs	12	5
Discount Factor(γ)	0.99	0.99
Normalized Observations	True	True
Gradient Clipping	True	True
Exploration Policy	Mix	-
N -step target	3	-
Warm-up Steps	32	-
Replay Buffer Size	10^6	-
Clip Ratio	-	0.2
GAE	-	True
λ	-	0.95

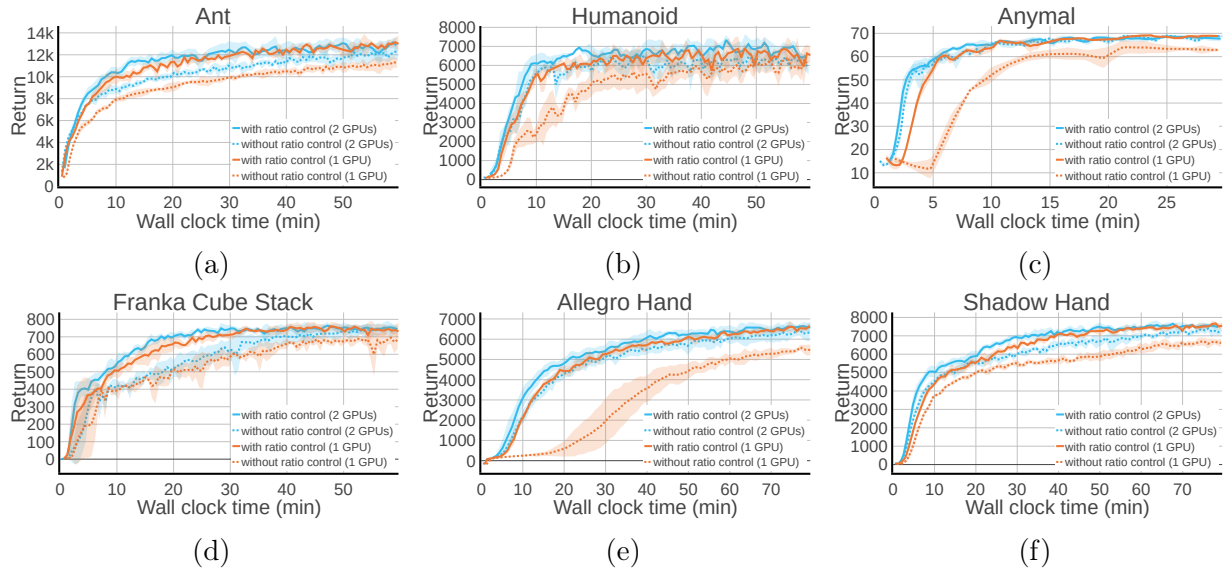


Figure B.2: Comparison of the learning performance with and without the speed ratio control ($\beta_{p:v}, \beta_{a:v}$) on two RTX3090 GPUs and on 1 RTX3090 GPU, respectively.

GPU hardware The simulation speed and network training speed vary across different GPU models. In Table B.3, we list how much time it takes for the simulator to generate 1M environment interaction data with 4096 parallel environments on four machines with

different GPU models. In our test, the simulation speed on different GPU models is as follows: GeForce 3090 > Tesla A100 > Tesla V100 > GeForce 2080Ti. We test PQL performance on all these four different machine configurations (Table B.3). Figure B.3c and Figure B.3d show that different GPU models affect the policy learning speed, especially on complex tasks like *Shadow Hand* which takes more simulation time.

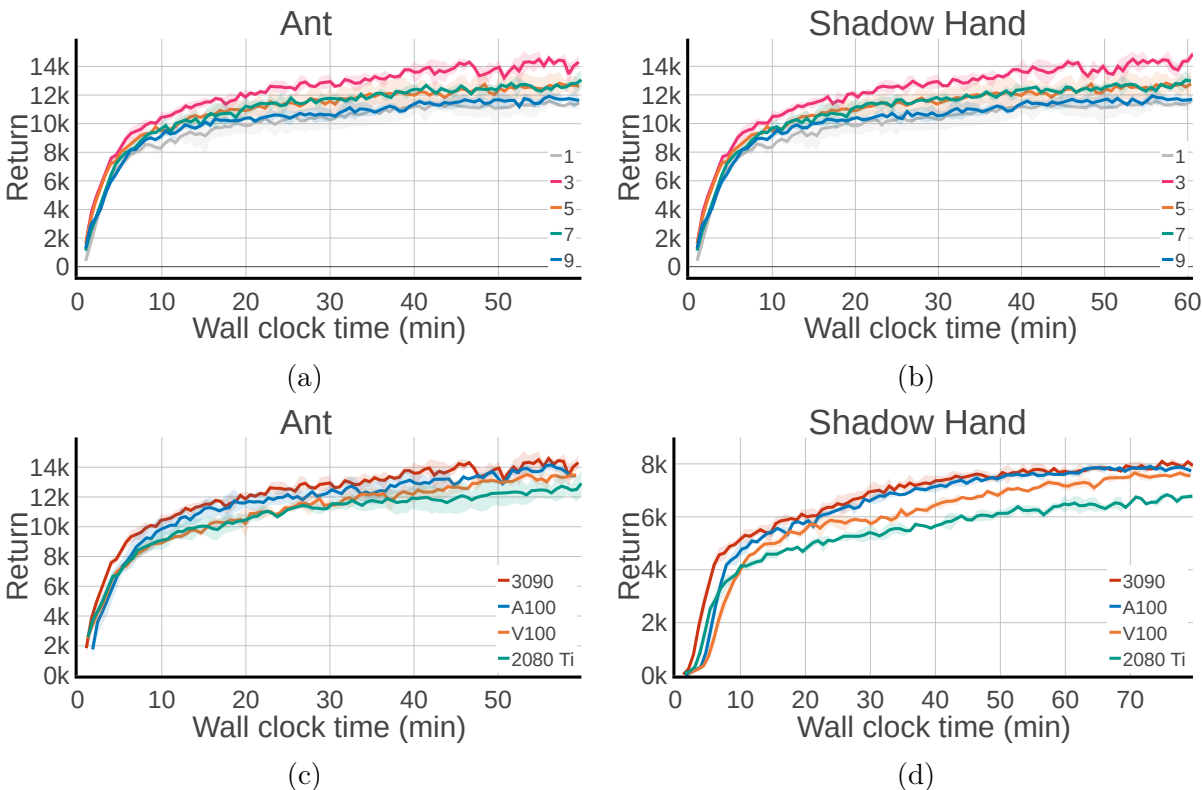


Figure B.3: (a) and (b): effect of n -step return. $n = 3$ performs the best. (c) and (d): effect of GPU models used for running PQL. Overall, we see that PQL works robustly across different GPU models, and running on newer GPUs tends to give faster learning.

