論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Efficient Learning for Dexterous Manipulation through Interactive Tactile Perception
著者(和文)	太田佳
Author(English)	Kei Ota
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第12912号, 授与年月日:2024年9月20日, 学位の種別:課程博士, 審査員:篠田 浩一,岡崎 直観,横田 理央,下坂 正倫,松原 崇充,金崎 朝子
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第12912号, Conferred date:2024/9/20, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctoral Dissertation

Efficient Learning for Dexterous Manipulation through Interactive Tactile Perception

Kei Ota

Graduate Major in Artificial Intelligence
School of Computing
Tokyo Institute of Technology

Supervisor: Asako Kanezaki

July, 2024

Abstract

While recent advancements in foundation models have allowed robots to handle general tasks, applications have largely been restricted to simple pick-and-place tasks. In this thesis, we address the challenge of enabling robots to perform complex manipulation tasks that involve physical contact interactions in real-world environments. Identifying efficiency in learning in the real system and estimation of an object's representation as key challenges, this thesis studies the following three pivotal questions.

- Part I: *How can we train high-control-performance and sample-efficient RL policies?* Chapter 2 investigates whether increasing input dimensionality improves control performance and sample efficiency in model-free deep reinforcement learning (RL) algorithms. We propose an online feature extractor network (OFENet) that utilizes neural networks to generate effective representations as input for deep RL algorithms. Chapter 3 proposes a novel framework that facilitates the training of larger networks by combining OFENet, DenseNet architecture, and distributed training. This framework enables the training of 150 times larger networks, achieving approximately 2 times higher returns and 100 times better sample efficiency, making it suitable for real-world applications.

- Part II: *How can we estimate an effective object's representation for performing manipulation?* Chapter 4 proposes a method "Hypothesize, Simulate, Act, Update, and Repeat" (H-SAUR), a probabilistic estimation framework that efficiently estimates objects' articulation, an essential representation to manipulate articulated objects, from a few interactions. Chapter 5 combines tactile sensing and the framework developed in Chapter 4 to solve a general connector mating task, a partially observable task due to the small size of the tactile sensor. We introduce *Tactile Filter* which allows robots to interactively estimate the corresponding shapes of mating connectors with unknown geometries.

- Part III: *How can we solve a desired dexterous manipulation task?* Chapter 6 tackles the task of stably stacking an irregular object on top of an unknown tower. To solve this task, it is essential to estimate an extrinsic contact patch, a contact area between a grasped object and a bottom object, which is not directly observable, thus requiring a robot to estimate from raw tactile signals. We utilize the probabilistic estimation framework developed in Part II to reliably estimate the contact patch by aggregating information from multiple interactions. Then, we input the estimated contact patch into the RL framework developed in Part I. We demonstrate this combination allows the robot to efficiently solve the task by training solely in the real system.

The conclusion in Chapter 7 summarizes the answers to the above three questions, highlighting the importance of the high-control-performance and sample-efficient RL framework, and the probabilistic estimation framework that allows robots to estimate the essential object's representations from interactions. In addition, we address limitations and possible future directions, including sim2real and combination with foundation models for better efficiency and generalization capability, for further investigating the questions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Approximately a decade has passed since the introduction of AlexNet [99], a Deep Neural Networks (DNN)-based method markedly improved performance on the ImageNet challenge [37] in 2012. Subsequently, artificial intelligence (AI), particularly DNN-based methodologies, has undergone rapid advancement, fundamentally reshaping traditional paradigms. In domains such as Computer Vision (CV) and Natural Language Processing (NLP), significant progress has been achieved through the development of sophisticated architectures [78, 181, 182], enabling the training of large models on the Internet-scale data [20, 97, 148]. This transformation has expanded to robotics, where DNN applications have demonstrably improved performance across various domains.

Deep Reinforcement Learning (DRL) has proven to be well suited to solve sequential decision-making problems. Policies trained using DRL techniques have either matched or surpassed human performance in complex competitive games, such as Atari [45, 128, 154], Go [154, 161, 162, 163], chess [154, 162], StarCraft [183], and GranTurismo [51, 192]. While most achievements have been confined to simulated environments, recent advances have transcended these limitations, tackling complex real-world challenges. For example, legged robots trained primarily in simulation have completed extended hikes [125], parkour [32], and an autonomous system capable of drone racing at the level of human world champions [93].

Nevertheless, robotic manipulation remains a challenge [123]. Despite significant research efforts, applications have largely been restricted to pick-and-place tasks, which necessitate less intricate physical interactions. Recent strides in Large Language Model (LLM)- and Vision Language Model (VLM)-based approaches have enabled robots to recognize large numbers of objects in an environment, thereby enhancing generalization capabilities [18, 19, 34]. However, these methods encounter limitations when applied to tasks that require complex physical interactions, such as stable stacking of irregular objects or precise insertion of objects into tight tolerant holes. Such limitations arise because these models primarily rely on linguistic or visual data, lacking comprehensive understanding of physical interactions like force-torque and/or tactile sensing.

Several promising approaches have emerged to address these challenges and enable robots to perform dexterous manipulation tasks. These include training policies in high-

fidelity simulators and applying them to physical robots [172], employing model-based methods by providing object-specific information and numerically solving problems [159], and utilizing teleoperation data for supervised learning [50, 206]. Despite their potential, these methods have yet to achieve human-level proficiency and are generally applicable only to straightforward manipulation tasks.

To solve complex robotic manipulation tasks, it is essential to bridge the gap between current capabilities and complex physical interactions in the real world. The central question thus emerges: *How can we enable robots to accomplish complex manipulation tasks that involve physical contact interactions?*

1.2 Problem Statement

This thesis focuses on the question of how to enable robots to perform complex manipulation tasks that involve *physical contact interactions*. By *physical contact interactions* we mean that robots have to sense the contacts and utilize them to solve manipulation tasks. An example task we consider is shown in Fig. 1.1.

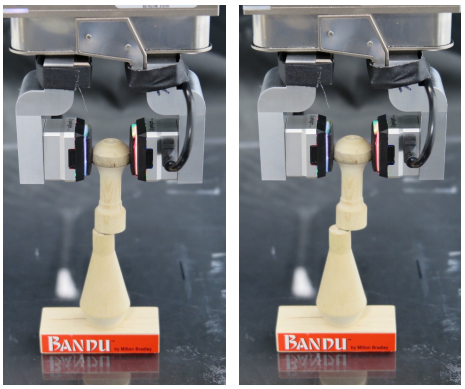


Figure 1.1: An example of the task we consider in this thesis. The robot has to stably place the grasped object without falling down the bottom tower. The left configuration is stable while the right is unstable, meaning it fails to stack the object upon release due to the misalignment.

To conceptualize the problem, Eq. (1.1) formulates a typical robotic control problem using the language of RL[1]:

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t} r(s_t, a_t) \right] \tag{1.1a}$$

subject to

$$s_{t+1} = f(s_t, a_t) \tag{1.1b}$$

$$a_t = \pi(s_t), \tag{1.1c}$$

where π is a policy to be learned through optimization, s_t and a_t are the state and action, r is the reward function, and f represents the dynamics. To solve Eq. (1.1), specifically for robotic manipulation, we need to discuss the following challenges:

1. In Eq.(1.1), we focus on using Deep RL to train the policy π as demonstrated by its impressive success in surpassing human experts described in Section 1.1. However, while previous human-level policies are primarily trained in simulation (typically massively parallelized to improve wall-clock time efficiency), it is well-known that simulating contact phenomena, which is crucial for solving manipulation tasks, is highly challenging [172]. Therefore, training the policy in the real system is essential, necessitating a sample-efficient RL algorithm due to the limited time and available robotic resources.

[1]We can also formulate the equations by using terminology from optimal control theory by using x_t, u_t, c_t as state, action, and cost function and making it minimize costs instead of minimizing rewards.

2. In Eq.(1.1b), the state space is represented as s_t. For typical robotic tasks, this state space often encompasses the pose and joint angles of the robot, along with relevant information about the environment. In addition, to address complex manipulation tasks, it is crucial to integrate tactile information, obtained from force-torque (F/T) sensor installed on the wrist and tactile sensors installed on tip of fingers. These signals provide comprehensive data regarding contact formations and geometric details about objects during interactions. For example, consider the task of stably stacking a highly irregular object on top of an unstable tower, as illustrated in Fig. 1.1. Estimating the stability of such a configuration solely from visual input would be extremely challenging. The stability of the tower is primarily influenced by the relative positioning of the grasped and bottom objects, more concretely, the contact patch between the two objects, which cannot be directly observed and must be inferred from tactile signals. Therefore, deriving useful intermediate representations from observations is essential for solving complex manipulation tasks.

Inspired by these challenges, this thesis investigates the following questions.

Question 1: High-Performance and Efficient RL How can we train high-control-performance and sample-efficient RL policies?

Question 2: Object's Representation How can we estimate an effective object's representation for performing manipulation?

Question 3: Integration How can we solve a desired dexterous manipulation task?

1.3 Contributions

This thesis consists of three parts, each addressing the above questions above to solve manipulation problems. The summaries for each part are shown in Fig. 1.2.

1.3.1 Part I

To address the question posed in *How can we train high-control-performance and sample-efficient RL policies?*, Part I is divided into two chapters as follows:

Chapter 2 investigates whether increasing the input dimensionality enhances the performance and sample efficiency of model-free deep RL algorithms. We introduce an online feature extractor network (OFENet) that employs neural networks to deliberately generate high-dimensional representations to serve as inputs for deep RL algorithms. Contrary to the conventional belief that the lower dimensionality of the state vector facilitates faster and more effective learning, we demonstrate that higher-dimensional input can achieve 1.3 times higher returns and 10 times better sample efficiency on the MuJoCo [175] standard benchmark.

In Chapter 3, we make an attempt to comprehensively understand and address the training of larger networks for deep RL. First, we show that simply increasing network capacity does not improve performance. We then propose a novel framework comprising:

1) wider networks with DenseNet connections, 2) decoupling representation learning from RL using the OFENet developed in Chapter 2, and 3) a distributed training to address overfitting issues. Employing these three strategies, we demonstrate that very large networks (150 times larger than the original ones) can be trained, resulting in approximately 2.0 times higher control performance and 100 times better sample efficiency, thus are suitable for real-world challenging tasks. Furthermore, the proposed framework can solve challenging sparse reward settings, some of which the original algorithms cannot solve.

1.3.2 Part II

This part answers the question: *How can we estimate an effective object's representation for performing manipulation?*, consisting of the two following chapters.

Chapter 4 proposes a method "Hypothesize, Simulate, Act, Update, and Repeat" (H-SAUR), a probabilistic generative framework that simultaneously generates a distribution of hypotheses about object articulation based on input observations, captures the uncertainty of these hypotheses over time, and infers plausible actions for exploration and goal-conditioned manipulation. Our results demonstrate that the proposed framework significantly outperforms the current state-of-the-art in articulated object manipulation by accurately estimating the joint configurations of objects through interactions. Furthermore, we improve the test time efficiency of H-SAUR by integrating a learned prior from learning-based vision models. We also introduce a novel benchmark, *PuzzleBoxes*, which consists of locked boxes that require multiple steps to solve. Unlike previous benchmarks, which typically involve only one joint, this new benchmark allows us to measure the manipulation performance for objects with multiple joint configurations.

In Chapter 5, we introduce *Tactile-Filter*, a combination of tactile sensing and the probabilistic estimation framework developed in Chapter 4, to address a general connector part mating task. This task involves estimating a mating part and its pose through multiple touches of both pegs and holes, which is inherently a partially observable task due to the limited size (19×14 mm) of the vision-based tactile sensor employed. We train a DNN model that takes pairs of tactile images for pegs and holes, and predict the probability if they fit together. This trained model, in conjunction with the probabilistic estimation framework, enables the robot to incrementally reduce uncertainty in its estimates of the objects' correspondences. Utilizing this framework, we propose a maximum likelihood method that generates actions to maximally reduce uncertainty, thereby minimizing the number of interactions required during the perception task. We evaluate our method with previously unseen pairs of pegs and holes in the real system, and demonstrate higher performance compared to the current state-of-the-art method that does not use multiple interactions.

1.3.3 Part III

Finally, in Part III, to address the question: *How can we solve a desired dexterous manipulation task?*, Chapter 6 tackles the problem of stably stacking a highly irregular object on top of an unknown tower. To solve this task, it is essential to estimate an *extrinsic* contact

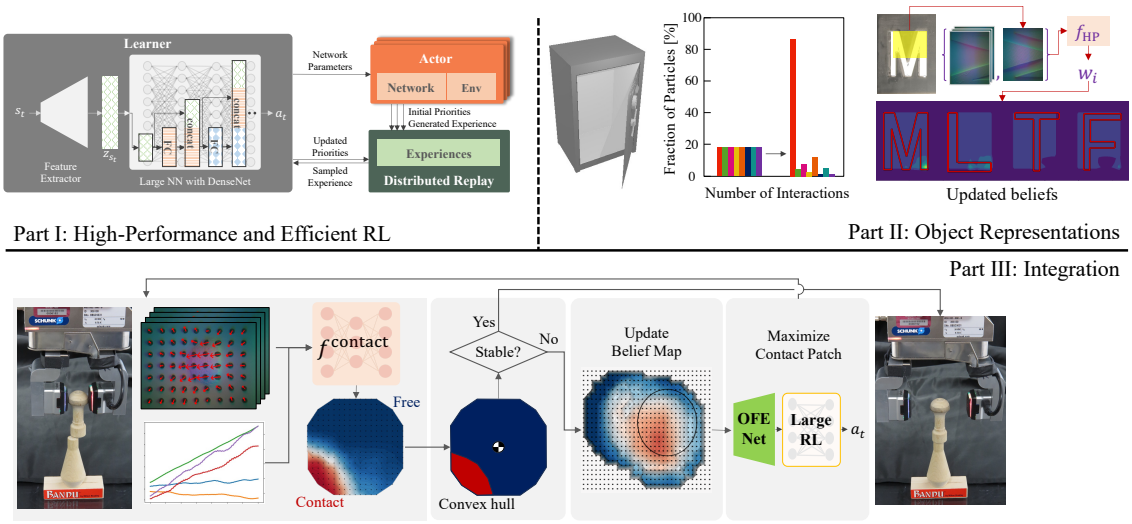


Figure 1.2: Summary of the thesis. Our thesis consists of three parts to answer the three questions posed in Section. 1.2. Part I focuses on developing an RL framework that allows us to train high-performant and efficient RL policies. The developed framework achieves approximately 2.0 times higher returns and 100 times better sample efficiency depending on RL algorithms and environments. Part II focuses on estimating object's representation that is essential to solve a task. We develop a probabilistic estimation framework that allows us to reduce uncertainty through interactions, and demonstrate it can solve the two tasks: articulated object manipulation and general connector part mating task using tactile sensing. Part III tackles a stable stacking task that requires to estimate contact patch between grasped object and its environment. We realize that by utilizing the probabilistic estimation framework developed in Part II, and input the estimated contact patch to the RL framework developed in Part I. This combination enables the robot to solve the stacking task with 100 % success rate, with only 1000 interactions to train the model.

patch, a contact area between the grasped object and the bottom object (or environment) that governs the stability of a configuration. However, since the extrinsic contact patch is not directly observable, robots have to estimate it from raw tactile signals, that is, F/T measurements and tactile images. To do this, we utilize the probabilistic estimation framework developed in Part II to reliably estimate the contact patch by aggregating information from multiple interactions. This estimated extrinsic contact patch is then input into the RL framework, developed in Part I, to learn a policy that guides the robot towards stable configurations. Through experiments in the real system, we demonstrate that this combination allows the robot to train a policy solely in the real system within 1000 interactions (approximately 3 hours).

1.4 Related Work

A substantial body of research is related to this thesis. To present a clear and unified perspective on this work, I will provide a concise overview of the primary areas of relevant literature. Throughout the discussion, I will indicate the chapters in which additional details can be found. The three main areas of related work for this thesis are reinforcement learning, interactive perception, and robotic manipulation that are closely related to each

part.

1.4.1 High-performance and efficient RL

In this thesis, we will develop an RL framework that achieves higher sample efficiency and control performance so that policies can be trained and solve challenging manipulation tasks in the real system. Our basic strategy is to make the networks bigger, following CV and NLP community, where larger networks make search space larger, resulting in convergence to better local optima [28, 89]. However, in deep RL, it is known that larger networks make training unstable [67].

The central problem of this instability is known as the Deadly Triad [177], where deep RL training becomes unstable when combined with *function approximation*, *bootstrapping*, and *off-policy training*. This is shown in Eq.(1.2), in which the gradient of the Q-function loss with the parameter θ is shown.

$$\nabla_{\theta_i} L(\theta_i) = \underbrace{\mathbb{E}_{s,a\sim\pi(\cdot);s'\sim\mathcal{D}}}_{\text{Off-policy Training}} [\underbrace{(r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) - Q\left(s, a; \theta_i\right))}_{\text{Bootstrapping}} \nabla_{\theta_i} Q\left(s, a; \theta_i\right)] \quad (1.2)$$

While the bootstrapping problem has been mitigated, for example, by employing the target network to fix the regression target [127], double Q-networks to mitigate overestimation problem [128], and multistep return to reduce bias from the learned Q-function [178], these are not directly related to the network size. Therefore, we specifically focus on the other two aspects: function approximation and off-policy training.

Regarding the function approximation problem, we make the network larger with two main strategies. First, in Chapter 2, we increase the dimensionality of input to RL algorithms by projecting the state space to a higher-dimensional feature space. The closest work to ours is Munk *et al.* [131], where they learn a compact representation, no greater than one-third of its input, by using a loss function to predict the next state and reward. In contrast, we deliberately obtain higher-dimentional features, while using the simpler loss function of predicting the next state. Second, in Chapter 3, we increase the network size of RL itself by employing sophisticated network architectures developed in the CV/NLP community [78, 182]. While some work has examined the effects of making networks larger [48, 177], we conduct a more extensive experiments to figure out *when* it fails by changing number of layers and units, *why* it fails by visualizing the loss surface [106] and measuring effective rank [9], and *what* can stabilize training by changing networks architectures.