PQL for SAC As discussed above, PQL framework is flexible and can be combined with different Q -learning methods. Here, we show that PQL can be combined with SAC as well. Figure B.4 shows that adding the PQL framework to SAC substantially speeds up the learning speed of SAC.

Sample efficiency compared to baselines Figure B.5 shows the sample efficiency of each algorithm on different environments. Overall, we see that PQL achieves the best sample efficiency. In addition, DDPG(n) also outperforms SAC(n) in terms of sample efficiency on these tasks.

Sweep over different $\beta_{a,v}$ and $\beta_{p,v}$ Figure B.6 shows the complete learning curves with different $\beta_{p,v}$ values and different number of environments. Similarly, Figure B.7 shows the learning curves for different $\beta_{a,v}$.

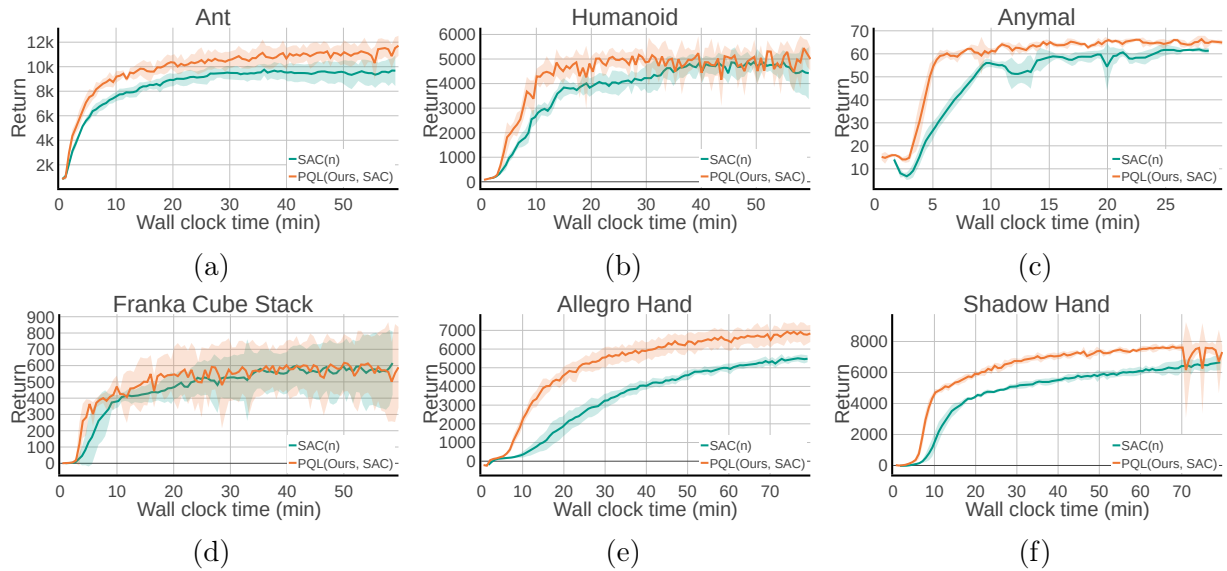


Figure B.4: We apply our parallel Q -learning to SAC. PQL + SAC achieves faster learning than SAC itself.

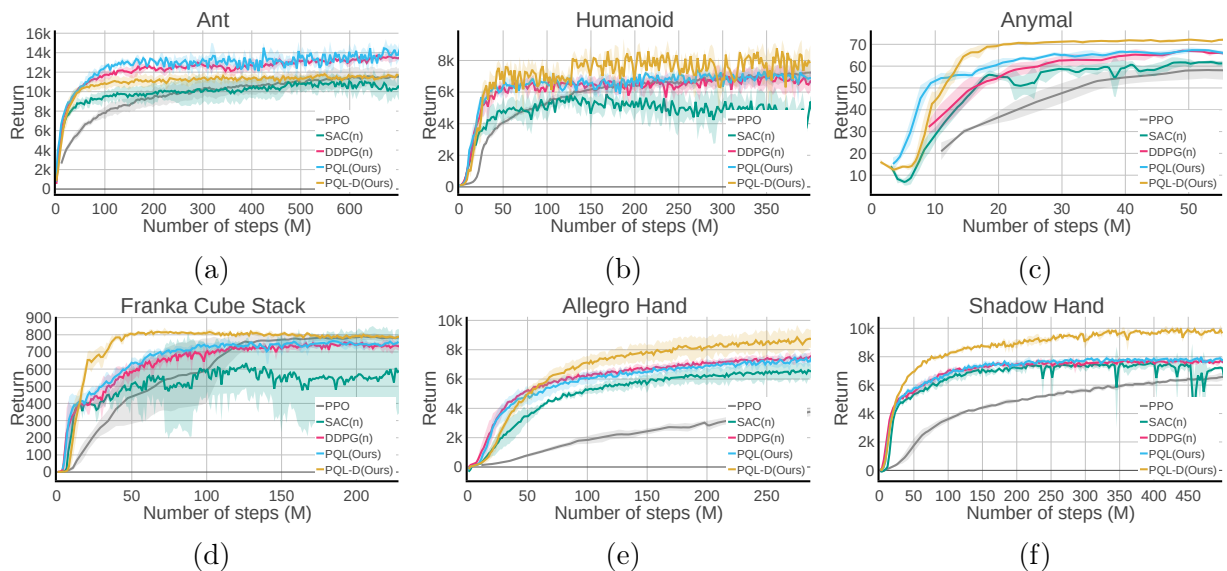


Figure B.5: Similar to Figure 4.3, we show that our method (PQL) also achieves better sample efficiency than baselines.

Comparison of our implementation with RL-games In this work, we implemented all the algorithms (PQL and all the baselines) from scratch, as it gives us the most flexibility in exploring different design choices that can affect learning performance. To show that our codebase provides good performance, we compare it against the most commonly used RL codebase used for Isaac Gym, which is RL-games [174]. However, RL-games only support PPO and SAC. Hence, we compare our implementations of PPO and SAC against the ones in RL-games.

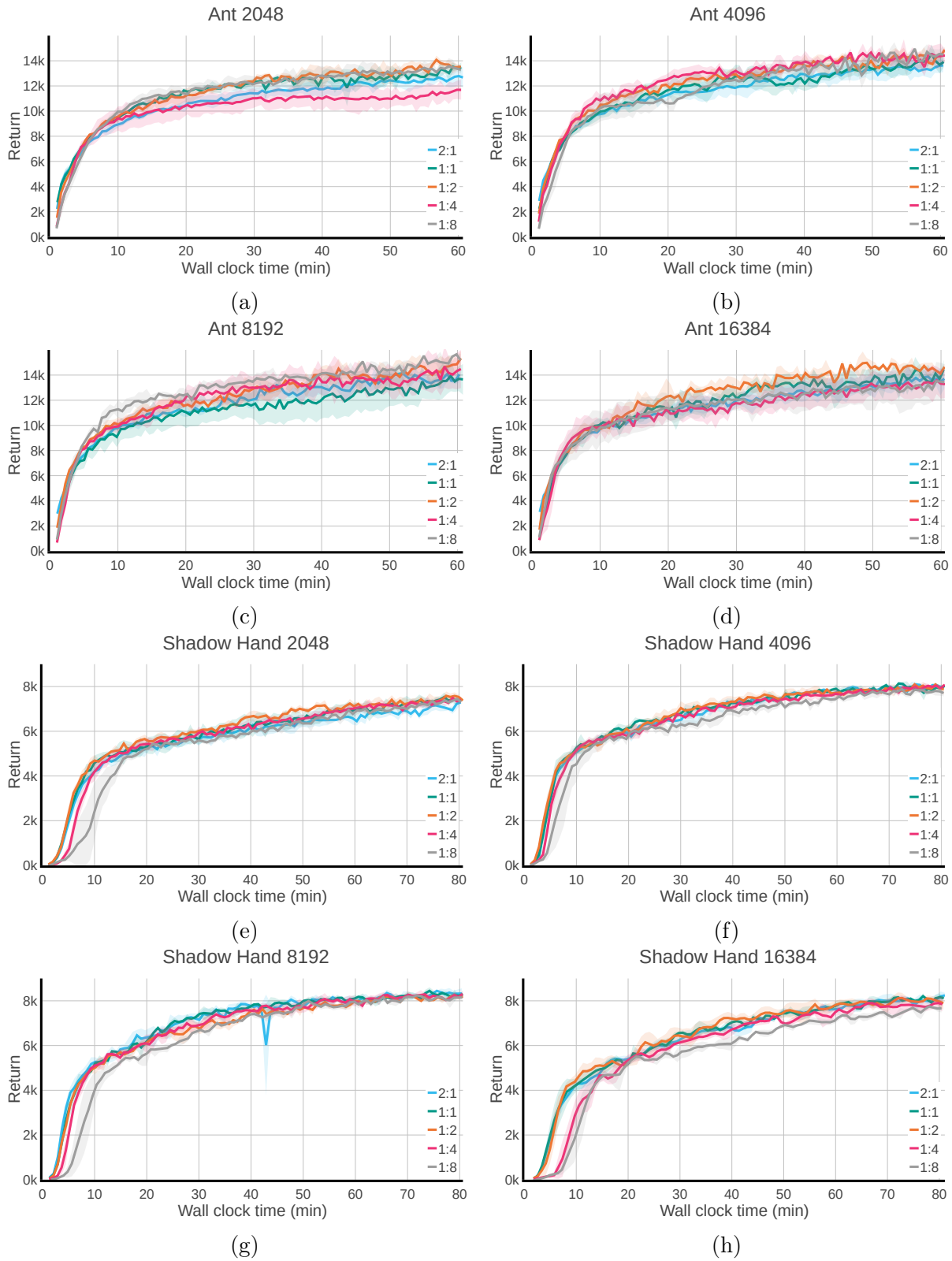


Figure B.6: Learning curves for different $\beta_{p:v}$.

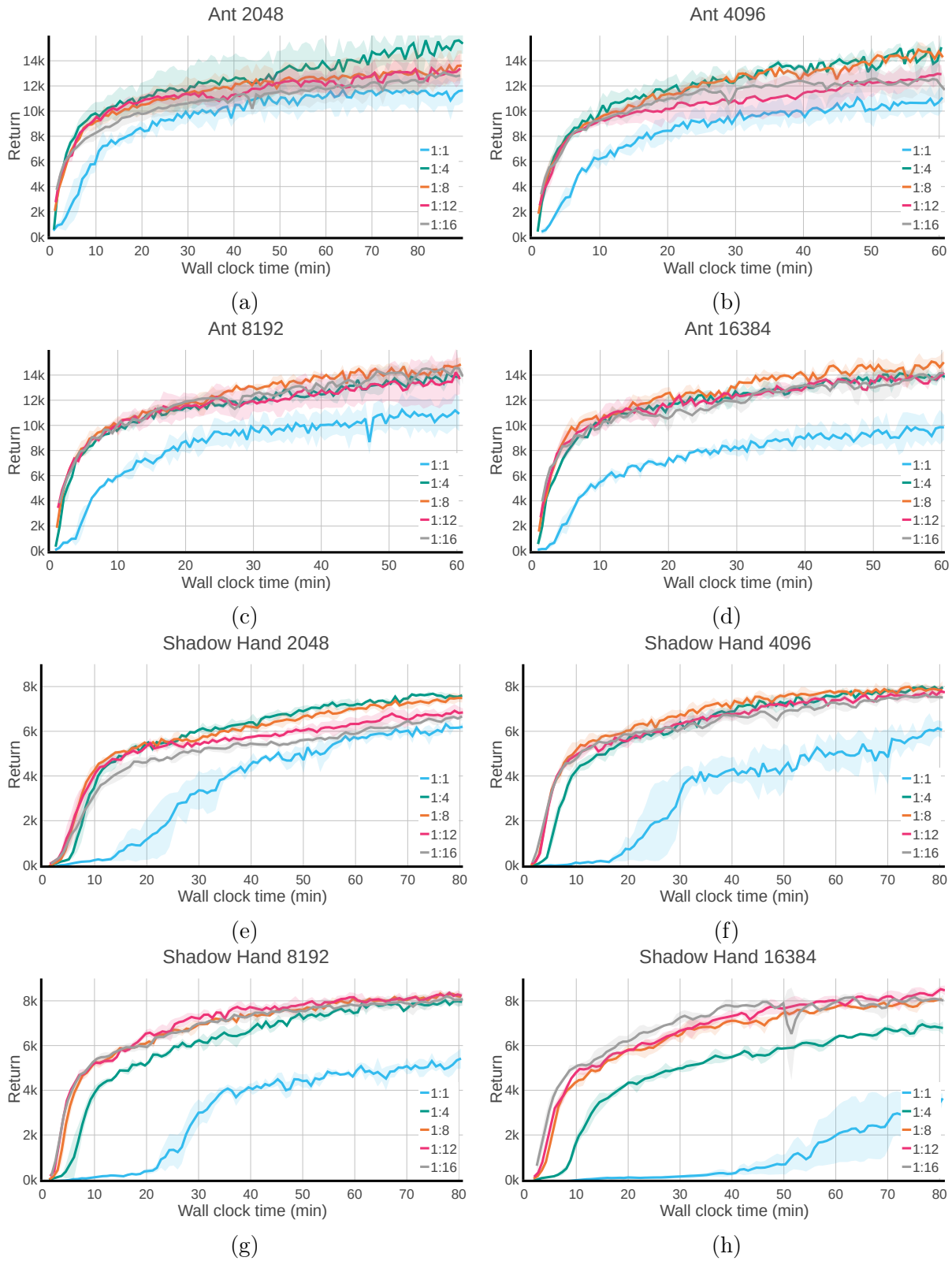


Figure B.7: Learning curves for different $\beta_{a,v}$.