Regarding the off-policy training problem, the fundamental issue is the distribution mismatch between the current policy and the data stored in the replay buffer ($\mathcal{D}$ in Eq. 1.2) [177]. Therefore, previous work tried to mitigate it by changing the ratio of exploration and update [46], and changing priorities when sampling data from the replay buffer [153, 177]. In Chapter 3, we specifically employ distributed learning, where exploration and update are decoupled. Although there is much work proposing different distributed RL [77, 90, 126, 166], we specifically use a modified version of Ape-X [77] to improve

efficiency in the use of GPUs.

1.4.2 Object's representation for manipulation

To efficiently manipulate objects, understanding their underlying structures is crucial. Consider, for example, the simple daily task of opening a door. There are many types of doors: hinged, sliding, vertical, and locked doors, each with varying joint poses, varioius sizes, diverse visual textures, or even transparency. Despite major progress in computer vision [85, 129, 197] and vision foundation models [120], understanding these structures from vision alone remains quite challenging. In visually ambiguous situations, or more generally, situations where only partial observations are available, it is important to aggregate information from multiple observations to estimate the states of the environment, such as the position and type of joint required for opening a door. In these situations, perception can be facilitated by interaction with the environment. These approaches are broadly called *Interactive Perception (IP)* [16], and we employ this for Chapter 4, 5, and 6 to estimate essential objects' representations for manipulation.

In Chapter 4, we estimate the joint configurations necessary for manipulating articulated objects. As previously discussed, vision alone struggles with estimation, particularly in ambiguous situations such as closed doors. To resolve this problem using IP, the existing literature presents offline estimation approaches that rely on fiducial markers [168] or markerless tracking [92, 145], as well as online approaches [122]. While these methods assume predefined actions, recent methods have also incorporated reasoning about actions to actively reduce uncertainty [11, 62, 142], though they still rely on fixed actions or simple settings with only one sliding joint [142]. In contrast, we introduce *H-SAUR*, a probabilistic estimation framework that initializes various hypothetical joint configurations and refines them through interactions with actions generated by a physics engine. Additionally, we introduce the *PuzzleBoxes* benchmark, which includes a set of locked doors with dependent joint configurations, and demonstrate that H-SAUR can effectively solve these challenges.

In Chapter 5, we estimate the correspondence of pegs and holes from tactile images, again, using IP, specifically applying the framework developed in Chapter 4. Pose estimation with a tactile sensor is a partially observable task because the sensor size is typically much smaller than the target objects. Tac2Pose [15] attempts to estimate the pose of objects from a single tactile image but fails due to ambiguity, i.e. distinguishable features often cannot be captured from single interaction due to the small sensor size. MidasTouch [170] addresses this problem by collecting multiple interactions by sliding objects on surfaces and using a particle filter to reduce uncertainty. However, the sliding actions are performed by human hands and their method requires thousands of interactions, which is impractical. Furthermore, these methods typically require 3D models and simulators to compute objects' correspondences. In contrast, our proposed method, *TactileFilter*, simultaneously estimates the object's class and pose without requiring 3D models of the objects and generates optimal actions that reduce uncertainty.

In Chapter 6, we address the problem of stacking highly irregular objects on top of unstable towers by estimating *extrinsic* contact patches – the contact patches between the grasped and the bottom objects. This is a partially observable task because the robot can

only observe tactile images of the grasped objects and force-torque data during interaction. Previous works represent contacts as sets of points [95, 121] and lines [96, 118]. While line contacts require active exploration involving changes in gripper, which is challenging in our setting, where the tower is extremely unstable. The closest work to ours is the neural contact fields (NCF) [73], where the authors estimate the contact patch between a grasped object and its environment. While NCF is evaluated in simulation and on a limited number of objects, we will test our method on unknown geometries of the environment, which can be applied to an appropriate downstream task in a real system, such as stable stacking.

1.4.3 Robotic Manipulation

Robotic manipulation has a long history, beginning with classical model-based approaches [114, 124] and evolving into more modern techniques [26, 76]. These approaches typically assume that the robot has access to complete information about the environment, including state and dynamics, often relying on motion capture systems such as Vicon or fiducial markers. Recent advances in CV techniques have alleviated this problem and allowed robots to perceive the pose of objects from vision sensors [119, 203]. However, achieving precise manipulation, such as inserting a peg into a hole with a tolerance of less than 0.1 millimeters, cannot be accomplished solely with vision sensors [139].

Humans rely on touch and tactile sensing for many dexterous manipulation tasks. Our tactile sensing provides extensive information about contact formations and geometric details of objects during interactions. For example, in Chapter 6, we address the problem of gently stacking two lightweight objects on top of each other without visual assistance (see Fig. 1.1). Humans can perform this task effortlessly by sensing fine tactile signals as they place and release the grasped piece. Inspired by these human capabilities, vision-based tactile sensors such as GelSight [199] and GelSlim [173] have been developed. These sensors capture the contact formation between the fingers and the grasped object as high-resolution images. Previous works have applied these sensors for insertion [14, 43], harness following [157], and tool manipulation [159].

When combined with RL, previous works have directly mapped raw tactile signals, such as RGB images or marker displacements, to robot actions for insertion [42], pivoting [169], and in-hand manipulation [180]. However, since contact formation is complex and very hard to interpret, mapping raw tactile signals to complex policies is likely to be sample-inefficient. In Chapter 6, we combine interactive perception and RL by estimating an effective intermediate representation of the task through interaction and inputting it into the RL model. We demonstrate that this combination allows robots to solve the task more efficiently than inputting raw tactile signals.

1.5 Outline

This thesis is divided into three parts.

Part I addresses the question of *How can we train high-control-performance and sample-efficient RL policies?*, by enlarging networks for training deep RL policies while mitigating potential issues that lead to instability during training. Chapter 2 increases the network

capacity by increasing input dimensionality. Chapter 3 introduces a framework that enables increasing the size of RL models.

Part II comprises two chapters and answers the question of *How can we estimate an effective object's representation for performing manipulation?*. Chapter 4 proposes a probabilistic estimation framework for estimating object articulation, and Chapter 5 applies this framework to a real robot equipped with tactile sensing to estimate mating pairs of connectors (pegs and holes).

Part III addresses the question of *How can we solve a desired dexterous manipulation task?*. In Chapter 6, we tackle the problem of stable placement of highly irregular objects by integrating the high-control-performance and sample-efficient RL framework from Part I with the probabilistic estimation framework from Part II.

Chapter 7 concludes the thesis, discussing key findings, limitations, and future research directions.

Part I

How can we train high-control-performance and sample-efficient RL policies?

Chapter 2

Can Increasing Input Dimensionality Improve Deep RL?

Deep reinforcement learning (RL) algorithms have achieved impressive success in various difficult tasks such as computer games [127] and robotic control [56]. Significant research effort in the field has led to the development of several successful RL algorithms [52, 60, 155, 156]. Their success is partly based on the expressive power of deep neural networks that enable the algorithms to learn complex tasks from raw sensory data. Whereas neural networks have the ability to automatically acquire task-specific representations from raw sensory data, learning representations usually requires a large amount of data. This is one of the reasons that the application of RL algorithms typically needs millions of steps of data collection. This limits the applicability of RL algorithms to real-world problems, especially problems in continuous control and robotics. This has driven tremendous research in the RL community to develop sample-efficient algorithms [21, 88, 101, 132].

In general, state representation learning (SRL) [105] focuses on representation learning where learned features are in low dimension, evolve over time, and are influenced by actions of an agent. Learning lower dimensional representation is motivated by the intuition that state of a system represents the *sufficient statistic* required to predict its future, and in general, *sufficient statistic* for a lot of physical systems is fairly small dimensional. In the SRL framework, the raw sensory data provided by the environment in which RL agents are deployed is called an *observation*, and its low-dimensional representation is called a *state*. Such a state variable is expected to have all task-relevant information, and ideally only such information. The representation is usually learned from *auxiliary tasks*, that enables the state variable to contain prior knowledge of the task domain [86] or the dynamics of the environment. In contrast to auxiliary tasks, the task that RL agents need to learn is ultimately called the actual task in this chapter.

Conventional wisdom suggests that the lower the dimensionality of the state vector, the faster and better RL algorithms will learn. This reasoning justifies various algorithms for learning compact state representations from high-dimensional observations, for example [188]. However, while probably correct, this reasoning likely applies to the intrinsic dimensionality of the state (the *sufficient statistic*). An interesting question is whether RL problems with an intrinsically low-dimensional state can benefit by intentionally increasing its dimensionality

using a neural network with good feature propagation. This paper explores this question empirically, using several representative RL tasks and state-of-the-art RL algorithms.

Additionally, we borrow motivation from the fact that larger networks generally allow better solutions as they increase the search space of possible solutions. The number of units in the hidden layers of multi-layer perceptrons (MLP) is often larger than the number of inputs, in order to improve the accuracy of function approximation. The importance of the size of the hidden layers has been investigated by a number of authors. Lu, *et al.* [115] theoretically shows that MLPs with a hidden layer smaller than the input dimension are very limited in their expressive power. In addition, in deep RL, neural networks often have hidden layers larger than the dimension of observations [48, 67]. Since the state representation is an intermediate variable of processing just like the hidden units, it is reasonable to expect that high-dimensional representations of state might improve the expressive power of the neural networks used in RL agents in their own right.

Based on this idea, we propose OFENet: an Online Feature Extractor Network that constructs and uses high-dimensional representations of observations and actions, which are learned in an online fashion (i.e., along with the RL policy). We use a neural network for OFENet to produce the representations. It is desirable that the neural network can be easily optimized and produce meaningful high-dimensional representations. To meet these requirements, we use MLP-DenseNet as the network architecture; it is a slightly modified version of a densely connected convolutional network [78]. The output of MLP-DenseNet is the concatenation of all layers' outputs. This network is trained with the incentive to preserve the *sufficient statistic* using an auxiliary task to predict future observations of the system. Consequently, the RL algorithm receives higher-dimensional features learned by OFENet that have *good* predictive power of future observations.

We believe that the representation trained with the auxiliary task allows our agent to learn an effective, higher-dimensional representation for input to the RL algorithm. This in turn allows the agent to learn complex policies more efficiently. We present results that empirically demonstrate that the representations produced by OFENet improve the performance of RL agents in non-image continuous control tasks. OFENet with several state-of-the-art RL algorithms [52, 60, 156] consistently achieves state-of-the-art performance in various tasks, without changing the original hyperparameters of the RL algorithm.

2.1 Related Work

Our work is broadly motivated by Munk *et al.* [131] that proposed to use the output of a neural network layer as the input for a deep actor-critic algorithm. Our method is also based on this general idea. However, a key difference is that their goal of representation learning is to learn compact representation from noisy observation, while we propose the idea of learning *good* higher-dimensional representations of state observations. For clarity of presentation, we describe the method in Munk *et al.* [131] in detail later.

While the classic reinforcement learning paradigm focuses on reward maximization, streaming observation contains an abundance of other possible learning targets [82]. These learning tasks are known as auxiliary tasks, and they generally accelerate acquisition of

useful state representations. There is a lot of literature on using different auxiliary tasks for different tasks [5, 10, 30, 71, 87, 105, 131, 179, 204]. In the proposed work, we use the auxiliary task of predicting the next observation for training the OFENet with the motivation that this allows the higher dimensional outputs of the OFENet to preserve the *sufficient statistic* for predicting future observations of the system. Thus, these representations can prove effective in learning meaningful policies too.

The network architecture of OFENet is based on the success of recently reported research on deep learning. In the advertisement domain, Wide & Deep model [31] and DeepFM [58] have an information flow that passes through deep networks, in order to utilize low-order feature interactions. Similarly, OFENet also has connections between shallow layers and the output to produce higher-dimensional representations. Contrary to ours, Rajeswaran *et al.* [149] enriches the representational capacity using random Fourier features of the observations. Our proposed method is not orthogonal to their approach: OFENet can be combined with any type of expanded input spaces, including RBF policy. That would further expand the search space to explore, and it might result in even better performance.

The approach most similar to our method has been proposed in Zhang *et al.* [204]. Their DDR ONLINE produces higher-dimensional representations than the original observations, similar to our method. However, our approach to constructing good representations is quite different. They used a recurrent architecture of the network in order to incorporate temporal dependencies, and it contained several auxiliary tasks to increase the number of training signals. In contrast, our method uses only a single auxiliary task, and we embed additional information, such as information about the action, into the representation instead of increasing the number of training signals. Consequently, our training algorithm is much simpler than that presented in Zhang *et al.* [204].

2.2 Background

In this section, we provide relevant background and introduce some notations that are used throughout the paper.

2.2.1 Reinforcement Learning

Reinforcement learning considers the setting of an RL agent interacting with an environment to learn a policy that decides the optimal action of the agent. The environment is modeled as a Markov decision process (MDP) defined by the tuple $(\mathcal{O}, \mathcal{A}, p, r)$, where $\mathcal{O}$ is the space of possible observation and $\mathcal{A}$ is the space of available actions. We assume that observations and actions are continuous. The unknown dynamics $p(o_{t+1}|o_t, a_t)$ represents the distribution of the next observation o_{t+1} given the current observation o_t and the current action a_t. The reward function $r(o_t, a_t)$ represents the reward r_{t+1} obtained for o_t and a_t. The goal of the RL agent is to acquire the policy $\pi : \mathcal{O} \to \mathcal{A}$ that maximizes the expected sum of rewards in an episode. Note that the word *state* represents the notion of the essential information in the observation in SRL, so we call the information provided by the environment *an observation*.

2.2.2 Soft Actor Critic

Soft Actor Critic (SAC) [60] is an off-policy actor-critic algorithm. In actor-critic algorithms, the critic learns the action-value function $Q_\pi(o_t, a_t)$, while the actor learns a policy $\pi(a_t|o_t)$ to frequently select a_t that has a high value in Q_π. The SAC policy gives the distribution of actions $\pi(a_t|o_t)$, and the action is sampled stochastically during training. SAC favors the policy that maximizes not only the expected sum of rewards, but also the expected entropy of the distribution $\mathcal{H}(\pi(a_t|o_t))$. Consequently, its objective function is defined as:

$$\tilde{J}(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(o_t,a_t)\sim\rho_\pi(o_t,a_t)}[r_{t+1} + \mathcal{H}(\pi(\cdot|o_t))],$$

where T is the final time step of an episode, and $\rho_\pi(o_t, a_t)$ represents the distribution of observation-action pairs given policy π. This objective function $\tilde{J}(\pi)$ gives RL agents the incentive to explore more widely, while giving up on less promising avenues.

In SAC, $Q_\pi(o_t, a_t)$ and $\pi(\cdot|o_t)$ are represented by neural networks whose parameters are optimized to maximize the objective function $\tilde{J}(\pi)$. In addition, the state value function $V(o_t)$ is introduced to stabilize training. The function $Q_\pi(o_t, a_t)$ is learned by $V(o_t)$. As a consequence, SAC has three neural networks: $Q(o_t, a_t), V(o_t)$ and $\pi(\cdot|o_t)$. Our objective is to replace the inputs of these networks with our proposed representations.

2.2.3 Model learning deep deterministic policy gradient

Munk *et al.* [131] proposed the Model Learning Deep Deterministic Policy Gradient (ML-DDPG) algorithm to learn representations of observations. They introduced a *model network*, which is trained to construct the observation representation z_{o_t}. The representation is used as the input in the DDPG algorithm [111]. The model network is a three-layer network, which produces z_{o_t} as an internal representation, and predicts the next observation representation $\hat{z}_{o_{t+1}}$ and the reward $\hat{r}_{t+1}$ from the observation o_t (Figure 2.1).

The model network is trained by minimizing the following loss function

$$L_m = ||z_{o_{t+1}} - \hat{z}_{o_{t+1}}||^2 + \lambda_m||r_{t+1} - \hat{r}_{t+1}||^2,$$

where λ_m represents the trade-off between predicting the reward and the next observation representation. The minimization is done with samples collected before the agent starts learning, and then the parameters of the model network are fixed during learning.

While we construct high-dimensional representations with OFENet, in the experiments of ML-DDPG, the dimension of the observation representation z_{o_t} was not greater than one third of the dimension of the observation o_t.

2.3 Method

In this section, we describe our proposed method for learning higher-dimensional state representations for training RL agents. In the standard reinforcement learning setting, an RL agent interacts with the environment over a number of discrete time steps. At any time

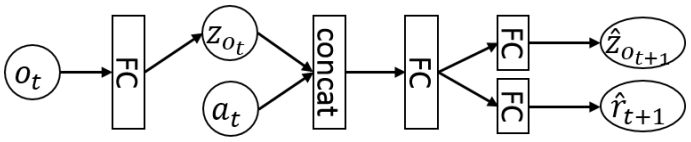


Figure 2.1: The model network of ML-DDPG. *FC* represents a fully-connected layer, and *concat* represents a concatenation of its inputs.

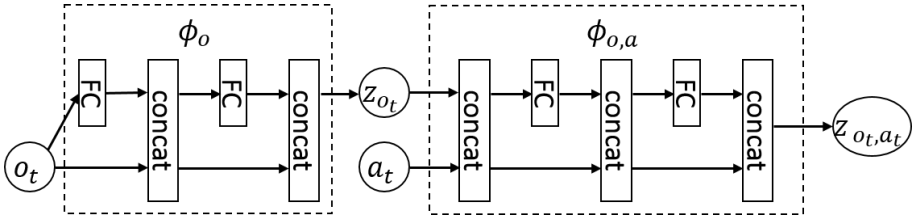


Figure 2.2: An example of the online feature extractor. **FC** represents a fully connected layer with an activation function, and **concat** represents a concatenation of the inputs.

t, the agent receives an observation o_t along with a reward r_t and emits an action a_t. During training, standard RL agents receive observation-action pair as input to learn the optimal policy. We propose the Online Feature Extractor Network (OFENet), which constructs high-dimensional representations for observation-action pair, which is then used by the RL algorithm as input to learn the policy (instead of the raw observation and action pair). The observation used by the components is replaced by the observation representation z_{o_t}, and the observation-action pair is also replaced by the observation-action representation z_{o_t,a_t}.