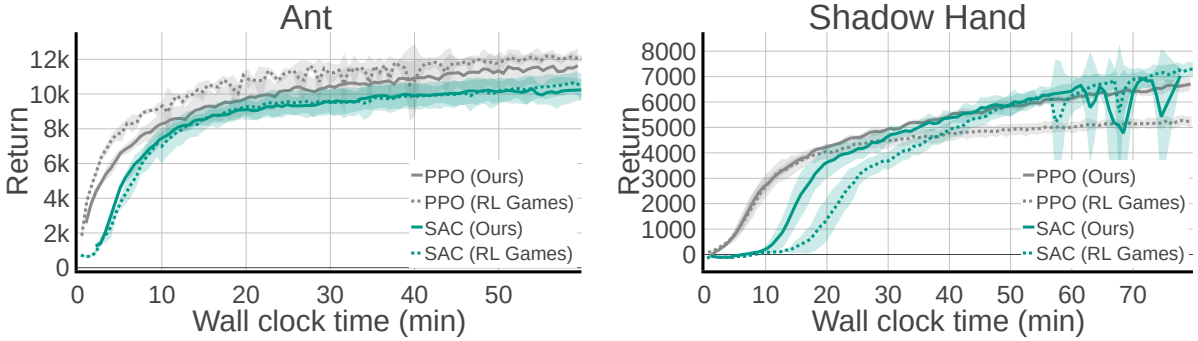


Figure B.8: Comparison between our implementations of PPO and SAC against the ones provided in RL-games. We can see that both codebases provide similar performance, showing that our implementation is good and reliable. On *Shadow Hand*, our PPO learns even faster and better than the PPO in RL-games.

Distributional critic update We investigate how a distributional version of the critic update affects the policy learning performance. Here, we utilize categorical parameterization that outputs a discrete-value distribution defined over a fixed set of atoms z_i [108]. We use the same hyper-parameters across the six tasks, where the number of atoms $l = 51$ and the bounds on the support from $(-10, 10)$. To make sure the values lie on the support defined by the atoms, we scale the reward into a similar range via different scaling factors shown in Table B.2 and apply the categorical projection operator before minimizing the cross-entropy.

References

- [1] M. T. Mason and J. K. Salisbury Jr, “Robot hands and the mechanics of manipulation,” 1985.
- [2] A. M. Okamura, N. Smaby, and M. 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)*, IEEE, vol. 1, 2000, pp. 255–262.
- [3] N. C. Daffe, A. Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge, “Extrinsic dexterity: In-hand manipulation with external forces,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 1578–1585.
- [4] I. Mordatch, Z. Popović, and E. Todorov, “Contact-invariant optimization for hand manipulation,” in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, 2012, pp. 137–144.
- [5] Y. Bai and C. K. Liu, “Dexterous manipulation using both palm and fingers,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 1560–1565.
- [6] B. Sundaralingam and T. Hermans, “Relaxed-rigidity constraints: Kinematic trajectory optimization and collision avoidance for in-grasp manipulation,” *Autonomous Robots*, vol. 43, no. 2, pp. 469–483, 2019.
- [7] D. Rus, “In-hand dexterous manipulation of piecewise-smooth 3-d objects,” *The International Journal of Robotics Research*, vol. 18, no. 4, pp. 355–381, 1999.
- [8] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, “Predictive sampling: Real-time behaviour synthesis with mujoco,” *arXiv preprint arXiv:2212.00541*, 2022.
- [9] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, “Learning to poke by poking: Experiential learning of intuitive physics,” in *Advances in neural information processing systems*, 2016, pp. 5074–5082.
- [10] V. Kumar, E. Todorov, and S. Levine, “Optimal control with learned local models: Application to dexterous manipulation,” in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 378–383.
- [11] V. Kumar, A. Gupta, E. Todorov, and S. Levine, “Learning dexterous manipulation policies from experience and imitation,” *arXiv preprint arXiv:1611.05095*, 2016.

- [12] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, “Deep dynamics models for learning dexterous manipulation,” in *Conference on Robot Learning*, PMLR, 2020, pp. 1101–1112.
- [13] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” in *International conference on machine learning*, PMLR, 2017, pp. 2778–2787.
- [14] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” *arXiv preprint arXiv:1810.12894*, 2018.
- [15] O. M. Andrychowicz, B. Baker, M. Chociej, *et al.*, “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.
- [16] OpenAI, I. Akkaya, M. Andrychowicz, *et al.*, “Solving rubik’s cube with a robot hand,” *arXiv preprint arXiv:1910.07113*, 2019.
- [17] H. Van Hoof, T. Hermans, G. Neumann, and J. Peters, “Learning robot in-hand manipulation with tactile features,” in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2015, pp. 121–127.
- [18] T. Chen, J. Xu, and P. Agrawal, “A system for general in-hand object re-orientation,” in *Conference on Robot Learning*, PMLR, 2022, pp. 297–307.
- [19] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal, “Visual dexterity: In-hand reorientation of novel and complex object shapes,” *Science Robotics*, vol. 8, no. 84, eadc9244, 2023.
- [20] I. Radosavovic, X. Wang, L. Pinto, and J. Malik, “State-only imitation learning for dexterous manipulation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 7865–7871.
- [21] J. Wang, Y. Qin, K. Kuang, Y. Korkmaz, A. Gurumoorthy, H. Su, and X. Wang, “Cyberdemo: Augmenting simulated human demonstration for real-world dexterous manipulation,” *arXiv preprint arXiv:2402.14795*, 2024.
- [22] J. Ye, J. Wang, B. Huang, Y. Qin, and X. Wang, “Learning continuous grasping function with a dexterous hand from human demonstrations,” *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2882–2889, 2023.
- [23] Y. Qin, Y.-H. Wu, S. Liu, H. Jiang, R. Yang, Y. Fu, and X. Wang, “Dexmv: Imitation learning for dexterous manipulation from human videos,” in *European Conference on Computer Vision*, Springer, 2022, pp. 570–587.
- [24] A. Gupta, C. Eppner, S. Levine, and P. Abbeel, “Learning dexterous manipulation for a soft robotic hand from human demonstrations,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 3786–3793.
- [25] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

- [26] Z. Hu, A. Rovinsky, J. Luo, V. Kumar, A. Gupta, and S. Levine, “Reboot: Reuse data for bootstrapping efficient real-world dexterous manipulation,” *arXiv preprint arXiv:2309.03322*, 2023.
- [27] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, “Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 6664–6671.
- [28] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, “Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost,” in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 3651–3657.
- [29] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, “The ingredients of real world robotic reinforcement learning,” in *International Conference on Learning Representations*, 2019.
- [30] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, “Isaac gym: High performance gpu-based physics simulation for robot learning,” *arXiv preprint arXiv:2108.10470*, 2021.
- [31] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning robust perceptive locomotion for quadrupedal robots in the wild,” *Science Robotics*, vol. 7, no. 62, eabk2822, 2022.
- [32] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, “Learning by cheating,” in *Conference on Robot Learning*, PMLR, 2020, pp. 66–75.
- [33] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, “Rapid locomotion via reinforcement learning,” *Robotics: Science and Systems (RSS)*, 2022.
- [34] T. Chen, J. Xu, and P. Agrawal, “A system for general in-hand object re-orientation,” in *Conference on Robot Learning*, PMLR, 2022, pp. 297–307.
- [35] T. Chen, E. Cousineau, N. Kuppaswamy, and P. Agrawal, “Vegetable peeling: A case study in constrained dexterous manipulation,” in *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023*, 2023.
- [36] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2017, pp. 23–30.
- [37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [38] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [39] M. T. Mason, J. K. Salisbury, and J. K. Parker, *Robot hands and the mechanics of manipulation*. The MIT Press, 1989.

- [40] 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.
- [41] V. Kumar, Y. Tassa, T. Erez, and E. Todorov, “Real-time behaviour synthesis for dynamic hand-manipulation,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 6808–6815.
- [42] N. Furukawa, A. Namiki, S. Taku, and M. Ishikawa, “Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, IEEE, 2006, pp. 181–187.
- [43] A. Bhatt, A. Sieler, S. Puhlmann, and O. Brock, “Surprisingly robust in-hand manipulation: An empirical study,” *Robotics: Science and Systems (RSS)*, 2021.
- [44] B. Calli, A. Kimmel, K. Hang, K. Bekris, and A. Dollar, “Path planning for within-hand manipulation over learned representations of safe states,” in *International Symposium on Experimental Robotics*, Springer, 2018, pp. 437–447.
- [45] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,” *Robotics: Science and Systems (RSS)*, 2017.
- [46] G. Khandate, M. Haas-Heger, and M. Ciocarlie, “On the feasibility of learning finger-gaiting in-hand manipulation with intrinsic sensing,” in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 2752–2758.
- [47] T. Ishihara, A. Namiki, M. Ishikawa, and M. Shimojo, “Dynamic pen spinning using a high-speed multifingered hand with high-speed tactile sensor,” in *6th IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2006, pp. 258–263.
- [48] B. Calli and A. M. Dollar, “Vision-based model predictive control for within-hand precision manipulation with underactuated grippers,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 2839–2845.
- [49] S. Abondance, C. B. Teeple, and R. J. Wood, “A dexterous soft robotic hand for delicate in-hand manipulation,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5502–5509, 2020.
- [50] A. S. Morgan, K. Hang, B. Wen, K. Bekris, and A. M. Dollar, “Complex in-hand manipulation via compliance-enabled finger gaiting and multi-modal planning,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4821–4828, 2022.
- [51] R. Jeong, J. T. Springenberg, J. Kay, D. Zheng, Y. Zhou, A. Galashov, N. Heess, and F. Nori, “Learning dexterous manipulation from suboptimal experts,” in *Conference on Robot Learning*, PMLR, 2020, pp. 915–934.
- [52] L. Sievers, J. Pitz, and B. Bäuml, “Learning purely tactile in-hand manipulation with a torque-controlled hand,” in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 2745–2751. DOI: [10.1109/ICRA46639.2022.9812093](https://doi.org/10.1109/ICRA46639.2022.9812093).

- [53] A. Allshire, M. Mittal, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, and A. Garg, “Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 11 802–11 809.
- [54] G. B. Margolis, T. Chen, K. Paigwar, X. Fu, D. Kim, S. Kim, and P. Agrawal, “Learning to jump from pixels,” in *Conference on Robot Learning*, PMLR, 2022, pp. 1025–1034.
- [55] A. Kumar, Z. Fu, D. Pathak, and J. Malik, “Rma: Rapid motor adaptation for legged robots,” *Robotics: Science and Systems (RSS)*, 2021.
- [56] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning quadrupedal locomotion over challenging terrain,” *Science robotics*, vol. 5, no. 47, eabc5986, 2020.
- [57] J. Xu, T. Aykut, D. Ma, and E. Steinbach, “6dls: Modeling nonplanar frictional surface contacts for grasping using 6-d limit surfaces,” *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2099–2116, 2021.
- [58] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar, “Robel: Robotics benchmarks for learning with low-cost robots,” in *Conference on Robot Learning*, PMLR, 2020, pp. 1300–1313.
- [59] J. L. Hintze and R. D. Nelson, “Violin plots: A box plot-density trace synergism,” *The American Statistician*, vol. 52, no. 2, pp. 181–184, 1998.
- [60] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, *et al.*, “Dextreme: Transfer of agile in-hand manipulation from simulation to reality,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 5977–5984.
- [61] C. Chen, P. Culbertson, M. Lepert, M. Schwager, and J. Bohg, “Trajectorytree: Trajectory optimization meets tree search for planning multi-contact dexterous manipulation,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 8262–8268.
- [62] T. Pang, H. Suh, L. Yang, and R. Tedrake, “Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models,” *arXiv preprint arXiv:2206.10787*, 2022.
- [63] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal, “An end-to-end differentiable framework for contact-aware robot design,” *Robotics: Science and Systems*, 2021.
- [64] S. Brahmhatt, C. Ham, C. C. Kemp, and J. Hays, “ContactDB: Analyzing and predicting grasp contact via thermal imaging,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [65] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *Proceedings of the Sixteenth International Conference on Machine Learning*, vol. 99, 1999, pp. 278–287.