OFENet learns the mappings $z_{o_t} = \phi_o(o_t)$ and $z_{o_t,a_t} = \phi_{o,a}(o_t, a_t)$, which have parameters $\theta_{\phi_o}, \theta_{\phi_{o,a}}$ as depicted in Figure 2.2. To learn the mappings, we use an auxiliary task whose goal is to predict the next observation from the current observation and action. The learning of the auxiliary task is done concurrently with the learning of the actual task (and thus we call our proposed network as Online Feature Extraction Network, OFENet).

In the following, we describe in detail the auxiliary task, the neural network architecture for the mappings $\phi_o, \phi_{o,a}$, and how to select hyper-parameters for OFENet.

2.3.1 Auxiliary task

In this section, we incorporate *auxiliary tasks* to learn effective higher-dimensional representations for state and action that the RL agent will use. It is common knowledge that incorporating auxiliary tasks into the reinforcement learning framework can promote faster training, more robust learning, and ultimately higher performance for RL agents [44, 82].

We introduce the module f_{pred}, which receives the observation-action representation z_{o_t,a_t} as input to predict the next observation o_{t+1}. The module f_{pred} is represented as a linear combination of the representation z_{o_t,a_t}, which has parameters θ_{pred}.

Thus, effectively, along with learning the actual RL objective, we optimize the parameter set $\theta_{\text{aux}} = \{\theta_{\text{pred}}, \theta_{\phi_o}, \theta_{\phi_{o,a}}\}$ to minimize the auxiliary task loss defined as:

$$L_{\text{aux}} = \mathbb{E}_{(o_t,a_t)\sim p,\pi}[||f_{\text{pred}}(z_{o_t,a_t}) - o_{t+1}||^2] \tag{2.1}$$

Algorithm 1 Training of OFENet

Initialize parameters $\theta_{\text{aux}} = \{\theta_{\text{pred}}, \theta_{\phi_o}, \theta_{\phi_{o,a}}\}$
Initialize experience replay buffer $\mathcal{B}$

for each environment step **do**
  $a_t \sim \pi(a_t|o_t)$
  $o_{t+1} \sim p(o_{t+1}|o_t, a_t)$
  $\mathcal{B} \Leftarrow \mathcal{B} \cup \{(o_t, a_t, o_{t+1}, r_{t+1})\}$
  sampling mini-batch $\{o_{t,\mathcal{B}}, a_{t,\mathcal{B}}, o_{t+1,\mathcal{B}}\}$ from $\mathcal{B}$
  $\theta_{\text{aux}} \Leftarrow \theta_{\text{aux}} - \lambda_{\theta_{\text{aux}}} \nabla_{\theta_{\text{aux}}} L_{\text{aux}}$
  resampling mini-batch $\{o_{t,\mathcal{B}}, a_{t,\mathcal{B}}, o_{t+1,\mathcal{B}}\}$ from $\mathcal{B}$
  $z_o \Leftarrow \phi_o(o_{t,\mathcal{B}})$
  $z_{o,a} \Leftarrow \phi_{o,a}(z_o, a_{t,\mathcal{B}})$
  Update the agent (e.g., SAC) parameters with the representations $z_o, z_{o,a}$
end for

The auxiliary task defined by Equation (2.1) is used by the OFENet to learn higher-dimensional representations. This loss function incentivizes higher-dimensional representations that preserve the dynamics information of the system (or loosely preserve the sufficient statistic to predict future observations of the system). The expectation is that the RL agent can now learn much more complex policies using these higher-dimensional features as this effectively increases the search space for the parameters of the policies.

The transitions (o_t, a_t, o_{t+1}) required for learning are sampled as mini-batches $\{o_{t,\mathcal{B}}, a_{t,\mathcal{B}}, o_{t+1,\mathcal{B}}\}$ from the experience replay buffer $\mathcal{B}$, which stores the past transitions that the RL agent has received. Algorithm 1 outlines this procedure.

2.3.2 Network architecture

A neural network is used to represent the mappings $\phi_o, \phi_{o,a}$ in OFENet. As it is known that deeper networks have advantages with respect to optimization ability and expressiveness, we employ them in our network architecture. In addition to this, we also leverage the fact that observations often have intuitively useful information in non-image RL tasks. For example, when position and velocity of a robot are present in an observation, it is advantageous to include them in the representation when solving reaching tasks. Moreover, because the linear combination of position and velocity can approximate the position for the next time step, outputs of shallow layers are also expected to be physically meaningful.

To combine the advantages of deep layers and shallow layers, we use MLP-DenseNet, which is a slightly modified version of DenseNet [78], as the network architecture of OFENet. Each layer of MLP-DenseNet has an output y which is the concatenation of the input x and the product of a weight matrix W_1 and x defined as:

$$y = [x, \sigma(W_1 x)]$$

where $[x_1, x_2]$ means concatenation, σ is the activation function, and the biases are omitted to simplify notation. Since each layer's output is contained in the next layer's output, the raw input and the outputs of shallow layers are naturally contained in the final output.

The mappings $\phi_o, \phi_{o,a}$ are represented with an MLP-DenseNet. The mapping ϕ_o

receives the observation o_t as input, and the mapping $\phi_{o,a}$ receives the concatenation of the observation representation z_{o_t} and the action a_t as its input. Figure 2.2 shows an example of these mappings in the proposed OFENet.

RL algorithms take the learned representations z_{o_t} and z_{o_t,a_t} as input, and compute the optimal policy by optimizing the regular objective of maximizing the expected reward. It is important to note that these representations, however, are learned simultaneously with the RL algorithms. This might lead to change in distribution of the inputs to the RL algorithm. The RL algorithm, therefore, needs to adapt to the possible change of distribution in those base layers constructed by MLP-DenseNet. To alleviate this potential problem, we normalize the output of the base layer by using Batch Normalization [80] to suppress changes in input distributions.

2.3.3 Hyperparameter selection

Effective training of agents with OFENet requires selection of size of the hidden layers and the type of activation functions. In Henderson *et al.* [67], authors show that the size of the hidden layers and the type of activation function can greatly affect the performance of RL algorithms, and the combination that achieves the best performance depends strongly on the environment and the algorithm. Thus, ideally, we would like to choose the architecture of OFENet by measuring the performance on the actual RL task, but this would be very inefficient. Therefore, we use the performance on the auxiliary task as an indicator for selecting the architecture for each task.

In order to measure performance on the auxiliary task, first we collect transitions using a random action policy for the agent. The transitions are randomly split into a training set and a test set. Then, we train each architecture on the training set, and measure the average loss L_{aux} on the test set over five different random seeds. On the actual task, we use the architecture which achieves the minimum on the average auxiliary loss. We call this average loss the *auxiliary score*. Since we can reuse the transitions for each OFENet training, and do not have to train RL agents, this procedure is sample-efficient and doesn't incur much computational cost either.

We use an experience replay buffer to simulate the learning with the RL agent, where OFENet samples a mini-batch up to the Nth data item of the training set at the Nth step. The pseudocode for the proposed method is presented in Algorithm 1. It should be noted that we do not tune the hyperparameters of the RL algorithm in order to show that agents can learn effective policies by using the representations learned by the proposed method during the learning process. This allows more flexibility in training RL agents for a wide range of tasks.

2.4 Experiments

In this section, we try to answer the following questions with our experiments to describe the performance of OFENet.

- What is a good architecture that learns effective state and state-action representations

for training better RL agents?

- Can OFENet learn more sample efficient and better performant polices when compared to some of the state-of-the-art techniques?
- What leads to the performance gain obtained by OFENet?
- How does the dimensionality of OFENet representation affects performance?

In the rest of this section, we present experiments designed to answer the above questions. All these experiments are performed in the MuJoCo simulation environment.

2.4.1 Architecture comparison

We compare the auxiliary score defined in Section 2.3.3 with the performance on the actual task for various network architectures on the Walker2d-v2 task, where the dimensions of observations and actions are respectively 17 and 6.

First, we define the *actual score*, which is the metric of performance on the actual task. In this paper,

- *return* represents a cumulative reward over an episode;
- *step score* represents the average return over 10 episodes with the RL agent at each step;
- *actual score* represents the average of the step scores over latest 100,000 steps, where the step score is measured every 5,000 steps.

We measure the auxiliary score and the actual score for each network architecture. In this section, each architecture is characterized by a connectivity of architecture, number of layers, and an activation function. We compare three connectivity architectures: MLP-DenseNet defined in Section 2.3.2, standard MLP, and MLP-Resnet, which is a slightly modified version of ResNet [64]. MLP-ResNet has skip connections similar to the original one, and its layers have the output y defined as:

$$y = \sigma(W_2\sigma(W_1 x) + x) \tag{2.2}$$

where W_1, W_2 are weight matrices, x is the input, and σ is the activation function. The biases are omitted to simplify notation.

Each architecture has multiple options for the combination of a layer number and a hidden layer size. In this experiment, $\phi_o, \phi_{o,a}$ have the same layer number for each architecture. MLP has 1, 2, 3, or 4 layers for ϕ_o. MLP-ResNet and MLP-DenseNet have 2, 4, 6, or 8 layers for ϕ_o.

To find the most efficient architecture over same feature size, the dimensions of z_{o_t}, z_{o_t,a_t} are respectively fixed to 137 and 263. This means that the dimensionality increments of z_{o_t}, z_{o_t,a_t} from their inputs are 120. While the numbers of hidden units in $\phi_o, \phi_{o,a}$ are respectively 137 and 263 in MLP and MLP-Resnet, the number of hidden units in

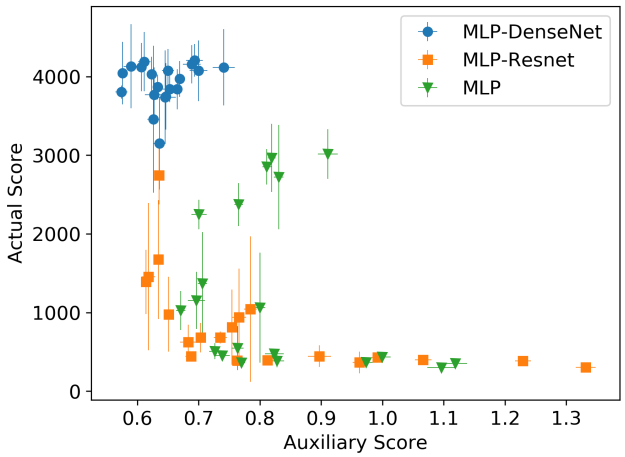


Figure 2.3: The actual scores and the auxiliary scores of various architectures in Walker2d-v2. The error bars represent the standard deviation of 5 trials. We can see a weak trend that the smaller the auxiliary score is, the better the actual score is.

MLP-DenseNet depends on the number of layers. All the layers in MLP-DenseNet have the same number of hidden units. For example, when ϕ_o has 4 layers, the number of hidden units is $120/4 = 30$.

Additionally, we compare the following activation functions: ReLU, tanh, Leaky ReLU, Swish [150] and SELU [98]. In total, we compare 3 connectivity architectures, 4 layer-size combinations, and 5 activation functions, resulting in a total of 60 network architectures.

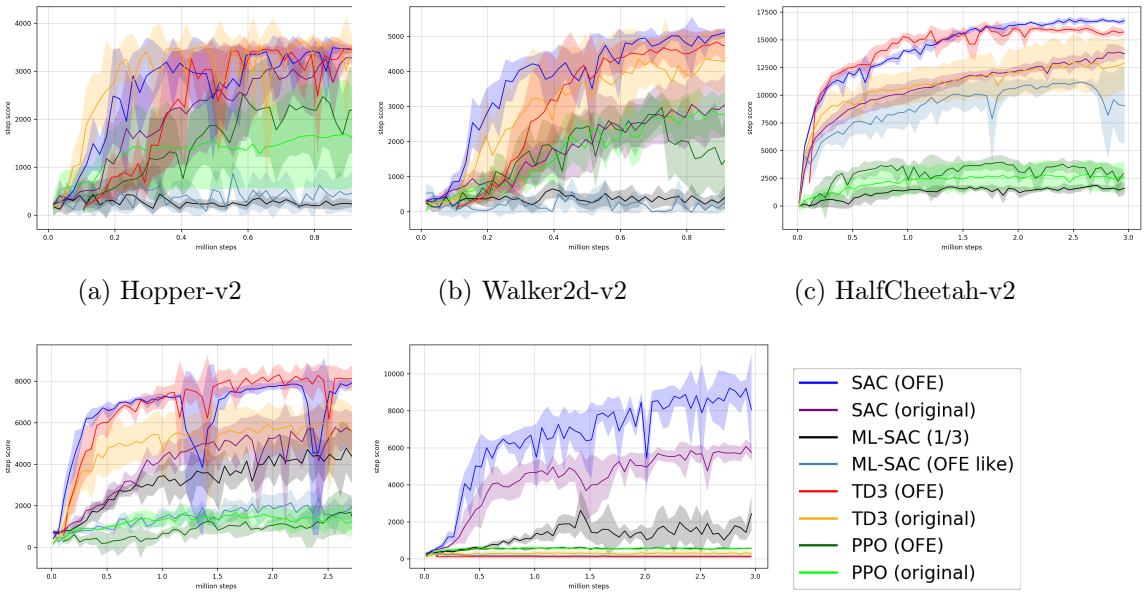
To measure the auxiliary score, we collect 100K transitions as a training set and 20K transitions as a test set, using a random policy. Each architecture is trained for 100K steps. To measure the actual score, each architecture is trained with the SAC agent for 500K steps with Algorithm 1. The SAC agent is trained with the hyper-parameters described in [60], where the networks have two hidden layers which have 256 units. All the networks are trained with mini-batches of size 256 and Adam optimizer, with a learning rate $3 \cdot 10^{-4}$.

Figure 2.3 shows the actual score and the auxiliary score of each network architecture in Walker2d-v2. The results show that DenseNet consistently achieves higher actual scores than other connectivity architectures. With respect to the auxiliary score, DenseNet also achieves better performance than others in many cases.

Overall, we can find a weak trend that the smaller the auxiliary scores, the better the actual scores. Therefore, in the following experiment, we select the network architecture that has the smallest value of the auxiliary score among the 20 DenseNet architectures for the actual task for each environment.

2.4.2 Comparative evaluation

To evaluate OFENet, we measure the performance of SAC, twin delayed deep deterministic policy gradient algorithm (TD3) [52] for off-policy RL algorithms, and proximal policy optimization (PPO) [156] for on-policy RL algorithm with OFENet representations and raw observations, on continuous control tasks in the MuJoCo environment. The dimensionality



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 2.4: Training curves on OpenAI Gym tasks. The solid lines represent average returns over five instances with different random seeds. The shaded region represents the standard deviation of the five instances. OFE outperforms original algorithm on both on-policy (PPO) and off-policy methods (SAC and PPO).

increments of z_{o_t}, z_{o_t,a_t} from their inputs are 240 in all experiments, and we select the best network architecture for each task as described in Section 2.3.3. The network architectures and optimizer, hyperparameters of SAC, TD3, PPO are the same as used in their original papers [52, 60, 156] even we combine them with OFENet. The mini-batch size in Fujimoto *et al.* [52], however, is different from original paper. We use mini-batches of size 256 instead of 100, similarly to SAC.

Moreover, we measure the performance of SAC with the representations which are produced by a model network of ML-DDPG, which we call *ML-SAC*. The hidden layer size of ML-SAC is 100, its activation function is ReLU, and λ_m in Section 2.2.3 is 10. We train the model network with an Adam optimizer, with a learning rate $1 \cdot 10^{-3}$. We set the dimension of the observation representation to one third of that of the observation according to Munk *et al.* [131]. In addition to this, we measure the performance of ML-SAC with the observation representation which has the same dimension as OFENet. Whereas in Munk *et al.* [131] the model network was trained with samples collected before the learning of the agent, we train the network with samples collected by the learning agent, such as OFENet.

In order to eliminate dependency on the initial parameters of the policy, we use a random policy to store transitions to the experiment replay buffer for the first 10K time steps for SAC and 100K time steps for TD3 and PPO as described in Fujimoto *et al.* [52]. We also pretrain OFENet to stabilize the input to each RL algorithm with these randomly trained samples. Note that as described in Section 2.3.1, OFENet predicts the future observation to learn the high-dimensional representations. In Ant-v2 and Humanoid-v2,

Table 2.1: The highest average returns over five different seeds for each environment. The bold number indicates the best score among each algorithm (SAC, TD3, and PPO). OFE outperforms original algorithm in most environments.

	SAC				TD3		PPO	
	OFE (ours)	Original	ML-SAC (1/3)	ML-SAC (OFE like)	OFE (ours)	Original	OFE (ours)	Original
Hopper-v2	**3511.6**	3316.6	750.5	868.7	3488.3	**3613.0**	**2525.6**	1753.5
Walker2d-v2	**5237.0**	3401.5	667.4	627.4	**4915.1**	4515.6	**3072.1**	3016.7
HalfCheetah-v2	**16964.1**	14116.1	1956.9	11345.5	**16259.5**	13319.9	**3981.8**	2860.4
Ant-v2	**8086.2**	5953.1	4950.9	2368.3	**8472.4**	6148.6	**1782.3**	1678.9
Humanoid-v2	**9560.5**	6092.6	3458.2	331.7	120.6	**345.2**	**670.3**	652.4

Table 2.2: The network architectures of OFENet for each MuJoCo task.

	Number of layers	Activation function
Hopper-v2	6	Swish
Walker2d-v2	6	Swish
HalfCheetah-v2	8	Swish
Ant-v2	6	Swish
Humanoid-v2	8	Swish

the observation contains the external forces to bodies, which are difficult to predict because of their discontinuity and sparsity. Thus, OFENet does not predict these external forces. Table. 2.2 shows the network architecture of the OFENet for each environment. As described in Section. 2.3.2, the selected network architecture is the one that receives the smallest value of the auxiliary score among 20 DenseNet architectures: the number of layers is selected from $\{2, 4, 6, 8\}$, and the activation function is selected from {ReLU, tanh, Leaky ReLU, Swish, SELU} while the network architecture of the RL algorithms { SAC, TD3, PPO } are the same with their original papers [52, 60, 156].

Figure 2.4 shows the learning curves of the methods, and Table 2.1 shows the highest average returns on five different seeds. SAC (OFE), i.e. SAC with OFENet representations, outperforms SAC (raw), i.e. SAC with raw observations. Especially in Walker2d-v2, Ant-v2, and Humanoid-v2, the sample efficiency and final performance of SAC (OFE) outperform significantly those of the original SAC. Since TD3 (OFE) and PPO (OFE) also outperform original algorithm, it can be concluded that OFENet is an effective method for improving deep RL algorithms on various benchmark tasks.