- [66] C. Choy, J. Gwak, and S. Savarese, “4d spatio-temporal convnets: Minkowski convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3075–3084.
- [67] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Oct. 2014, pp. 1724–1734.
- [68] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, “Sim-to-real: Learning agile locomotion for quadruped robots,” *Robotics: Science and Systems (RSS)*, 2018.
- [69] N. Hansen, S. D. Müller, and P. Koumoutsakos, “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es),” *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.
- [70] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, *et al.*, “Dextreme: Transfer of agile in-hand manipulation from simulation to reality,” *arXiv preprint arXiv:2210.13702*, 2022.
- [71] Z.-H. Yin, B. Huang, Y. Qin, Q. Chen, and X. Wang, “Rotating without seeing: Towards in-hand dexterity through touch,” *arXiv preprint arXiv:2303.10880*, 2023.
- [72] H. Qi, A. Kumar, R. Calandra, Y. Ma, and J. Malik, “In-hand object rotation via rapid motor adaptation,” in *Conference on Robot Learning*, PMLR, 2023, pp. 1722–1732.
- [73] B. Calli, K. Srinivasan, A. Morgan, and A. M. Dollar, “Learning modes of within-hand manipulation,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 3145–3151.
- [74] X. Da, Z. Xie, D. Hoeller, B. Boots, A. Anandkumar, Y. Zhu, B. Babich, and A. Garg, “Learning a contact-adaptive controller for robust, efficient legged locomotion,” in *Conference on Robot Learning*, PMLR, 2021, pp. 883–894.
- [75] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, “Reinforcement learning for robust parameterized locomotion control of bipedal robots,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 2811–2817.
- [76] J. Pitz, L. Röstel, L. Sievers, and B. Bäuml, “Dextrous tactile in-hand manipulation using a modular reinforcement learning architecture,” *arXiv preprint arXiv:2303.04705*, 2023.
- [77] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, “Asymmetric actor critic for image-based robot learning,” *arXiv preprint arXiv:1710.06542*, 2017.
- [78] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, “Deep object pose estimation for semantic robotic grasping of household objects,” *arXiv preprint arXiv:1809.10790*, 2018.

- [79] M. Deitke, D. Schwenk, J. Salvador, L. Weihs, O. Michel, E. VanderBilt, L. Schmidt, K. Ehsani, A. Kembhavi, and A. Farhadi, “Objaverse: A universe of annotated 3d objects,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 13 142–13 153.
- [80] J. Ho and S. Ermon, “Generative adversarial imitation learning,” *Advances in neural information processing systems*, vol. 29, 2016.
- [81] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, “Amp: Adversarial motion priors for stylized physics-based character control,” *ACM Transactions on Graphics (ToG)*, vol. 40, no. 4, pp. 1–20, 2021.
- [82] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [83] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, Feb. 1987, Conference Name: IEEE Journal on Robotics and Automation, ISSN: 2374-8710. DOI: [10.1109/JRA.1987.1087068](https://doi.org/10.1109/JRA.1987.1087068).
- [84] R. Tedrake and the Drake Development Team, *Drake: Model-based design and verification for robotics*, 2019. [Online]. Available: <https://drake.mit.edu>.
- [85] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, *et al.*, “Grounded sam: Assembling open-world models for diverse visual tasks,” *arXiv preprint arXiv:2401.14159*, 2024.
- [86] K. Shaw, A. Agarwal, and D. Pathak, “Leap hand: Low-cost, efficient, and anthropomorphic hand for robot learning,” *arXiv preprint arXiv:2309.06440*, 2023.
- [87] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [88] 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*, vol. 4, no. 26, eaau5872, 2019.
- [89] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, “Distributed prioritized experience replay,” *arXiv preprint arXiv:1803.00933*, 2018.
- [90] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures,” in *International Conference on Machine Learning*, PMLR, 2018, pp. 1407–1416.
- [91] A. Allshire, M. Mittal, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, and A. Garg, “Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger,” *arXiv preprint arXiv:2108.09779*, 2021.

- [92] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, “Learning to walk in minutes using massively parallel deep reinforcement learning,” in *Conference on Robot Learning*, PMLR, 2022, pp. 91–100.
- [93] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, *et al.*, “Massively parallel methods for deep reinforcement learning,” *arXiv preprint arXiv:1507.04296*, 2015.
- [94] 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.
- [95] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” 2016.
- [96] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, “Data-efficient deep reinforcement learning for dexterous manipulation,” *arXiv preprint arXiv:1704.03073*, 2017.
- [97] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.
- [98] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [99] H. Hasselt, “Double q-learning,” *Advances in neural information processing systems*, vol. 23, 2010.
- [100] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, “Recurrent experience replay in distributed reinforcement learning,” in *International conference on learning representations*, 2018.
- [101] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*, PMLR, 2016, pp. 1928–1937.
- [102] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, “Reinforcement learning through asynchronous advantage actor-critic on a gpu,” *arXiv preprint arXiv:1611.06256*, 2016.
- [103] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski, “Seed rl: Scalable and efficient deep-rl with accelerated central inference,” *arXiv preprint arXiv:1910.06591*, 2019.
- [104] A. V. Clemente, H. N. Castejón, and A. Chandra, “Efficient parallel methods for deep reinforcement learning,” *arXiv preprint arXiv:1705.04862*, 2017.
- [105] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, *et al.*, “Emergence of locomotion behaviours in rich environments,” *arXiv preprint arXiv:1707.02286*, 2017.
- [106] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, “Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames,” *arXiv preprint arXiv:1911.00357*, 2019.

- [107] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.
- [108] M. G. Bellemare, W. Dabney, and R. Munos, “A distributional perspective on reinforcement learning,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 449–458.
- [109] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan, and I. Stoica, “Ray: A distributed framework for emerging ai applications. corr abs/1712.05889 (2017),” *arXiv preprint arXiv:1712.05889*, 2017.
- [110] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International conference on machine learning*, PMLR, 2018, pp. 1587–1596.
- [111] J. Achiam, “Spinning Up in Deep Reinforcement Learning,” 2018.
- [112] Y. Fujita, P. Nagarajan, T. Kataoka, and T. Ishikawa, “Chainerrl: A deep reinforcement learning library,” *Journal of Machine Learning Research*, vol. 22, no. 77, pp. 1–14, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-376.html>.
- [113] G. Yang, A. Ajay, and P. Agrawal, “Overcoming the spectral bias of neural value approximation,” *arXiv preprint arXiv:2206.04672*, 2022.
- [114] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, *et al.*, “Bootstrap your own latent—a new approach to self-supervised learning,” *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020.
- [115] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *International conference on machine learning*, PMLR, 2020, pp. 1597–1607.
- [116] W. Bottcher, P. Machado, N. Lama, and T. M. McGinnity, “Object recognition for robotics from tactile time series data utilising different neural network architectures,” in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–8.
- [117] M. Bauza, O. Canal, and A. Rodriguez, “Tactile mapping and localization from high-resolution tactile imprints,” in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 3811–3817.
- [118] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, IEEE Computer Society, 2005, pp. 539–546. DOI: [10.1109/CVPR.2005.202](https://doi.org/10.1109/CVPR.2005.202). [Online]. Available: <https://doi.org/10.1109/CVPR.2005.202>.
- [119] A. van den Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *CoRR*, vol. abs/1807.03748, 2018. arXiv: [1807.03748](https://arxiv.org/abs/1807.03748). [Online]. Available: <http://arxiv.org/abs/1807.03748>.

- [120] Z. Kappassov, J.-A. Corrales, and V. Perdereau, “Tactile sensing in dexterous robot hands,” *Robotics and Autonomous Systems*, vol. 74, pp. 195–220, 2015.
- [121] S. Sundaram, P. Kellnhofer, Y. Li, J.-Y. Zhu, A. Torralba, and W. Matusik, “Learning the signatures of the human grasp using a scalable tactile glove,” *Nature*, vol. 569, no. 7758, pp. 698–702, 2019.
- [122] C. M. Boutry, M. Negre, M. Jorda, O. Vardoulis, A. Chortos, O. Khatib, and Z. Bao, “A hierarchically patterned, bioinspired e-skin able to detect the direction of applied pressure for robotics,” *Science Robotics*, vol. 3, no. 24, eaau6914, 2018.
- [123] M. R. Cutkosky, R. D. Howe, and W. R. Provancher, “Force and tactile sensors.,” *Springer Handbook of Robotics*, vol. 100, pp. 455–476, 2008.
- [124] R. Bhirangi, T. Hellebrekers, C. Majidi, and A. Gupta, “Reskin: Versatile, replaceable, lasting tactile skins,” *arXiv preprint arXiv:2111.00071*, 2021.
- [125] B. Ward-Cherrier, N. Pestell, L. Cramphorn, B. Winstone, M. E. Giannaccini, J. Rossiter, and N. F. Lepora, “The tactip family: Soft optical tactile sensors with 3d-printed biomimetic morphologies,” *Soft robotics*, vol. 5, no. 2, pp. 216–227, 2018.
- [126] N. Kuppuswamy, A. Alspach, A. Uttamchandani, S. Creasey, T. Ikeda, and R. Tedrake, “Soft-bubble grippers for robust and perceptive manipulation,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 9917–9924.
- [127] M. Lambeta, P.-W. Chou, S. Tian, B. Yang, B. Maloon, V. R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, *et al.*, “Digit: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation,” *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 3838–3845, 2020.
- [128] Y. She, S. Wang, S. Dong, N. Sunil, A. Rodriguez, and E. Adelson, “Cable manipulation with a tactile-reactive gripper,” *The International Journal of Robotics Research*, vol. 40, no. 12-14, pp. 1385–1401, 2021.
- [129] C. Wang, S. Wang, B. Romero, F. Veiga, and E. Adelson, “Swingbot: Learning physical features from in-hand tactile exploration for dynamic swing-up manipulation,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5633–5640.
- [130] S. Dong, D. K. Jha, D. Romeres, S. Kim, D. Nikovski, and A. Rodriguez, “Tactile-rl for insertion: Generalization to objects of unknown geometry,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 6437–6443.
- [131] A. N. Chaudhury, T. Man, W. Yuan, and C. G. Atkeson, “Using collocated vision and tactile sensors for visual servoing and localization,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3427–3434, 2022.
- [132] J. Yang, H. Liu, F. Sun, and M. Gao, “Object recognition using tactile and image information,” in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, IEEE, 2015, pp. 1746–1751.

- [133] T. Corradi, P. Hall, and P. Irvani, “Object recognition combining vision and touch,” *Robotics and biomimetics*, vol. 4, no. 1, pp. 1–10, 2017.
- [134] W. Yuan, Y. Mo, S. Wang, and E. H. Adelson, “Active clothing material perception using tactile sensing and deep learning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 4842–4849.
- [135] R. Li and E. H. Adelson, “Sensing and recognizing surface textures using a gelsight sensor,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1241–1247.
- [136] Z. A. Pezzementi, C. Reyda, and G. D. Hager, “Object mapping, recognition, and localization from tactile geometry,” in *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, IEEE, 2011, pp. 5942–5948. DOI: [10.1109/ICRA.2011.5980363](https://doi.org/10.1109/ICRA.2011.5980363). [Online]. Available: <https://doi.org/10.1109/ICRA.2011.5980363>.
- [137] S. Pohtongkam and J. Srinonchat, “Tactile object recognition for humanoid robots using new designed piezoresistive tactile sensor and dcnn,” *Sensors*, vol. 21, no. 18, p. 6024, 2021.
- [138] J. Lin, R. Calandra, and S. Levine, “Learning to identify object instances by touch: Tactile recognition via multimodal matching,” *CoRR*, vol. abs/1903.03591, 2019. arXiv: [1903.03591](https://arxiv.org/abs/1903.03591). [Online]. Available: <http://arxiv.org/abs/1903.03591>.
- [139] N. Jamali, C. Ciliberto, L. Rosasco, and L. Natale, “Active perception: Building objects’ models using tactile exploration,” in *16th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2016, Cancun, Mexico, November 15-17, 2016*, IEEE, 2016, pp. 179–185. DOI: [10.1109/HUMANOIDS.2016.7803275](https://doi.org/10.1109/HUMANOIDS.2016.7803275). [Online]. Available: <https://doi.org/10.1109/HUMANOIDS.2016.7803275>.
- [140] U. Martinez-Hernandez, G. Metta, T. J. Dodd, T. J. Prescott, L. Natale, and N. F. Lepora, “Active contour following to explore object shape with robot touch,” in *2013 World Haptics Conference, WHC 2013, Daejeon, Korea (South), April 14-17, 2013*, IEEE, 2013, pp. 341–346. DOI: [10.1109/WHC.2013.6548432](https://doi.org/10.1109/WHC.2013.6548432). [Online]. Available: <https://doi.org/10.1109/WHC.2013.6548432>.
- [141] Z. Yi, R. Calandra, F. Veiga, H. van Hoof, T. Hermans, Y. Zhang, and J. Peters, “Active tactile object exploration with gaussian processes,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016*, IEEE, 2016, pp. 4925–4930. DOI: [10.1109/IROS.2016.7759723](https://doi.org/10.1109/IROS.2016.7759723). [Online]. Available: <https://doi.org/10.1109/IROS.2016.7759723>.
- [142] D. Dri , P. Englert, and M. Toussaint, “Active learning with query paths for tactile object shape exploration,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, IEEE, 2017, pp. 65–72. DOI: [10.1109/IROS.2017.8202139](https://doi.org/10.1109/IROS.2017.8202139). [Online]. Available: <https://doi.org/10.1109/IROS.2017.8202139>.
- [143] J. Xu, S. Song, and M. Ciocarlie, “Tandem: Learning joint exploration and decision making with tactile sensors,” *arXiv preprint arXiv:2203.00798*, 2022.