ML-SAC (1/3), i.e. ML-SAC with low dimensional representation performed poorly on all tasks. Since ML-DDPG is supposed to find compact representations from noisy observations, the model network probably could not find a compact representation from the non-redundant observations in the tasks. ML-SAC (OFE like), i.e., ML-SAC with the high dimensional representations, also performed poorly. In addition to this, extracting representation with MLP got much worse actual scores than MLP-DenseNet in Section 2.3.3. These show that constructing high dimensional representations is not a trivial task, and OFENet resolves this difficulty with MLP-DenseNet.

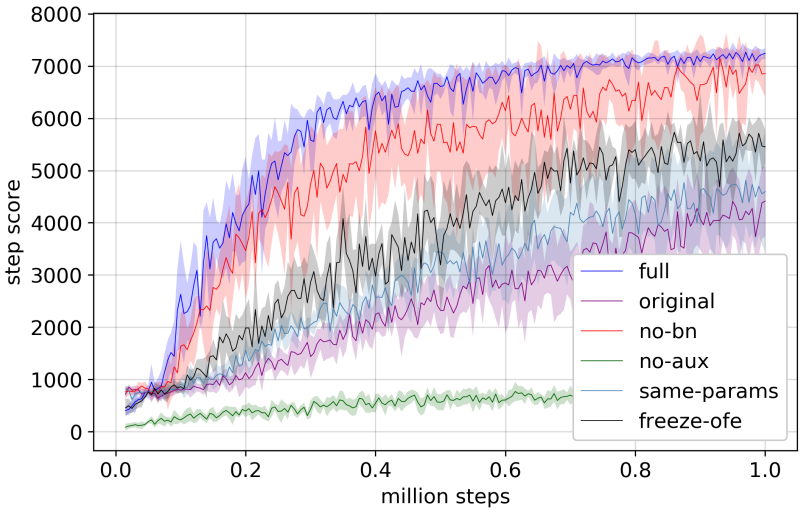


Figure 2.5: Training curves of the derived methods of SAC on Ant-v2. This shows that just increasing the input dimensionality of the input representation doesn't help in learning better policies. The higher-dimensional representations need to learned with the auxiliary task proposed in the paper.

2.4.3 Ablation study

Our hypothesis is that we can extract effective features from an auxiliary task in environments with a low-dimensional observation vectors. Furthermore, we would like to verify that simply increasing the dimensionality of the state representation will not help the agent learn better policies and that, in fact, generating *effective* higher-dimensional representations using the OFENet is required to get better performance. To verify this, we conducted an ablation study to show what components the improvement of OFENet comes from.

Figure 2.5 shows the ablation study over SAC with Ant-v2 environment. *full* and *original* are the same plots of SAC (OFE) and SAC (original) from Figure. 2.4.

no-bn removes Batch Normalization from OFENet. The bigger standard deviation of *no-bn* indicates that adding Batch Normalization stabilizes the full learning process. Since the OFENet is learned on-line, the distribution of the input to the RL algorithms changes during training. Batch Normalization effectively works to suppress this covariate shift during training and thus the learning curve of *full* is more stable than *no-bn* .

no-aux removes auxiliary task and train both OFENet and RL algorithms with actual task objective of reward maximization. In other words, they have to be learned by only scalar reward signal. The much lower scores of *no-aux* shows that learning the complex OFENet structure from just the reinforcement signal is difficult, and using the auxiliary task for learning good high-dimensional features enables better learning of control policy.

same-params increases the number of units of original SAC to $(401, 401)$, instead of $(256, 256)$ as suggested in Haarnoja *et al.* [60] so that it has the same number of parameters with our algorithm. The performance does increase compared to the original unit size, but its still not as good as the full algorithm in terms of both sample efficiency and

Table 2.3: The number of parameters of SAC with OFENet for Ant-v2 environment.

		Input units	Output units	Parameters
OFENet: z_o	1st layer	111	40	4640
	2nd layer	151	40	6240
	3rd layer	191	40	7840
	4th layer	231	40	9440
	5th layer	271	40	11040
	6th layer	311	40	12640
	Total	-	-	**51840**
SAC	1st layer	351	256	90112
	2nd layer	256	256	65792
	Output layer	256	8	2056
	Total	-	-	**157960**
SAC (OFE)	Total	-	-	**209800**

performance. This shows that just increasing the number of parameter does not help improve performance, but the auxiliary task helps with efficient exploration in the bigger parameter space. Table. 2.3 shows the number of parameters for SAC (OFE) used in the experiments. The number of parameters of *same-params* in Figure 2.5 matches the sum of the parameters for OFENet and the number of parameters of SAC. Note that the OFENet uses MLP-DenseNet architecture, and the output units of OFENet in Table. 2.3 ignores the units of previous layer.

As done in Munk *et al.* [131], *freeze-ofe* trains OFENet only before training of RL agent with randomly collected transitions as discussed in Section 2.4.1, and does not simultaneously train OFENet along with RL policy (i.e., skip line 5 and 6 in Algorithm.1). Since the accuracy of predicting future observation becomes worse when an RL agent explores unseen observation space, freezing the OFENet trained with only randomly collected data cannot produce good representations.

2.4.4 Effect of Dimensionality of Representation

In this section, we try to test whether increasing the dimension of the OFENet representation could lead to monotonic improvements in the performance of the RL agent. Figure 2.6 shows the improvement in the performance of an SAC agent on the HalfCheetah-v2 environment when we increase the dimension of the OFENet representation by increasing the numbers of hidden units in an 8-layer OFENet from 4 to 128. The step score of the RL agent generally increases with the increase of the dimensionality of representation, until a threshold is reached. This shows that we need sufficient output dimensionality to get the benefit of increasing the dimensionality of state representations using any feature extraction network.

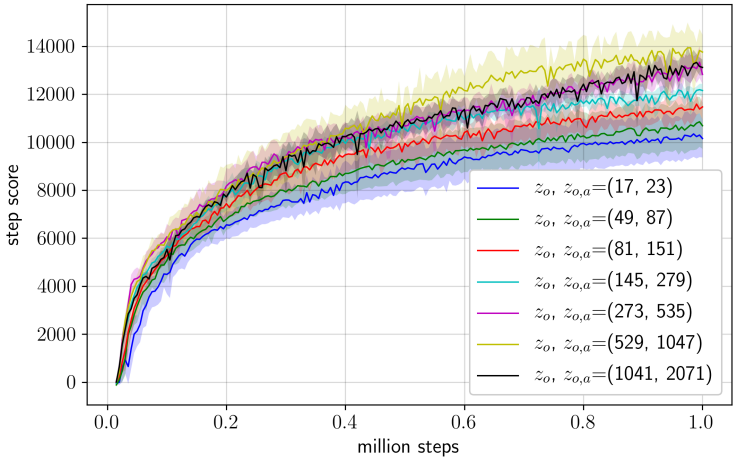


Figure 2.6: Comparison of various dimensional representations on HalfCheetah-v2. This shows that increasing the size of the OFENet representation generally helps to improve the policy performance.

2.5 Discussion

So, can increasing input dimensionality improve deep reinforcement learning? Recent success of deep learning has allowed us to design RL agents that can learn very sophisticated policies for solving very complex tasks. It is common belief that allowing smaller state representation helps in learning complex RL polices. In this paper, we wanted to challenge this hypothesis with the motivation that larger feature representations for state can allow bigger solution space and thus, can find better policies for RL agents. To demonstrate this, we presented an Online Feature Extractor Network (OFENet), a neural network that provides much higher-dimensional representation for originally low-dimensional input to accelerate the learning of RL agents. Contrary to popular belief we provide evidence suggesting that representations that are much higher-dimensional than the original observations can significantly accelerate learning of RL agents. However, it is important to note that the high-dimensional representations should be learned so as to retain some knowledge of the task or the system. In the current paper, it was learned using the auxiliary task of predicting the next observation. Our experimental evaluation demonstrated that OFENet representations can achieve state-of-the-art performance on various benchmark tasks involving difficult continuous control problems using both on-policy and off-policy algorithms. Our results suggest that RL tasks, where the observation is low-dimensional, can benefit from state representation learning. Additionally, the feature learning by OFENet does not require tuning the hyper-parameters of the underlying RL algorithm. This allows flexible design of RL agents where the feature learning is separated from policy learning.

Chapter 3

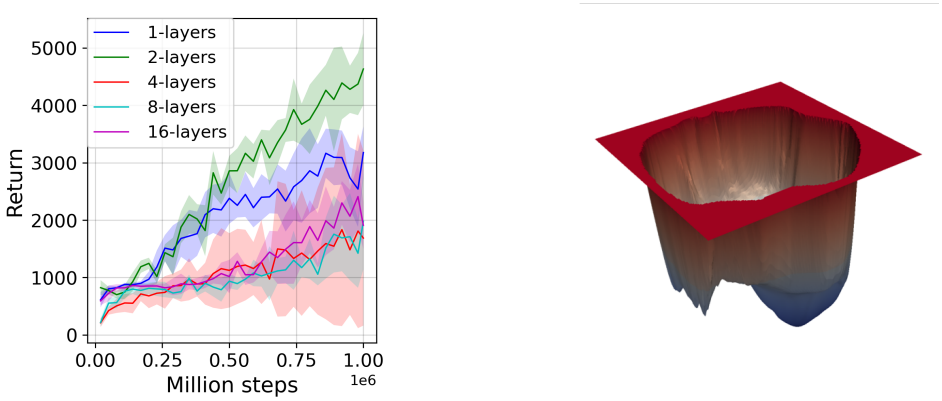
A Framework for Training Larger Networks for Deep RL

We have witnessed considerable improvements in the fields of computer vision (CV) [28, 64, 78, 99], natural language processing (NLP) [20, 38, 89, 207], and robotics [2, 19] in the last decade. These developments could be largely attributed to the training of very large neural networks with millions (or even billions or trillions) of parameters that can be trained using huge amounts of data and an appropriate optimization technique to stabilize training [133]. In general, the motivation for training larger networks comes from the intuition that larger networks allow better solutions as they increase the search space of possible solutions. Having said that, neural network training largely relies on finding good minimizers of highly non-convex loss functions. These loss functions are also governed by the choices of network architecture, batch size, etc. This has also driven a lot of research in these communities towards understanding the underlying reasoning for performance gains [106, 115, 134, 205]

In striking contrast, the Deep Reinforcement Learning (DRL) community has not reported a similar trend with regard to training larger networks for RL. Some studies have reported that deep RL agents experience instability while training with larger networks [1, 67, 164, 177]. As an example, in Fig. 3.1, we show the results of the Soft Actor Critic (SAC) [60] agent that uses Multi-layered Perceptron (MLP) for function approximation with the increasing number of layers while fixing its unit size to 256 (also notice the loss surface). These plots show that the use of deeper networks naively leads to poor performance for a deep RL agent. Consequently, using larger networks to train deep RL networks is not fully understood and thus is limiting in several ways. As a result, most of the reported work in the literature ends up using similar hyperparameters such as network structure, number, and size of layers.

Our work is motivated by this limitation. We explore the interplay between the size, structure, training, and performance of deep RL agents to provide some intuition and guidelines for using larger networks.

We present a large-scale study and provide empirical evidence for the use of larger networks to train DRL agents. First, we highlight the challenges that one might encounter when using larger networks to train deep RL agents. To circumvent these problems, we integrate a three-fold approach: decoupling feature representation from RL to efficiently



(a) Average return.

(b) Loss surface.

Figure 3.1: Training curves of SAC agents with the different numbers of layers while fixing the unit size (256) on Ant-v2 environment, and the loss function surface [106] of the deepest (16-layers) Q-network. The training curves suggest that simply building a deeper MLP with a fixed number of units does not improve the performance of DRL while building a larger network is generally effective in supervised learning. The loss surface shows that deeper networks have a more complex loss surface that could be susceptible to the choice of hyperparameters [106]. Motivated by this, we conduct an extensive study on how to train larger networks that contribute to performance gain for RL agents.

produce high-dimensional features (also known as OFENet, developed in Chapter 2), employing DenseNet architecture to propagate richer information, and using distributed training methods to collect more on-policy transitions to reduce overfitting. Our method is a novel architecture that combines these three elements, and we demonstrate that our proposed method significantly improves the performance of RL agents in continuous control tasks. We also conduct an ablation study to show which component contributes to the performance gain. In this paper, we consider learning from state vectors, i.e., not from high-dimensional observations, such as images.

Our contributions in this chapter can be summarized as follows:

- We conduct a large-scale study on employing larger networks for DRL agents and empirically show that, contrary to deeper networks, wider networks can improve performance.
- We propose a novel framework that synergistically combines recently proposed techniques to stabilize training: decoupling representation learning from RL, DenseNet architecture, and distributed training. Although each of these components has previously been proposed, the combination is novel, and we demonstrate that it significantly improves performance.
- We analyze the performance gain of our method using metrics of effective ranks of features and visualization of the loss function landscape of RL agents.

3.1 Related Work

Our work is motivated by Henderson et al. [67] which empirically demonstrates that DRL algorithms are vulnerable to different training choices like architectures, hyperparameters, activation functions, etc. The paper compares performance on the different numbers of units and layers and demonstrates that larger networks do not consistently improve performance. This is contrary to our intuition considering recent progress in solving computer vision tasks such as ImageNet [37]: larger and more complex network architectures have proven to achieve better performance [64, 78, 99, 171].

Hasselt *et al.* [177] identify a *deadly triad* of *function approximation*, *bootstrapping*, and *off-policy learning*. When these three properties are combined, learning can be unstable and potentially diverge, with value estimates becoming unbounded. Several previous works have attempted to address this issue by implementing various techniques, such as target networks [128], double Q-learning [178], and n-step learning [72]. Our challenge of training larger networks is specifically related to function approximation. However, as the deadly triad is entangled in a complex manner, we also have to deal with other problems. Regarding network size, some studies investigate the effect of making the network larger for continuous control tasks using MLP [1, 48] and concluded that larger networks tend to perform better, but also become unstable and prone to diverge more. In a related investigation, Hasselt *et al.* [177] employed CNNs to approximate functions for Atari games, which also revealed the instability that arises when expanding networks based on Deep Q Networks [128]. The study also found that training stability can sometimes be achieved through the use of Double Q-Networks, although it was not consistent with the size of the networks. Similar studies on on-policy methods are performed in Andrychowi *et al.* [6] and Liu *et al.* [113], showing that too small or large networks could cause a significant drop in policy performance. Although these studies are limited to relatively small sizes (hundreds of units with several layers), we will have a more thorough study on much larger networks, specifically focusing on learning from state vectors.

To build a large network, unsupervised learning has been used to learn powerful representations for downstream tasks in natural language processing [38, 148] and computer vision [27, 63]. In the context of RL, auxiliary tasks, such as predicting the next state conditioned on the past state(s) and action(s) have been widely studied to improve the sample efficiency of RL algorithms [59, 82, 109, 158]. Researchers have generally focused on learning a good representation of the state input setting that produces low-dimensional features [105, 131]. In contrast to this, in Chapter 2, we proposed the use of an online feature extractor network (OFENet) that intentionally increases input dimensionality and demonstrates that a larger feature size enables improving RL performance in both sample efficiency and control performance. We leverage this idea and use larger input (or feature) for RL agents, as well as larger networks for the policy and value function networks.

One can also use the AutoRL approaches [143] that dynamically adjust hyperparameters during training to build large networks. For example, Wan *et al.* [184] employ a combination of the population-based method [81] and the Bayesian optimization framework to optimize hyperparameters, including network architecture, to maximize returns during training.

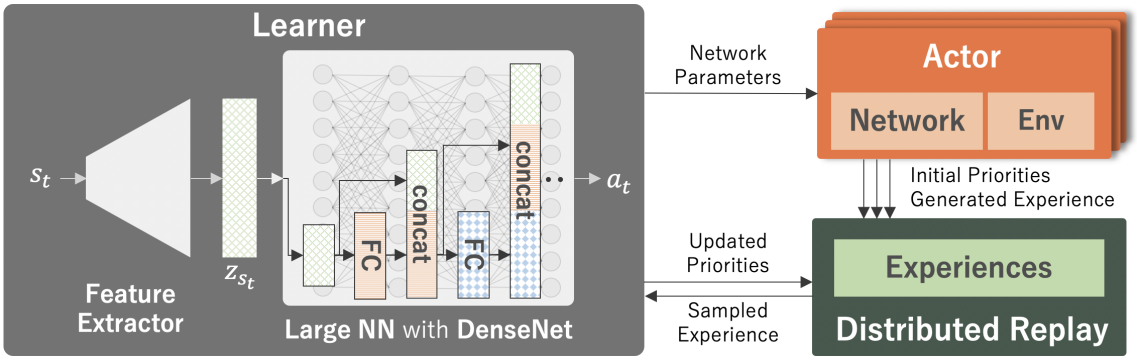


Figure 3.2: Proposed framework to train larger networks for deep RL agents. We combine three elements. First, we decouple the representation learning from RL to extract an informative feature z_{s_t} from the current state s_t using a feature extractor network that is trained using an auxiliary task to predict the next state s_{t+1}. Second, we use large networks using DenseNet architecture, which allows for stronger feature propagation. Finally, we employ the Ape-X-like distributed training framework to mitigate the overfitting problems that tend to happen in larger networks and enable to collect more on-policy data, i.e., the distribution of the data for updating the policy is closer to the data collected by the current policy, which can improve performance. FC refers to a fully-connected layer.

Mohan *et al.* [130] empirically demonstrate that hyperparameter landscapes vary over time during training, which requires AutoRL algorithms to adjust hyperparameters dynamically. Although this work focuses on proposing a framework to train large networks while improving the performance of RL algorithms, the proposed framework can also be easily combined with these AutoRL approaches.

3.2 Method

While recent studies suggest that larger networks for DRL agents have the potential to improve performance, it is non-trivial to alleviate some potential issues that lead to instability when using larger networks to train RL agents.

Our method is based on three key ideas: (1) decoupling representation learning from RL, (2) allowing better feature propagation using good network architectures, and (3) using huge amounts of more on-policy data using distributed training to avoid overfitting in larger networks. We first obtain good features apart from RL using an auxiliary task and then propagate the features more efficiently by employing the DenseNet [78] architecture. Additionally, we use a distributed RL framework that can mitigate the potential overfitting problem. In the following, we describe in detail the three elements that we use to train larger networks for deep RL agents. Our proposed approach is shown schematically in Fig. 3.2.

3.2.1 Decoupling Representation Learning from RL

While the simplicity of learning the entire pipeline in an end-to-end fashion is appealing, updating all parameters of a large network using only a scalar reward signal can result in very inefficient training [167]. Decoupling unsupervised pretraining from downstream tasks is common in computer vision [63, 66] and has proven to be very efficient. Taking

inspiration from this, we adopt the online feature extractor network (OFENet) proposed in Chapter 2 to learn meaningful features separately from RL training.

As we have seen in Chapter 2, OFENet learns the representation vectors of states z_{s_t} and state-action pairs z_{s_t,a_t}, and provides them to the agent instead of the original inputs s_t and a_t, delivering significant performance improvements in continuous robot control tasks[1]. As the representation vectors z_{s_t} and z_{s_t,a_t} are designed to have much higher dimensionality than the original input, OFENet matches the philosophy of providing a larger solution space that allows us to find a better policy. The representations can be obtained by learning the mappings $z_{s_t} = \phi_s(s_t)$ and $z_{s_t,a_t} = \phi_{s,a}(s_t, a_t)$. The ϕ_s and $\phi_{s,a}$ are neural networks with arbitrary architecture that have parameters $\theta_{\phi_s}, \theta_{\phi_{s,a}}$, and trained by minimizing an auxiliary task of predicting the next state s_{t+1} from the current state and action representation z_{s_t,a_t} as:

$$L_{\text{aux}} = \mathbb{E}_{(s_t,a_t)\sim p,\pi}\left[\|f_{\text{pred}}(z_{s_t,a_t}) - s_{t+1}\|^2\right], \tag{3.1}$$

where f_{pred} is represented as a linear combination of the representation z_{s_t,a_t}. The learning of the auxiliary task is done concurrently with the learning of the downstream RL task. Our experiments allow the input dimensionality to be much larger than previously presented in Chapter 2; we explore maximum 2048 dimentional representations compared to 256 in Chapter 2.

3.2.2 Distributed Training

In general, larger networks need more data to improve the accuracy of the function approximation [37, 70]. MLP with a large number of hidden layers is particularly known to cause an overfitting of training data, often resulting in inferior performance to shallow networks [151]. In the context of RL, while we train and evaluate in the same environment, there is still a problem of overfitting: the agent is only trained on limited trajectories it has experienced, which cannot cover the entire state-action space of the environment [113]. Fu *et al.* [48] showed that the overfitting to the experience replay does exist. To mitigate this overfitting problem, Fedus *et al.* [46] empirically showed that having more policy data in the replay buffer, i.e., collecting more than one transition while updating the policy one time, can improve the performance of the RL agent. However, it will be extremely slow.

In light of these studies, we employ the distributed RL framework, which leverages distributed training architectures that decouple learning from collecting transitions by utilizing many actors running in parallel on different environment instances [77, 90]. In particular, we use the Ape-X [77] framework, where a single learner receives experiences from distributed prioritized replay [153], and multiple actors collect transitions in parallel (see Fig. 3.2). This helps to increase the number of data close to the current policy, that is, more data on the policy, which can improve the performance of off-policy RL agents [46] and mitigate rank collapse problems in Q networks [9]. It is noted that one can collect

[1]While we denote state and state space as o_t and $\mathcal{O}$ to strictly follow the notation used in state representation learning [105] in Chapter 2, we use s_t and $\mathcal{S}$ to represent the state and state space in this chapter.

more on-policy data by collecting more than one transition at each policy update iteration while being much slower, as shown in Fedus *et al.* [46]. Furthermore, while we employ a distributed training framework proposed in Horgan *et al.* [77], we do not use the RL algorithm used there, but instead use standard off-policy RL algorithms: SAC [60] and the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [52] in our experiments. Furthermore, it should be noted that our approach uses a distributed training framework as suggested by Horgan *et al.* [77]. However, the RL algorithms we evaluate differ from theirs; we employ two widely used off-policy RL algorithms, namely SAC [60] and TD3 [52] in our experimental design.

3.2.3 Network Architectures

Tremendous developments have been made in the computer vision community in designing sophisticated architectures that enable training of very large networks by making the gradients more well-behaved, such as skip connections and batch normalization [64, 78, 80, 171]. We focus specifically on the use of the Dense Convolutional Network (DenseNet) architecture, which alleviates the problem of vanishing gradient, strengthens feature propagation, and reduces the number of parameters [78]. DenseNet has a skip connection that directly connects each layer to all subsequent layers as $y_i = f_i^{\text{dense}}([y_0, y_1, ..., y_{i-1}])$, where y_i is the output of the i^{th} layer; thus all the inputs are concatenated into a single tensor. Here, f_i^{dense} is a composite function that consists of a sequence of convolutions, Batch Normalization (BN) [80], and an activation function. An advantage of DenseNet is its improved flow of information and gradients throughout the network, making large networks easier to train. We borrow this architecture to train large networks for RL agents.

Although there are existing examples of applying the DenseNet architecture to Deep Reinforcement Learning (DRL) agents, the full potential of this approach has yet to be fully investigated. In this regard, Sinha *et al.* [164] proposed a novel modification to the DenseNet architecture, wherein the state or the state-action pair is concatenated to each hidden layer of the multi-layer perceptron networks (MLP), except the final linear layer, in their D2RL algorithm. In contrast to the modified version proposed by Sinha *et al.* [164], the *MLP-DenseNet* we proposed in Section 2.3.2 strictly adhered to the original DenseNet architecture, utilizing the dense connection that concatenates all previous layer outputs for OFENet training. We use this original DenseNet architecture to represent the policy and value function networks. The schematic of the DenseNet architecture is also shown in Fig. 3.2.

3.3 Experimental Settings

In this section, we summarize the settings we use for our experiments. We run each experiment independently with five random seeds. Average and ± 1 standard deviation results will be reported, which are solid lines and shaded regions when we show training curves. The horizontal axis of a training curve is the number of gradient steps, which is not identical to the number of steps an agent interacts with an environment only when we use the distributed training.

3.3.1 Metrics

We evaluate the experimental results on two metrics: average return and recently proposed *effective ranks* [9] of the features matrices of Q-networks. Kumar *et al.* [9] showed that MLPs used to approximate policy and value functions that use bootstrapping lead to a reduction in the effective rank of the feature, and this rank collapse for the feature matrix results in poorer performance. The effective rank can be computed as $\mathrm{srank}_\delta(\Phi) = \min \left\{ k : \frac{\sum_{i=1}^{k} \sigma_i(\Phi)}{\sum_{i=1}^{d} \sigma_i(\Phi)} \geq 1 - \delta \right\}$, where $\sigma_i(\Phi)$ are the singular values of the feature matrix Φ, which are the features of the penultimate layer of the Q-networks. We used δ to calculate the number of effective ranks in the experiments, as in Kumar *et al.* [9].

3.3.2 Implementations

RL agents We use the same hyperparameters as the original RL algorithms (SAC [60] and TD3 [52]). We conduct experiments on five different random seeds and report the average and standard deviation scores.

OFENet Regarding the parameters of OFENet, we also follow the implementation used in experiments in Chapter 2, that is, all OFENet networks that we used for our experiments consist of 8-layers DenseNet architectures with Swish activation [150]. We also used target networks [128] to stabilize OFENet training, since the distribution of experiences stored in the shared replay buffer can change more dynamically utilizing the distributed training setting as described in Sec. 3.2.2. The target networks are updated at each training step by slowly tracking the learned networks: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, where we assume that θ are the network parameters of the current OFENet and θ' are the target network parameters. We use the target smoothing coefficient $\tau = 0.005$, which is the same as the one used to update the target value networks in SAC [60], in other words, we do not tune this parameter.

Distributed training The distributed training setting we used is similar to Stooke *et al.* [166], which collects experiences using N^{core} cores on which each core contains N^{env} environments. Specifically, we used $N^{\mathrm{core}} = 2$ and $N^{\mathrm{env}} = 32$. Figure 3.3 shows the schematic of the distributed training. Since the actions are computed by the latest parameters, the collected experiences result in more on-policy data.

3.3.3 Visualizing loss surface of Q-function networks

Li *et al.* [106] proposed a method to visualize the loss function curvature by introducing *filter normalization* method. The authors empirically demonstrated that the non-convexity of the loss functions could be problematic, and the sharpness of the loss surface correlates well with test error and generalization error. In light of this, we also visualize the loss surface of the networks to specifically figure out why the deeper network could not lead to better performance, while the wider networks result in high-performance policies (Fig. 3.5).

To visualize the loss surface of our Q-networks, we use the authors' implementation[2]

[2]Code used for these plots can be found at `https://github.com/tomgoldstein/loss-landscape`

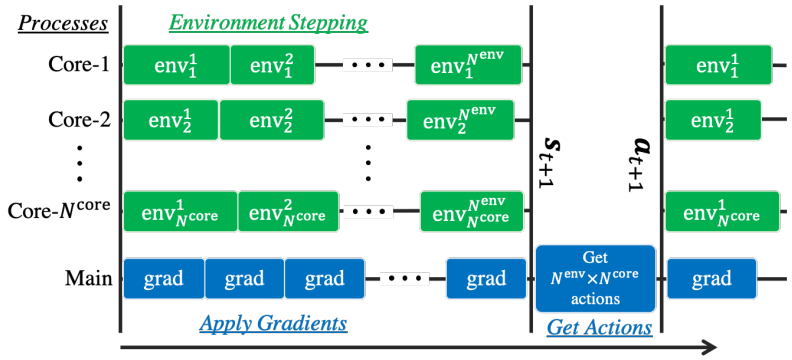


Figure 3.3: Schematic of asynchronous training. We use $N^{\text{core}} = 2$ cores for collecting experiences, where each core has $N^{\text{env}} = 32$ environments. Since the network parameters are shared, and the training and collecting transitions are decoupled, the collected experiences result in more on-policy data compared to the standard off-policy training, where the agent collects one transition while it applies one gradient step.

with the loss of:

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta \left(\mathbf{s}_t, \mathbf{a}_t \right) - \hat{Q} \left(\mathbf{s}_t, \mathbf{a}_t \right) \right)^2 \right], \tag{3.2}$$

with

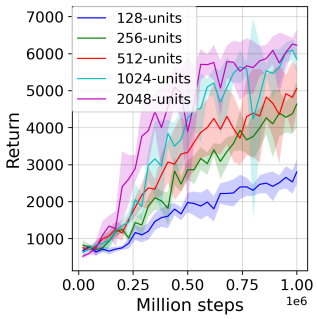
$$\hat{Q} \left(\mathbf{s}_t, \mathbf{a}_t \right) = r \left(\mathbf{s}_t, \mathbf{a}_t \right) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V_{\bar{\psi}} \left(\mathbf{s}_{t+1} \right) \right], \tag{3.3}$$

in which we exactly follow the notations used by SAC paper [60]. To compute this objective $J_Q(\theta)$, we collect all the transitions used in the training of deeper and wider networks and compute the target values of $\hat{Q} \left(\mathbf{s}_t, \mathbf{a}_t \right)$ after the training has been completed and store the tuples of $\left(s_t, a_t, \hat{Q} \left(\mathbf{s}_t, \mathbf{a}_t \right) \right)$ for all transitions in the training. Then, we use the authors' implementation to visualize the loss with the stored transitions and trained weights of the Q-network. Please refer to Li *et al.* [106] for more details.

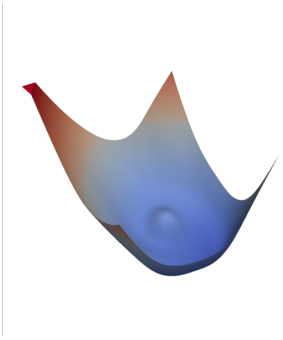
3.4 Experimental Results

In this section, we present the results of numerical experiments in order to answer some relevant underlying questions posed in this paper. In particular, we answer the following questions.

- Can RL agents benefit from the usage of larger networks during training? More concretely, can using larger networks lead to better policies for DRL agents?
- What characterizes a *good* architecture that facilitates better performance when using larger networks?
- Can our method work across different RL algorithms as well as different tasks, including sparse reward settings?
- How does the proposed framework perform in terms of wall clock time and sample efficiency?



(a) Average return.



(b) Loss surface.

Figure 3.4: Training curves of SAC agent with different number of units on Ant-v2 environment and the loss function surface of the widest (2048-units) Q-network. This shows that performance improves consistently when using wider MLPs.

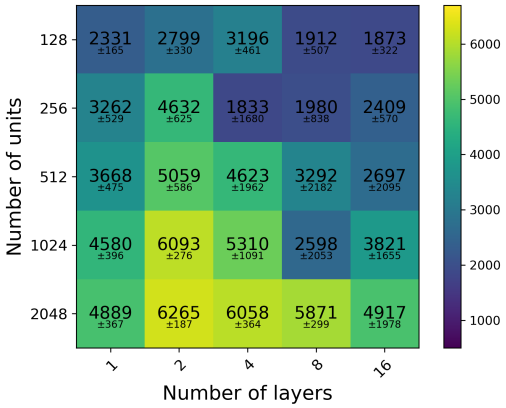


Figure 3.5: Grid search results of maximum average return at one-million training steps over the different number of units and layers for SAC agent on Ant-v2 environment. This demonstrates that a deeper MLP (see horizontally) does not consistently improve performance, while a wider MLP (see vertically) generally does.

3.4.1 Impact of network size on performance

In the first set of experiments, we try to investigate whether increasing the size of the network always leads to poor performance. We quantitatively measure the effectiveness of increasing the network size by changing the number of units N^{unit} and layers N^{layer}, while the other parameters are fixed.

Figure 3.1a shows the training curves when increasing the number of layers while the unit size is fixed to $N^{\text{unit}} = 256$. As we described in the beginning of this chapter, we observe that the performance gets worse as the network becomes deeper. In Fig. 3.4a, we show the effect of increasing the number of units while the number of layers is fixed to $N^{\text{layer}} = 2$. Contrary to the results when making the network deeper, we can observe a consistent improvement when making the network wider. To investigate more thoroughly, we also conduct a grid search, where we sample each parameter of the network from $N^{\text{unit}} \in \{128, 256, 512, 1024, 2048\}$, and $N^{\text{layer}} \in \{1, 2, 4, 8, 16\}$ and evaluate performance in Fig. 3.5. We can see a monotonic improvement in performance when we widen networks at almost all depths of the network.

(a) Wider network ($N^{\text{layer}} = 2, N^{\text{unit}} = 2048$)

(b) Deeper network ($N^{\text{layer}} = 16, N^{\text{unit}} = 256$)

Figure 3.6: Loss landscapes of models trained on HalfCheetah-v2 with one million steps, visualized using the technique in [106].

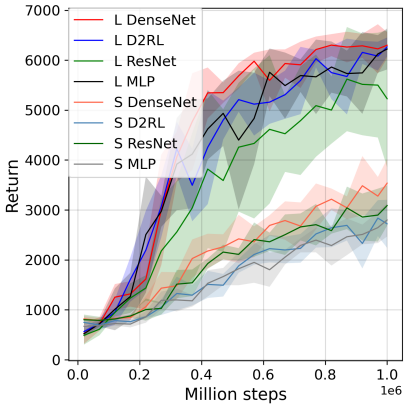
This result is in line with the general belief that training deeper networks is, in general, more difficult and more susceptible to the choice of hyperparameters [151]. This could be attributed to the initialization of the networks – it is known that the wider a layer, the closer the network is to the ideal conditions under which the initialization scheme was derived [55]. To understand why the deeper network is harder than the wider networks, we investigate the loss surface curvatures [106] of both deeper and wider networks. We show the loss surface of the deeper network ($N^{\text{layer}} = 16, N^{\text{unit}} = 256$) in Fig. 3.1b and the wider network ($N^{\text{layer}} = 2, N^{\text{unit}} = 2048$) of the SAC agent trained in the Ant environment in Fig. 3.4b as well as HalfCheetah-v2 environment in Fig. 3.6 using the visualization method proposed in Li *et al.* [106] with the loss of TD error of Q-functions of SAC agents. These figures show that wider networks have a nearly convex surface while deeper networks have more complex loss surface, which could be susceptible to the choice of hyperparameters [106]. Comparison of deeper and wider networks has also been done in several works [64, 106, 134, 191, 201], where wider networks are prone to have more generalization capability due to their smooth loss functions.

From these results, we observe and conclude that larger networks can be effective in improving deep RL performance. In particular, we achieve consistent performance gains when we widen individual layers rather than going deeper. Consequently, we fix the number of layers to $N^{\text{layer}} = 2$, and only change the number of units to learn larger networks in the following experiments.

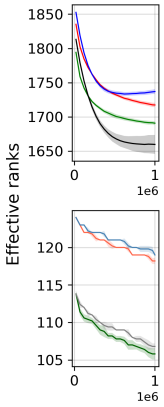
3.4.2 Architecture Comparison

In the next set of experiments, we try to investigate the role of a synergistic combination of connectivity architecture, state representation, and distributed training to allow the usage of larger networks for training deep RL agents. A brief introduction to these techniques is described in Sec. 3.2.

Connectivity architecture We first compare four connectivity architectures: standard MLP, MLP-ResNet, MLP-DenseNet, and MLP-D2RL, which is a recently proposed architec-



(a) Average return.



(b) Effective ranks.

Figure 3.7: Comparison of connectivity architecture on Ant-v2. Our proposed DenseNet architecture produces the best return on both large ($N^{\text{unit}} = 2048$, denoted by L) and small ($N^{\text{unit}} = 128$, S) networks while mitigating rank collapse as good as MLP-D2RL.