- [144] Y. Karayiannidis, C. Smith, F. E. Vina, and D. Kragic, “Online contact point estimation for uncalibrated tool use,” *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2488–2494, 2014.
- [145] N. Likar and L. Žlajpah, “External joint torque-based estimation of contact information,” *International Journal of Advanced Robotic Systems*, vol. 11, no. 7, p. 107, 2014.
- [146] R. Li, R. Platt, W. Yuan, A. ten Pas, N. Roscup, M. A. Srinivasan, and E. Adelson, “Localization and manipulation of small parts using gelsight tactile sensing,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 3988–3993.
- [147] S. Luo, W. Mou, K. Althoefer, and H. Liu, “Localizing the object contact through matching tactile features with visual map,” *CoRR*, vol. abs/1708.04441, 2017. arXiv: [1708.04441](http://arxiv.org/abs/1708.04441). [Online]. Available: <http://arxiv.org/abs/1708.04441>.
- [148] M. C. Koval, N. S. Pollard, and S. S. Srinivasa, “Pose estimation for planar contact manipulation with manifold particle filters,” *Int. J. Robotics Res.*, vol. 34, no. 7, pp. 922–945, 2015. DOI: [10.1177/0278364915571007](https://doi.org/10.1177/0278364915571007). [Online]. Available: <https://doi.org/10.1177/0278364915571007>.
- [149] G. Kissoum and V. Perdereau, “Simultaneous tactile localization and reconstruction of an object during robotic manipulation,” in *2021 20th International Conference on Advanced Robotics (ICAR)*, IEEE, 2021, pp. 948–954.
- [150] A. Murali, Y. Li, D. Gandhi, and A. Gupta, “Learning to grasp without seeing,” in *Proceedings of the 2018 International Symposium on Experimental Robotics, ISER 2018, Buenos Aires, Argentina, November 5-8, 2018*, J. Xiao, T. Kröger, and O. Khatib, Eds., ser. Springer Proceedings in Advanced Robotics, vol. 11, Springer, 2018, pp. 375–386. DOI: [10.1007/978-3-030-33950-0_33](https://doi.org/10.1007/978-3-030-33950-0_33). [Online]. Available: https://doi.org/10.1007/978-3-030-33950-0_33.
- [151] A. Petrovskaya and O. Khatib, “Global localization of objects via touch,” *IEEE Trans. Robotics*, vol. 27, no. 3, pp. 569–585, 2011. DOI: [10.1109/TRO.2011.2138450](https://doi.org/10.1109/TRO.2011.2138450). [Online]. Available: <https://doi.org/10.1109/TRO.2011.2138450>.
- [152] M. Kaboli, D. Feng, K. Yao, P. Lanillos, and G. Cheng, “A tactile-based framework for active object learning and discrimination using multimodal robotic skin,” *IEEE Robotics Autom. Lett.*, vol. 2, no. 4, pp. 2143–2150, 2017. DOI: [10.1109/LRA.2017.2720853](https://doi.org/10.1109/LRA.2017.2720853). [Online]. Available: <https://doi.org/10.1109/LRA.2017.2720853>.
- [153] T. Chen, A. Simeonov, and P. Agrawal, *AIRobot*, <https://github.com/Improbable-AI/airobot>, 2019.
- [154] M. H. Tippur, “Design and manufacturing methods for a curved all-around camera-based tactile sensor,” 2020, p. 69.
- [155] H. Sun, K. J. Kuchenbecker, and G. Martius, “A soft thumb-sized vision-based sensor with accurate all-round force perception,” *Nature Machine Intelligence*, vol. 4, no. 2, pp. 135–145, 2022.

- [156] S. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: [10.1109/TIT.1982.1056489](https://doi.org/10.1109/TIT.1982.1056489).
- [157] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, vol. 2, 2006, pp. 1735–1742. DOI: [10.1109/CVPR.2006.100](https://doi.org/10.1109/CVPR.2006.100).
- [158] Q. Zhang, Y. Li, Y. Luo, W. Shou, M. Foshey, J. Yan, J. B. Tenenbaum, W. Matusik, and A. Torralba, “Dynamic modeling of hand-object interactions via tactile sensing,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*, IEEE, 2021, pp. 2874–2881. DOI: [10.1109/IROS51168.2021.9636361](https://doi.org/10.1109/IROS51168.2021.9636361). [Online]. Available: <https://doi.org/10.1109/IROS51168.2021.9636361>.
- [159] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, “A simple framework for contrastive learning of visual representations,” *CoRR*, vol. abs/2002.05709, 2020. arXiv: [2002.05709](https://arxiv.org/abs/2002.05709). [Online]. Available: <https://arxiv.org/abs/2002.05709>.
- [160] H. Duan, P. Wang, Y. Huang, G. Xu, W. Wei, and X. Shen, “Robotics dexterous grasping: The methods based on point cloud and deep learning,” *Frontiers Neurorobotics*, vol. 15, p. 658 280, 2021. DOI: [10.3389/fnbot.2021.658280](https://doi.org/10.3389/fnbot.2021.658280). [Online]. Available: <https://doi.org/10.3389/fnbot.2021.658280>.
- [161] L. Downs, A. Francis, N. Koenig, B. Kinman, R. Hickman, K. Reymann, T. B. McHugh, and V. Vanhoucke, “Google scanned objects: A high-quality dataset of 3d scanned household items,” in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 2553–2560.
- [162] A. X. Chang, T. Funkhouser, L. Guibas, *et al.*, “ShapeNet: An Information-Rich 3D Model Repository,” Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.
- [163] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.
- [164] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, IEEE, vol. 1, 2005, pp. 539–546.
- [165] G. Khandate, S. Shang, E. T. Chang, T. L. Saidi, J. Adams, and M. Ciocarlie, “Sampling-based exploration for reinforcement learning of dexterous manipulation,” *arXiv preprint arXiv:2303.03486*, 2023.
- [166] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, “Shortest paths in graphs of convex sets,” *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024.
- [167] K. Mamou, E. Lengyel, and A. Peters, “Volumetric hierarchical approximate convex decomposition,” in *Game Engine Gems 3*, AK Peters, 2016, pp. 141–158.

- [168] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *4th International Conference on Learning Representations (ICLR)*, 2016.
- [169] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations (ICLR)*, 2015.
- [170] K. Daniilidis, “Hand-eye calibration using dual quaternions,” *The International Journal of Robotics Research*, vol. 18, no. 3, pp. 286–298, 1999.
- [171] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, “ROS: An open-source robot operating system,” in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.
- [172] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson, “Implicit behavioral cloning,” in *Conference on Robot Learning*, PMLR, 2022, pp. 158–168.
- [173] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, “On the continuity of rotation representations in neural networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5745–5753.
- [174] D. Makoviichuk and V. Makoviyuchuk, *Rl-games: A high-performance framework for reinforcement learning*, https://github.com/Denys88/rl_games, May 2022.