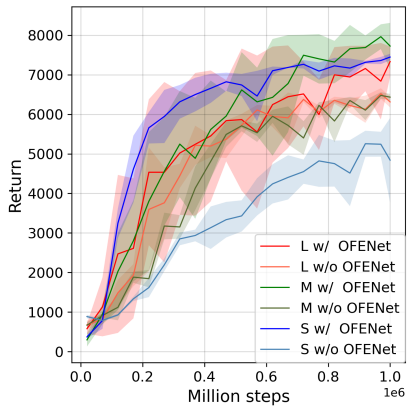
ture to improve RL performance[3]. MLP-ResNet is a modified version of Residual Networks (ResNet) [64, 65], which has a skip-connection that bypasses the nonlinear transformations with an identity function: $y_i = f_i^{\text{res}}(y_{i-1}) + y_{i-1}$, where y_i is the output of the i^{th} layer, and f_i^{res} is a residual module, which consists of a fully connected layer and a nonlinear activation function. An advantage of this architecture is that the gradient can flow directly through the identity mapping from the top layers to the bottom layers. MLP-D2RL is identical to Sinha *et al.* [164], and MLP-DenseNet is our proposed architecture defined in Sec. 3.2.3. We compare these four architectures on both small networks ($N^{\text{unit}} = 128$, denoted by S) and large networks ($N^{\text{unit}} = 2048$, denoted by L).

The training curves of the average return are shown in Fig. 3.7a, and the effective ranks in Fig. 3.7b. The results show that our MLP-DenseNet achieves the highest return on small and large networks while mitigating rank collapse comparable to MLP-D2RL. This shows that MLP-DenseNet is the best architecture among these four choices, and thus we employ this architecture for both the policy network and the value function network in the following experiments.

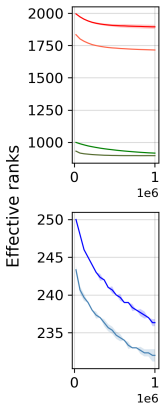
Decoupling representation learning from RL Next, we evaluate the effectiveness of using OFENet (see Sec. 3.2.1) to decouple representation learning from RL. In order to evaluate the performance on different network sizes, we sample the number of units from $N^{\text{units}} \in \{256, 1024, 2048\}$, which we respectively denote S, M, and L, and compare these against the baseline SAC agents, which do not use OFENet-like structure with the auxiliary loss and are trained only from a scalar reward signal. In other words, the baseline agents are identical to the DenseNet architecture of the previous connectivity comparison experiment.

The results in Fig. 3.8 show separating representation learning from RL improves control performance and mitigates rank collapse of Q-networks regardless of network size. Thus,

[3]It should be noted that while we have conducted a similar architecture comparison in Section 2.4.1, here we compare different architecture for RL agents (policy and value functions), not OFENet.



(a) Average return.



(b) Effective ranks.

Figure 3.8: Training curves of w/ and w/o OFENet on Ant-v2. This shows decoupling representation learning from RL is generally effective across the different sizes of the networks in terms of both control performance and mitigating rank collapse issues.

we can conclude using bigger representations, which are learned using the auxiliary task (see Sec. 3.2.1), contributes to improving performance on downstream RL tasks.

To investigate more in-depth, we also conduct a grid search over the different number of units for both SAC and OFENet in Fig. 3.9. The baseline is SAC agent without OFENet (see leftmost column). The results suggest that the performance does improve when compared against the baseline agent (see horizontally), however, it saturates around the average return of 8000. In the following experiments, we employ distributed replay and expect we can attain higher performance.

Distributed RL Finally, we add distributed replay [77] to further improve performance while using larger networks. We use an implementation similar to Stooke *et al.* [166], which collects experiences using N^{core} cores on which each core contains N^{env} environments, specifically we used $N^{\text{core}} = 2$ and $N^{\text{env}} = 32$.

Similarly to the previous experiments, we perform a grid search on the different number of units for SAC and OFENet with the distributed replay in Fig. 3.10, and also compare the training curves of three different network size S, M, and L in Fig. 3.12. Note that the horizontal axis in Fig. 3.12 is the number of times we applied gradients to the network, not the number of interactions. Comparing Fig. 3.10 and Fig. 3.9, we can clearly see that distributed training enables further performance gain in all sizes of networks. Furthermore, we can observe a monotonic improvement when we increase the number of units for both SAC and OFENet. Thus, we verified that combining distributed replay contributes to further performance gain while training larger networks.

How about generalization to different RL algorithms and environments? To quantitatively measure the effectiveness of our method across different RL algorithms and tasks, we evaluate two popular optimization algorithms, namely SAC and TD3 [52, 60], on five different locomotion tasks in MuJoCo [175]. We denote our method by *Ours*, which uses the largest network of $N^{\text{units}} = 2048$ among the previous experiments for the OFENet

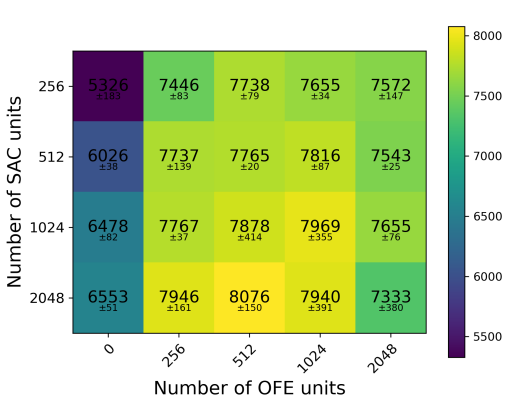


Figure 3.9: Grid search results of average maximum return over the different number of units between SAC and OFENet. OFENet can improve performance in almost all settings, but saturates around the return of 8000.

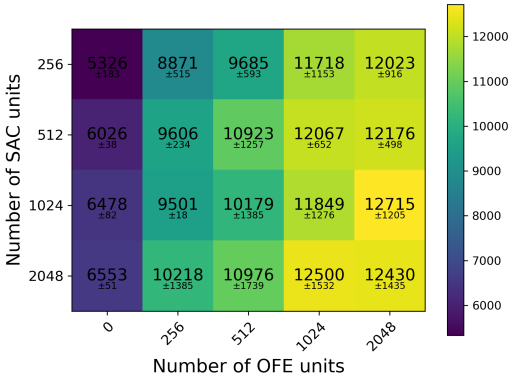
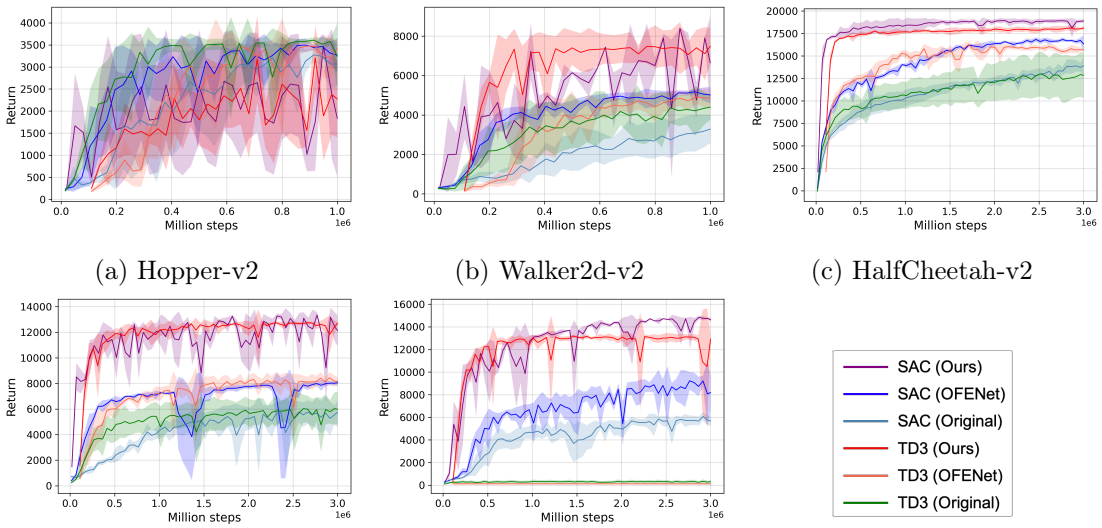


Figure 3.10: Grid search results of average maximum return over the different number of units between SAC and OFENet with ApeX-like distributed training. Compared to Fig. 3.9, adding distributed RL enables monotonic improvement when we widen SAC or OFENet.

and the RL algorithms. We compare the proposed method with two baselines: the original RL algorithm denoted by *Original*. Furthermore, we also compare OFENet, which can achieve current state-of-the-art performance on these tasks to the best of our knowledge.

We plot the training curves in Fig. 3.11 and list the highest average return in Table 3.1. In the figure and the table, our method *SAC (Ours)* and *TD3 (Ours)* achieves the best performance in almost all environments. Furthermore, we can see that our proposed method can work with both RL algorithms and thus is agnostic to the choice of the training algorithm. In particular, our method notably achieves much higher episode returns in Ant-v2 and Humanoid-v2, which are harder environments with larger state/action space, requiring more training samples. Interestingly, the proposed method does not achieve reasonable solutions in Hopper-v2, which has the smallest dimensionality among five environments. We consider that the performance in smaller dimension problems saturates early and even additional methods cannot provide any significant performance gain.



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 3.11: Training curves on five different MuJoCo tasks with two different RL algorithms (SAC and TD3). The proposed methods denoted as *SAC (Ours)* and *TD3 (Ours)* improve performance by employing larger networks.

Table 3.1: The highest average returns for each environment. The bold number indicates the best performance. Our method outperforms OFENet [140] and the original algorithm in most environments.

	SAC		
Environment	Ours	OFENet	Original
Hopper-v2	3467.3	**3511.6**	3316.6
Walker2d-v2	**8802.4**	5237.0	3401.5
HalfCheetah-v2	**19209.9**	16964.1	14116.1
Ant-v2	**14021.0**	8086.2	5953.1
Humanoid-v2	**14858.2**	9560.5	6092.6
	TD3		
Environment	Ours	OFENet	Original
Hopper-v2	3206.7	3488.3	**3613.0**
Walker2d-v2	**7645.8**	4915.1	4515.6
HalfCheetah-v2	**18147.5**	16259.5	13319.9
Ant-v2	**12811.3**	8472.4	6148.6
Humanoid-v2	**13282.0**	120.6	340.5

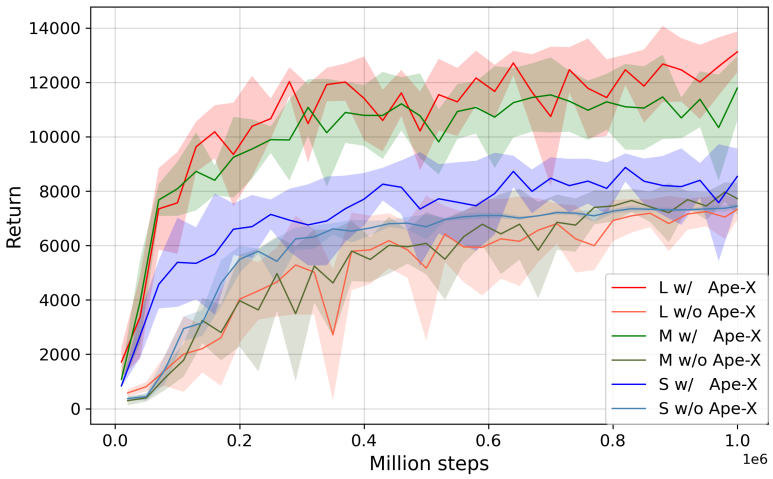


Figure 3.12: Training curves on the performance of our proposed framework w/ and w/o using Ape-X architecture on Ant-v2 environment.

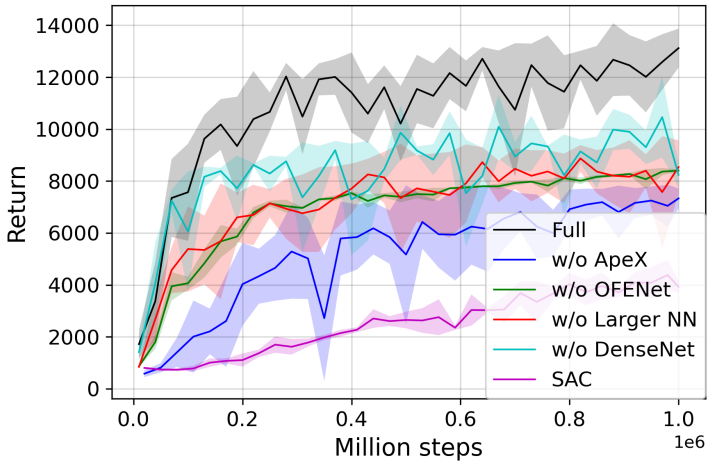


Figure 3.13: Training curves of the derived methods of SAC on Ant-v2. This shows that each element does contribute to performance gain, and our combination of DenseNet architecture, distributed training, and decoupled feature representation (shown as *Full*) allows us to train larger networks that perform significantly better compared against the baseline SAC algorithm (shown as *sac*).

3.4.3 Ablation study

Since our method integrates several ideas into a single agent, we conduct additional experiments to understand what components contribute to the performance gain. We highlight that our method consists of three elements: feature representation learning using OFENet, DenseNet architecture, and distributed training. Furthermore, we compare the results without increasing the network size to reinforce that a larger network improves performance. Figure 3.13 shows the ablation study on SAC with Ant-v2 environment. *Full* is our method, which combines all three elements we proposed and uses large networks ($N^{\text{unit}} = 2048, N^{\text{layer}} = 2$) for the SAC agent. *sac* is the original SAC implementation.

w/o Ape-X removes Ape-X-like distributed training setting. As distributed RL enables the collection of more experiences close to the current policy, we consider that the significant performance gain can be explained by learning from more on-policy data, which was also empirically shown by Fedus *et al.* [46]. Also, we believe that receiving more novel experiences helps the agent generalize to state-action space. In other words, although the off-policy training setting obtains a new experience by interacting with the environment, it is much less compared to the on-policy setting, resulting in overfitting to the limited trajectories, which becomes more problematic in harder environments, which have larger state/action space and larger neural networks. Fu *et al.* [48] have also empirically proven this issue.

w/o OFENet removes OFENet and trains the entire architecture using only a scalar reward signal. The much lower return shows that learning the large networks from just the scalar reinforcement signal is difficult, and training the bottom networks (close to the input layer), i.e. obtaining informative features by using an auxiliary task, enables better learning of control policy.

w/o Larger NN reduces the number of units from $N^{\text{unit}} = 2048$ to 256 for both OFENet and SAC. This also significantly decreases performance; therefore, we can conclude that

the use of larger networks is essential to achieve high performance.

Finally, *w/o DenseNet* replaces the MLP-DenseNet defined in Sec. 3.2.3 with the standard MLP architecture. The result shows that strengthening feature propagation does contribute to improving performance.

It is noted that while fully using the proposed architecture improves the performance the most, each component (decoupling representation learning from RL, distributed training, and network architecture) also contributes to the performance gain.

3.4.4 Sparse reward setting

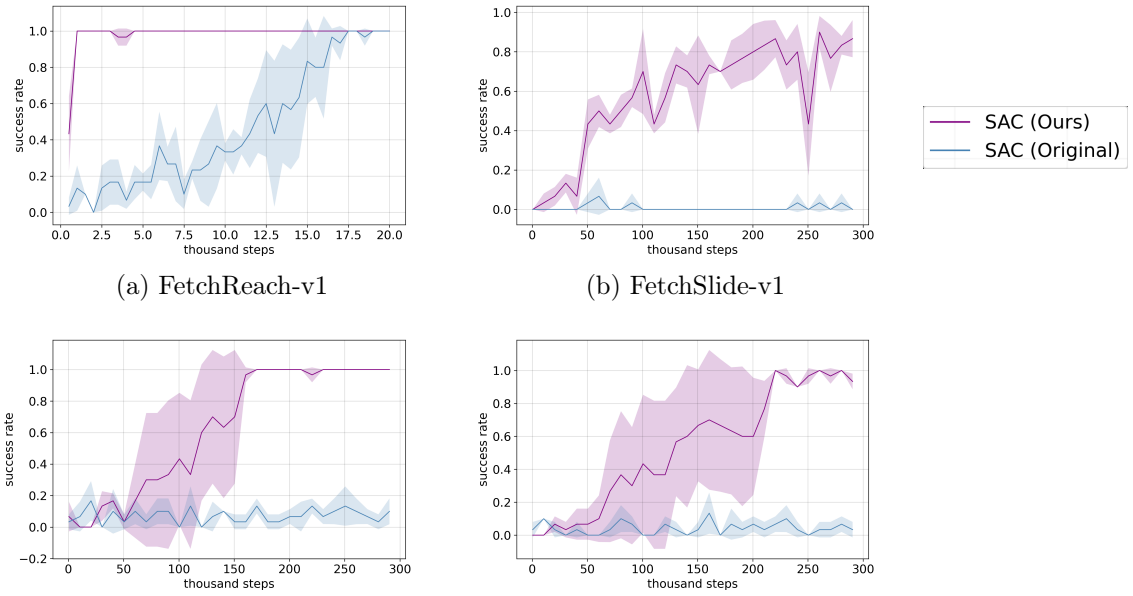
So far, we have evaluated the proposed framework in dense reward settings. In this section, we apply our framework to the sparse reward settings in multi-goal environments discussed in Plappert *et al.* [146], which includes a 7 DoF robot manipulator (Fetch) to perform Reach, Slide, Push, and PickAndPlace tasks. Since our framework does not need to change the underlying RL algorithms, it can be naturally combined with any plug-and-play method. Specifically, we combine our framework with Hindsight Experience Replay (HER) [7], which is a standard method for solving RL problems with sparse reward settings.

Figure 3.14 shows the success rates of the four different sparse reward environments. It clearly shows that the proposed framework enables the agent to learn a better policy compared to the baseline method, which is a naive combination of HER and SAC; that is, it does not include the proposed framework that consists of three components. Specifically in difficult settings (Slide, Push, and PickAndPlace), our method quickly converges to a success rate of almost 100 %, while the original algorithm does not achieve the goal in 300 thousands of steps.

3.4.5 Analysis on computation and sample efficiency

In the final experiment, we evaluate the performance of the proposed framework with respect to *wall clock time* and *sample efficiency*.

Environmental and gradient steps per second First, we analyze the number of environmental and gradient steps per second for the four different methods. Table 3.2 compares the gradient and environmental steps averaged over all environments for *Ours*, *Ours w/o ApeX*, *OFENet*, and *Original*. Comparing *Ours* and *Ours w/o ApeX*, which is our proposed framework without the ApeX-like distributed training, the proposed method collects $70.6 (= 1362.0/17.3)$ times more transitions per second by employing the distributed training framework. Therefore, more on-policy transitions are stored in a replay buffer compared to those without a distributed training framework, which is important to train high-performance policy. Focusing on the gradient steps, *Ours* has more steps than *Ours w/o ApeX*, because our framework conducts policy updates and transition collection asynchronously, which will be effective for performance in terms of wall clock time performance.



(a) FetchReach-v1 (b) FetchSlide-v1

(c) FetchPush-v1 (d) FetchPickAndPlace-v1

Figure 3.14: Training curves on four different sparse reward tasks with SAC combined with Hindsight Experience Replay. The proposed framework denoted as *SAC (Ours)* converges to a success rate of almost 100 % compared to the naive baseline *SAC (Original)* that does not include our proposed framework.

Table 3.2: The number of environment and gradient steps per second averaged over all environments. Note that the environment and gradient steps are different only for *Ours* because of the distributed training.

	Ours	Ours w/o ApeX	OFENet	Original
Env. steps / sec	1362.0	15.4	29.8	43.9
Grad. steps / sec	19.3	15.4	29.8	43.9
Env. steps / Grad. steps	70.6	1.0	1.0	1.0

Wall clock time performance Next, we compare the performance of the models with respect to the *wall clock time*, that is, we compare models trained over a fixed period of time. Specifically, we compare the performance of all methods for the amount of time taken by the naive baseline *SAC (Original)* to complete the training.

The result in Table 3.3 shows that our method *SAC (Ours)* achieves the best performance in almost all environments, although the number of gradient steps is 2.3(= 43.9/19.3) times less than *Original*. This is achieved by using the distributed training framework; our method collects more on-policy transitions thanks to the asynchronous data collection setup, which enables the agent to overfit its networks to the experience replay (see Section 3.2 for more details). Thus, we show that the proposed framework is effective with respect to the wall clock time.

Sample efficiency Next, we analyze the sample efficiency, where we compare our method with baselines with the same number of *environmental steps* (thus we do not consider the

	Ours w/o Dist. RL	Ours
Env. steps / sec	15.4	1362.0
Grad. steps / sec	15.4	19.3
Env. steps / Grad. steps	1.0	70.6

Table 3.3: The highest average returns for each environment with fixed wall clock time, specifically as of the naive baseline *SAC (Original)* agent completed training, for analyzing the performance gain with respect to wall clock time. The numbers in bold indicate the best performance. Our method outperforms OFENet [140] and the original algorithm in most environments for complex environments (HalfCheetah-v2, Ant-v2, and Humanoid-v2).

	SAC		
Environment	Ours	OFENet	Original
Hopper-v2	2798.9	**3406.4**	3316.6
Walker2d-v2	4633.8	**4813.7**	3401.5
HalfCheetah-v2	**18345.3**	16214.91	14116.1
Ant-v2	**13008.5**	7748.0	5953.1
Humanoid-v2	**12678.1**	7944.0	6092.6
	TD3		
Environment	Ours	OFENet	Original
Hopper-v2	1597.0	3458.4	**3613.0**
Walker2d-v2	**7336.2**	4392.5	4515.6
HalfCheetah-v2	**17784.5**	15808.6	13319.9
Ant-v2	**12463.4**	8199.9	6148.6
Humanoid-v2	**12970.8**	119.5	340.5

wall clock time).

Table. 3.4 shows the performance of each method with a fixed number of environmental steps (1 million for Hopper-v2 and Walker2D-v2 and 3 million for the others). Note that the returns of *SAC (Original)* and *SAC (OFENet)* are identical to those in Table 1 of the manuscript because these methods do not use the distributed training framework. Comparing *Ours* with baselines (*OFENet* and *Original*), our proposed framework performs worse than the baselines. This is because it collects a huge amount of samples very quickly, thanks to the asynchronous training architecture (Fig. 3 in the manuscript); the data collection speed of the proposed framework is 88 times faster than that of the naive baseline. Therefore, effectively, our proposed method is able to apply only fewer gradient steps (approximately 100 times less) than the compared baselines. This results in poor performance, as there are simply not enough updates to train the model.

Finally, to show the effectiveness of the other two components of the proposed framework, namely the DenseNet architecture and decoupling representation learning from RL, we also added results without distributed training, denoted by *Ours w/o ApeX* in Table. 3.4. From the table, *Ours w/o ApeX* achieves much higher performance than the baselines, while it has the same sample efficiency as the baselines, since it does not use distributed training. Therefore, one can choose the framework with or without distributed training based on the

Table 3.4: The highest average returns for each environment with the fixed number of environmental steps to analyze the performance with respect to *sample efficiency*. The numbers in bold indicate the best performance. Our method works worse than OFENet [140] and the original algorithm in most environments due to the nature of asynchronous training. However, our framework without distributed training (*Ours w/o ApeX*) works much better than baselines, indicating one can remove distributed training when sample efficiency is most important.

	SAC			
Environment	Ours	Ours w/o ApeX	OFENet	Original
Hopper-v2	532.1	3452.7	**3511.6**	3316.6
Walker2d-v2	621.2	**6385.3**	5237.0	3401.5
HalfCheetah-v2	14548.5	**17942.6**	16964.1	14116.1
Ant-v2	8508.0	**10249.5**	8086.2	5953.1
Humanoid-v2	1177.3	**10720.3**	9560.5	6092.6
	TD3			
Environment	Ours	Ours w/o ApeX	OFENet	Original
Hopper-v2	367.5	3562.1	3488.3	**3613.0**
Walker2d-v2	635.9	**6021.3**	4915.1	4515.6
HalfCheetah-v2	14474.9	**17402.8**	16259.5	13319.9
Ant-v2	7216.5	**9820.9**	8472.4	6148.6
Humanoid-v2	4302.1	**10235.3**	120.6	340.5

importance of sample efficiency; While our full framework is not very sample efficient due to the nature of distributed training, the proposed method without distributed training still performs much better than the baselines. Implementing the distributed framework can really improve performance due to the distributed implementation and the large amounts of on-policy diverse data.

3.5 Discussion

Deep learning has catalyzed huge breakthroughs in the fields of computer vision and natural language processing, making use of massive neural networks that can be trained with huge amounts of data. Although these domains have greatly benefitted from the use of larger networks, the RL community has not witnessed a similar trend in the use of larger networks to train high-performance agents. This is mainly due to the instability when using larger networks to train RL agents. In this paper, we studied the problem of using a larger network to train RL agents. To achieve this, we proposed a novel framework while reflecting on some of the important design choices that one has to make when using such networks. In particular, the proposed framework consists of three elements. First, we decouple the representation learning from RL using an auxiliary loss to predict the next state. This allows more informative features to be obtained to learn control policies with richer information than learning entire networks from a scalar reward signal. The learned representation is then propagated to the DenseNet architecture, which consists of very wide networks. Finally, a distributed training framework provides huge amounts of

on-policy data whose distribution is much closer to the current policy, and thus enables the RL agent to mitigate the overfitting problem and enhance generalization to novel scenarios. Our experiments demonstrate that this novel combination achieves significantly higher performance compared to current state-of-the-art algorithms across different off-policy RL algorithms and different continuous control tasks. Although this paper focused on learning from state vectors, we plan to apply the proposed framework to high-dimensional observations, such as images, in future work.

Part II

How can we estimate an effective object's representation for performing manipulation?

Chapter 4

Probabilistic Estimation for Understanding Objects' Articulations

Every day, we are surrounded by a number of articulated objects that require specific interactions to use: our laptops can be opened or shut, windows can be raised or lowered, and drawers can be pulled out or pushed back in. A robot designed to function in real-world contexts should therefore be able to understand and interact with these articulated objects.

Recent advances in deep reinforcement learning (RL) have focused on this problem and have allowed robots to manipulate articulated objects such as drawers and doors [57, 176, 190, 197]. However, these systems typically produce fixed actions based on observations of a scene, and thus when the articulated joint is ambiguous (e.g., a door that slides or swings), they cannot adapt their policies in response to failed actions. Although some systems attempt to adjust policies during test-time exploration to recover from failure modes [47, 187], they only propose local action adjustments (pull harder or run faster) and so are insufficient in cases where dramatically different strategies need to be applied, e.g., from "sliding the window" to "pushing the window outward from the bottom."

In contrast, humans and many other animals can quickly figure out how to manipulate complex articulated man-made objects, e.g., puzzle boxes, with very little training [8, 17, 174]. These capabilities are thought to be supported by rapid, strategic trial-and-error learning – interacting with objects in an intelligent way, but learning when actions lead to failures and updating mental representations of the world to reflect this information [4]. We argue that robotic systems that can learn how to manipulate articulated objects should be designed using similar principles.

In this work, we propose "Hypothesize, Simulate, Act, Update, and Repeat" (H-SAUR), an exploration strategy that allows an agent to figure out the underlying articulation mechanism of man-made objects from a handful of actions. At the core of our model is a probabilistic generative model that generates hypotheses of how articulated objects might deform given an action. Given a kinematic object, our model first generates several hypothetical articulation configurations of the object from 3D point clouds segmented by object parts. Our model then evaluates the likelihood of each hypothesis through analysis-by-synthesis – the proposed model simulates objects representative of each hypothetical configuration, using a physics engine to predict likely outcomes given an action. The virtual

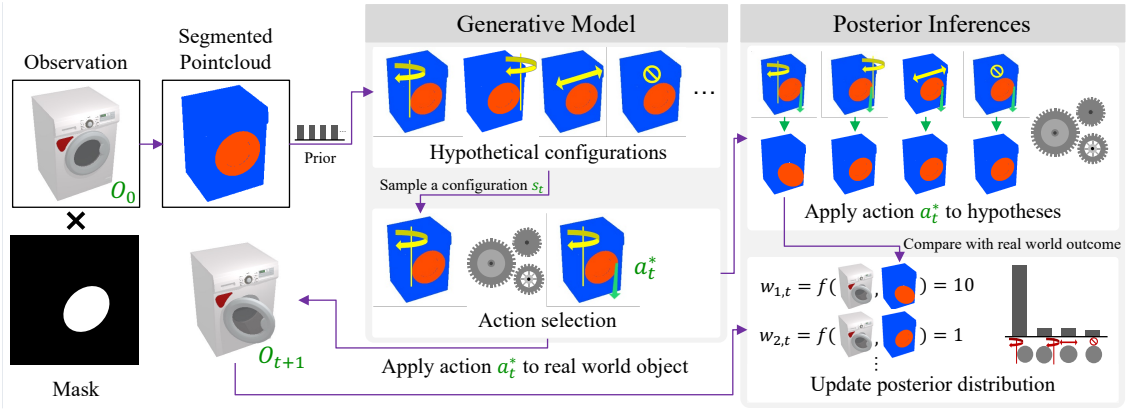


Figure 4.1: **Overview of our "Hypothesize, Simulate, Act, Update, and Repeat" (H-SAUR) framework.** We consider the task of estimating the kinematic structure of an unknown articulated object and use that structure to manipulate the object efficiently. **Left:** A generative model produces several hypothetical configurations given point cloud segments and simulates possible actions that maximally deform a sampled configuration. **Right:** By applying an action and observing the outcome, the posterior inference is performed using the same generative model by simulating and updating the posterior distribution. We repeat the process until convergence.

simulation helps resolve three critical components in this interactive perception setup: (1) deciding real-world exploratory actions that might produce meaningful outcomes, (2) reducing uncertainty over beliefs after observing the action-outcome pairs from real-world interactions, (3) generating actions that will lead to successful execution of a given task after fully figuring out the articulation mechanism. The contributions of this chapter can be summarized as follows:

1. We propose a novel exploration algorithm for efficient exploration and manipulation of puzzle boxes and articulated objects, by integrating the power of probabilistic generative models and forward simulation. Our model explicitly captures the uncertainty over articulation hypotheses.

2. We compare H-SAUR against existing state-of-the-art methods, and show it outperforms them in operating unknown articulated object, despite requiring many fewer interactions with the object of interest.

3. We propose a new manipulation benchmark – *PuzzleBoxes* – which consists of locked boxes that require multi-step sequential actions to unlock and open, in order to test the ability to explore and manipulate complex articulated objects.

4.1 Related Work

Kinematic Structure Estimation. A natural first step to manipulate an object is to predict the articulation mechanism of the object. Li *et al.* [110] and Wang *et al.* [186] proposed models to segment object point clouds into independently moving parts and articulated joints. However, this requires part and articulation annotations, and thus does

not generalize to unexpected articulation mechanisms. Previous work address this by proposing to visually parse articulated objects under motion [54, 68, 85, 91, 168, 208]. Yet, most work assumes the objects are manually articulated by humans or scripted actions from the robot. In this paper, we study how an agent can jointly infer articulation mechanism and exploratory actions that helps to reveal the articulation of an object, i.e., in an interactive perception setup [16]. Hausman *et al.* [62] addresses a similar setup, but only handles articulated objects with a single joint and assumes the robot knows where to apply forces. Kulick *et al.* [12] and Baum *et al.* [13] handle dependency joints but assume each joint is either locked or unlocked, which is ambiguous for general kinematic objects. H-SAUR takes raw point clouds and part segmentations as inputs, and infers both the joint structure of the object and how to act. This model can handle articulated objects with an arbitrary number of joints and joint dependencies by leveraging off-the-shelf physics simulation for general physical constraint reasoning.

Model-free approaches for manipulating articulated objects. Instead of explicitly inferring the articulation mechanism, recent works in deep RL learn to generate plausible object manipulation actions from pointclouds [129, 187, 190], RGB-(D) images [57, 176, 197], or the full 3D state of the objects and their segments [3, 144, 198]. While most of these RL approaches learn through explicit rewards, recent approaches have learned to manipulate objects in a self-supervised manner, through self-driven goals or imitation learning [40, 117]. However, all of these systems require a large number of interactions during training and cannot discover hidden mechanisms that are only revealed through test-time exploratory behaviors. Furthermore, while they focus on *training-time* exploration, our work focuses on *testing-time* exploration where only a small number of interactions is permitted.

4.2 Method

We consider a task of estimating kinematic structure of an unknown articulated object and use the estimation for efficient manipulation. We are particularly interested in manipulating a visually ambiguous object, e.g., a closed door that can be opened by pulling, pushing, sliding, etc. In such a situation, the agent needs to estimate its underlying kinematic configuration, and update its beliefs over different configurations based on the outcome of past failed actions.

We propose "Hypothesize, Simulate, Act, Update, and Repeat" (H-SAUR), a physics-aware generative model that represents an articulated object manipulation scene in terms of 3D shapes of object parts, articulation joint types and positions of each part, actions to apply on the object, and the change to the object after applying the actions. In this work, we assume to have access to a physics engine that can take as input 3D meshes (estimated from a point cloud) of a target unknown object with an estimated kinematic configuration, and produce hypothetical simulated articulations of this object when kinematically acted upon. The method consists of three parts. First, we initiate a number of hypothetical configurations that imitate a target object by sampling articulation structures from a prior distribution. The prior distribution can be uniform or from learned vision models. Second, we sample one of the hypotheses to generate an action that is expected to provide evidence

for or against that hypothesis. Finally, we apply the optimal action to the target object and update beliefs about object joints based on the outcome.

4.2.1 Generating Hypothetical Articulated Objects

Given the observed pointcloud O of a target object along with its part segmentation, m, we generate a number of kinematic replicas of the object. Since the true articulation mechanism is initially unknown, we generate these replicas by sampling different kinematic structures from uniform prior distributions over joint types and parameters.

Object Parts. From the observed pointcloud O and segmentation masks, $m_1, m_2, \cdots, m_{N_v}$, where N_v is number of available views, we can break the pointcloud into part-centric pointcloud $O^1, O^2, \cdots, O^{N_p}$ where N_p is the total number of object parts.

Articulation Joints. Each object part is attached to a base of the object with a joint. We consider three most common types of articulation joints: revolute (r), prismatic (p), and fixed (f). For revolute and prismatic joints, we further generate possible joint axes and positions, using the tight bounding boxes fitted to the part-centric pointcloud to obtain a total of J possible joints. The j^{th} joint is denoted as $\theta^{(j)} = (c, d)$ where $c \in \{r, p, f\}$ is the joint type and $d \in \mathbb{R}^6$ is the 6-DoF pose of the joint axis. The prior distribution $p(\theta^{(j)})$ for the joint type is assumed to be uniform at $t = 0$. One can also use learned prior from vision models that predict joint types.

In addition, most articulated joints have lower and upper limits of how much the joint can be deformed. We denote the limits as θ^{low} and θ^{high}. The prior distribution is sampled uniformly from $[-\theta_{\text{MAX}}, 0]$ and $[0, \theta_{\text{MAX}}]$, respectively. The full state of the joint for object part O^i is $s^i = (\theta^{(\sigma(i))}, \theta^{\text{low}_i}, \theta^{\text{high}_i}, \theta^{\text{cur}_i})$, where $\sigma(i) \in \{1, 2, \cdots, J\}$ is the joint configuration for the i^{th} object part, and θ^{cur_i} is the joint position at the current time step. The prior over all the latent variables is:

$$p(s^{1:N_p}) = \prod_{i=1}^{N_p} p(\theta^{(\sigma(i))}) p_{\text{unif}[-\theta_{\text{MAX}}, 0]}(\theta^{\text{low}_i}) p_{\text{unif}[0, \theta_{\text{MAX}}]}(\theta^{\text{high}_i}). \tag{4.1}$$

We approximate the distribution by maintaining a particle pool, $\mathcal{S}$, where each particle in the pool represents a particular setup for the articulation configurations.

4.2.2 Simulating and Selecting Informative Action

We utilize virtual simulations to generate an optimal action that reduces the uncertainty of joint configuration hypotheses. Yet, computing the optimal action that maximizes the information gain involves integral over all latent variables, which is intractable. One can approximate this by a sampling-based method [16]. However, the high computational requirements still prohibit the agent from solving the task within a reasonable time. We address this by using only a single particle to make a noisy approximation of the optimal action.

We sample a joint configuration from the set of particles $s^{(k)} \sim \mathcal{S}$ and obtain the optimal action by simulating different actions on the object with the physics simulation. The action $a_t = (p, r) \in \mathbb{R}^6$ is represented as a 3D point $p_t \in \mathbb{R}^3$ on the object and the direction $r_t \in \mathbb{R}^3$

to apply force. The optimal action is defined as the action that can maximally deform the object or a target object part over a single step. For multi-part objects, we maintain a list of parts-of-interest, which we will introduce shortly, and we sample a target part from the list to act on. We measure how much an object part i deforms by $d_i = \|\theta_{t+1}^{\text{cur}_i} - \theta_t^{\text{cur}_i}\|$. Although one can naively sample a huge number of actions and pick the best action through simulation, we found this can be extremely inefficient with large object parts. To improve inference speed, we instead treat the action inference as a particle filtering problem: we initialize a number of action proposals by randomly sampling 3D locations on the target point cloud and assign random 3D directions to apply force, then we use the measured distance d_j as the likelihood to update the posterior distribution of the particles. We add noise to the action while reproducing the particles from previous iterations. We continue this process three times and finally sample a particle from the pool to obtain the action a^*. [1] We found the inferred action a^* is often close to the oracle optimal action that maximizes d_i.

The probabilistic formulation of an articulation mechanism given past observation and action is

$$p(s_t|O_{1:t-1}a_{1:t-1}) = \int \underbrace{p(s_t|s_{t-1}, a_{t-1})}_{\text{forward dynamics}} \underbrace{p(s_{t-1}|O_{1:t-1}, a_{1:t-1})}_{\text{obtain through recursion}}, \tag{4.2}$$

where the first term is handled by the physics engine by forward simulation, and the second is initialized with the prior defined in Eq. (4.1) and can be obtained through recursion.

4.2.3 Updating hypotheses through analysis-by-synthesis

We apply the inferred action a^* on the target object O_t to observe outcome O_{t+1}. We then update the probability of each hypothesis through analysis-by-synthesis: we first apply the same action a^* on all the "imagined" objects, $s \in \mathcal{S}$ in the physics engine. After applying the action, we obtain $\hat{O}_{t+1}^{(k)}$ for each particle $s^{(k)}$. We define the likelihood of the particle $s^{(k)}$ as $w_k = \frac{1}{\text{dist}(O_{t+1}, \hat{O}_{t+1}^{(k)}) + \epsilon}$, where $\text{dist}(o_1, o_2) = \frac{1}{|o_1|} \sum_{x \in o_1} \min_{y \in o_2} \|x - y\|_2^2$ is the chamfer distance between two point cloud o_1 and o_2. The overall updated posterior is:

$$\begin{aligned} p(s_t|O_{1:t}, a_{1:t-1}) &\propto p(O_t|s_t) p(s_t|O_{1:t-1}, a_{1:t-1}) \\ &= \sum_{k=1}^{K} w_k p(s_t|O_{1:t-1}, a_{1:t-1}), \end{aligned} \tag{4.3}$$

where the second term can be computed from Eq.(4.2), and the whole inference is implemented through particle filtering with weighted sampling.

4.2.4 Handling Joints with Dependency in Goal-Conditioned Manipulation

A real puzzle box often consists of joints with dependencies, e.g., a lock needs to be open first in order to operate on another lock. Randomly selecting a part to act on is ineffective

[1] We found the particle filter (PF) generates nearly optimal action $1,500$ times faster compared to an oracle optimal action generated by exhaustive search (ES). We compare deformations caused by them and found that PF with 100 particles almost always generates the same action ($d_i^{\text{PF}}/d_i^{\text{ES}} = 0.995$).

and may not be sufficient to solve the problem since (1) the agent can act on a segment that is irrelevant to the task, e.g., a decoration on the box, and (2) the agent can underestimate the joint limit by ignoring the possibility that another part is blocking the current joint. To resolve this issue, we propose to keep track of the relevant parts and gradually grow a dependency tree throughout the exploration process.

Given goals in the form of "moving part X towards Y", we maintain a parts-of-interest list q_{POI} to keep track of task-relevant object parts and their desired position. For example, consider a door with a few locks, whose goal is to pull open the door. Thus, we initialize q_{POI} by adding the "door" part, O_0, and the desirable moving direction d_0. When selecting an action (see section 4.2.2), we always act on the most recently added object part. In the first run, we select the door since it is the only part in the list.

Using the physics engine, we not only infer the optimal action that would cause desirable changes to the target part, but also detect object parts, e.g., locks on the *PuzzleBoxes* we introduce shortly, that will collide with it. We consider these collided parts as having a dependency with target part at hand. We can further infer the desirable change direction d_i for each of these collided parts O_i that would unblock the current part. Then, we add the part along with the desired changing direction to q_{POI}. Sometimes multiple directions might lead to a successful unblock, in this case, we randomly select one direction to be put in the list. We expect the pool of particles to keep track of different sampling outcomes. We can keep adding "unsolved" parts with dependencies to the current parts to the list. A part is marked as "solved" and removed from the list if it can be and has been changed to a desired configuration that unblocks its parent node in the dependency tree.

Algorithm 2, 3, 4, and 5 show the pseudocode for the proposed method. Specifically, Algorithm 4, 5 handles joints with dependency, as described in this section. Since objects in the PartNet-Mobility dataset do not have such dependency, they can be solved with Algorithm 2, 3 without the dependency check. For objects in the PuzzleBox dataset, including the dependency check is critical to achieve reasonable performance.

Algorithm 2 Interactive Joint Type Estimation

Input Observed point cloud for the current movable part O^i, hypothetical joint configurations $\boldsymbol{\theta} = \{\theta^{(j)} | j \in \{1, 2, \cdots, J\}\}$, number of particles $N^{\text{particles}}$, threshold δ^{prob} to finish estimation, maximum steps to interact with the real-world object $N^{\max}$

Output Estimated joint type j^* and indexes of the collided movable parts k

Initialize a particle pool $\mathcal{S}^{\text{state}} = \emptyset$
for $k \leftarrow 1$ to $N^{\text{particles}}$ **do**
    Sample a joint configuration $\theta^{(\sigma(k))} \sim p_0(\boldsymbol{\theta})$ ▷ Can use a learned prior distribution
    Sample a joint upper limit $\theta^{\text{high}_k} \sim p_{\text{unif}[0, \theta_{\text{MAX}}]}$
    Sample a joint lower limit $\theta^{\text{low}_k} \sim p_{\text{unif}[-\theta_{\text{MAX}}, 0]}$
    Add the sampled particle $s^k = (\theta^{(\sigma(k))}, \theta^{\text{low}_k}, \theta^{\text{high}_k}, \theta^{\text{cur}_k} = 0)$ to the particle pool $\mathcal{S}^{\text{state}} \leftarrow \mathcal{S}^{\text{state}} \cup s^k$
end for
for $t \leftarrow 1$ to $N^{\max}$ **do**
    Infer an informative action a^* using Alg. 3
    Apply the inferred action a^* on the real-world object and observe the pointcloud O_{t+1}
    // Handling movable parts/joints with dependency
    if Detect contacts between the current movable part and other movable parts **then**
        Break the current loop
    end if
    for all $k \leftarrow 1$ to $N^{\text{particles}}$ **do**
        Apply the inferred action a^* on k-th hypothetical object and observe the pointcloud $\hat{O}^k_{t+1}$
        Compute importance weight for the particle $w_k = \frac{1}{\text{dist}(O_{t+1}, \hat{O}^k_{t+1}) + \epsilon}$
    end for
    Re-sample particles from $\mathcal{S}^{\text{state}}$ according to the importance weights. Compute updated posterior $p_{t+1}(\boldsymbol{\theta})$ from the particles.
    if $\max_{j \in \{1, \ldots, J\}} p_{t+1}(\theta^{(j)}) > \delta^{\text{prob}}$ **then**
        Break the current loop
    end if
end for
return The most probable joint configuration $j^* = \arg\max_{j \in \{1, \ldots, J\}} p(\theta^{(j)})$ and indexes of the collided movable part k

Algorithm 3 Informative Action Selection

Input Observed point cloud for the current movable part O_i, particle pool $\mathcal{S}^{\text{state}}$, number of particles for generating action $N^{\text{particles-action}}$, number of particle updates N^{update}

Output Informative action a^*

Sample a joint configuration $s \sim \mathcal{S}^{\text{state}}$
Reproduce a hypothetical object using the joint configuration s in a physics engine
Initialize a particle pool $\mathcal{S}^{\text{action}} = \emptyset$
for $k \leftarrow 1$ to $N^{\text{particles-action}}$ **do**
    Sample a point to interact $p_k \sim O_i$
    Sample a force direction $r_k \sim \{+\text{x},\text{-x}, +\text{y}, \text{-y}, +\text{z}, \text{-z}\}$
    Add the sampled particle $a_k = (p_k, r_k)$ to the particle pool $\mathcal{S}^{\text{action}} \leftarrow \mathcal{S}^{\text{action}} \cup a_k$
end for
for $n \leftarrow 1$ to N^{update} **do**
    for $k \leftarrow 1$ to $N^{\text{particles-action}}$ **do**
        Apply the sampled action a_k on the hypothetical object and observe the joint state $\theta^s_{t+1,k}$
        Compute importance weight for the particle $w_k = \|\theta^s_{t+1,k} - \theta^s_{t,k}\|$
    end for
    Re-sample particles from $\mathcal{S}^{\text{action}}$ according to the importance weights
    Add noise to the position $p_k \in \mathcal{S}^{\text{action}}$
end for
return The most probable particle $a^* = a_{k^*}$, where $k^* = \underset{k \in \{1,\ldots,N^{\text{particles-action}}\}}{\arg\max} w_k$

Algorithm 4 Unlock PuzzleBoxes

Input Pointcloud of the target object O, segmentation masks $m_1, m_2, \cdots, m_{N_v}$, target part ID t to open, desired direction d^t to displace the target part

Output "Success" or "Failure" to solve the task

1: Obtain segmented point clouds $O_1, O_2, \cdots, O_{N_v}$ from O and $m_1, m_2, \cdots, m_{N_v}$
2: Initialize the part-of-interest queue with the target part, e.g., the door, and its desired opening direction $q_{\text{POI}} = \{(O^t, d^t)\}$
3: **while** $|q_{\text{POI}}| \geq 0$ **do**
4: Get the most recently added object part and its moving direction (O^l, d^l) from the queue q_{POI}
5: Estimate the joint type of the target part O^l using Alg. 2
6: **if** Detect collided parts during joint type estimation **then**
7: Add the collided part O^k and its unknown moving direction d^k to the last of the part-of-interest list $q_{\text{POI}} \leftarrow q_{\text{POI}} \cup \{(O^k, d^k)\}$
8: **else**
9: **if** The moving direction d^l is "unknown" **then**
10: Compute the moving direction d^l that resolves collision using Alg. 5
11: **end if**
12: Infer an optimal action a^* on the estimated joint type using Alg. 3 while replacing the cost function $w_k = \|\theta^s_{t+1,k} - d^l\|$ in line.12 in Alg. 3 and apply it on the real-world object
13: **if** Goal is achieved (e.g., "door is open") **then**
14: Terminate this experiment with "Success"
15: **else if** Number of interactions exceeds $N^{\text{max-int}}$ **then**
16: Terminate this experiment with "Failure"
17: **else if** part O^l has been successfully displaced with d^l **then**
18: Remove the current part and direction $q_{\text{POI}} \leftarrow q_{\text{POI}} \backslash \{(O^l, d^l)\}$
19: **end if**
20: **end if**
21: **end while**
22: **return** "Failure"

Algorithm 5 Compute Action Direction to Resolve Collision

Input Pointcloud of the current part of interest O^l, pointcloud of the collided object part O^{c}, estimated current part's joint configuration $s = \{\theta^{\text{low}}, \theta^{\text{high}}, \theta^l\}$, number of samples to search for the desired joint state N

Output Desired joint state θ^* for the collided object part

1: Reproduce a hypothetical object using the joint configuration s in a physics engine.
2: Obtain pointclouds $\{O^l_{\theta_1}, ..., O^l_{\theta_N}\}$ of different joint states $\theta \in \{\theta_1, ..., \theta_N\}$ by setting the joint configuration s with joint position θ in a physics engine, where θ is evenly spaced between θ^{low} and θ^{high}.
3: **return** The joint state $\theta^* = \underset{\theta \in \{\theta_1, ..., \theta_N\}}{\arg\max}\ \text{dist}(O^l_\theta, O^{\text{c}})$

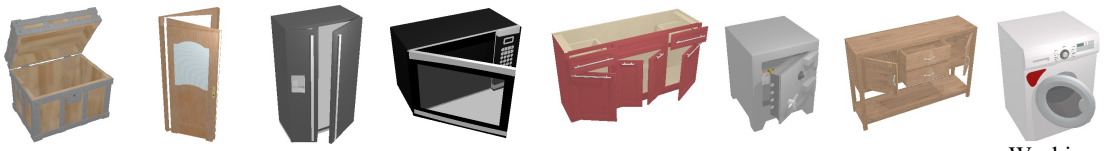


Figure 4.2: Categories from PartNet-Mobility dataset [194] used for our experiments.

Table 4.1: Statistics of the data splits.

Training Categories							Test Categories				
Category	Instances		Joints		Rev.	Pris.	Category	Instances	Joints	Rev.	Pris.
	Train	Test	Train	Test							
Box	10	3	10	3	X	-	Safe	29	29	X	-
Door	26	7	33	8	X	-	Table	62	153	X	X
Microwave	8	3	8	3	X	-	Washing	16	16	X	-
Fridge	34	9	56	17	X	-					
Cabinet	269	68	630	157	X	X					
All	347	90	737	188	X	X	All	107	198	X	X

4.3 Experiments

We evaluate H-SAUR on both the PartNet-Mobility dataset and the PuzzleBoxes dataset on the SAPIEN [194] physics engine.

The **PartNet-Mobility** dataset provides a wide variety of synthetic articulated objects. We specifically use 8 different categories as shown in Fig. 4.2 with two different settings: In the *closed* setting, all movable joints are closed, which is often the most visually ambiguous setup for an object. In the *half-opened* setting, all joints are initialized at the midway point between the joint limits. Detailed instance statistics and their joint types for each object category are listed in Table 4.1. For the object parts, we remove parts that are either too small or have a function irrelevant to physical articulation. For example, some instances in the *Washing Machine* category have tiny buttons to control the machines. We ignore these parts and focus on articulated parts, such as doors.

For physics simulation setups, we use frame rate 100 fps, and all the other settings as default in SAPIEN release. When interacting with an object, we apply the same force for 10 simulation steps, resulting in 10 fps simulation. To simulate actions from position control, we set the magnitude of the force by multiplying $100 \times m$, where m is the mass of the target part. Following Where2Act [129], we disable collision simulation between every pair of two parts connected by an articulated joint. We also set gravity to zero to better simulate articulated objects that have an opener that needs to be opened upwards, e.g., *Boxes*. Setting gravity to zero prevents the opener to gradually fall back from open to close without any force applied. For the rendering settings, we use four cameras to get pointcloud of the target object. We set the near plane to 0.1, far plane to 100, resolution to 640×480, and the field of view to $35°$. We obtain a pointcloud from the depth images by back-projecting the depth image into pointclouds, removing the far-away (background) points, and down-sampling the point to get a total of 10K points.

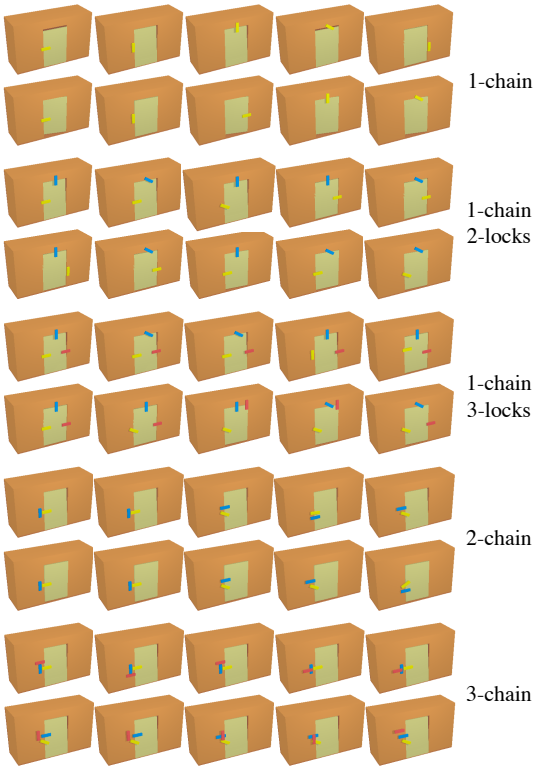


Figure 4.3: All instances from the Puzzle-Boxes Benchmark.

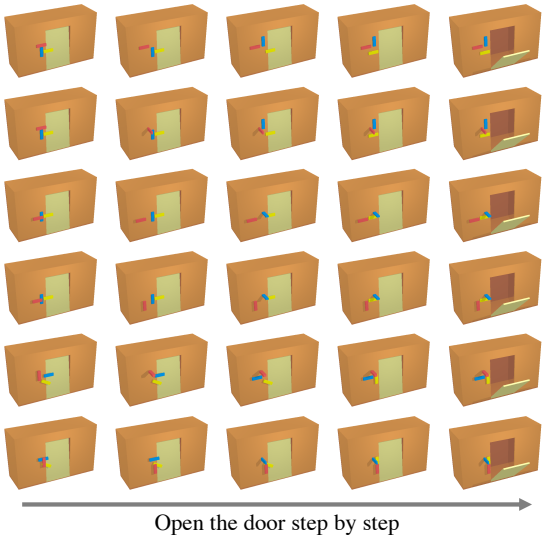


Figure 4.4: Visualizations of how the puzzle boxes will be opened by unlocking keys in 3-chain puzzle boxes.

The **PuzzleBoxes** dataset has more challenging configurations and joint dependency. We aim to create a more comprehensive version of Thorndike's puzzle box experiment [174], where a cat needs to escape a cage by exploring a locked door and finally opening it. We design the boxes with different level of difficulties by varying the number of *locks* (N^{locks}) and the number of decency *chains* (N^{chain}). As shown in Fig. 4.3, we prepared five different settings: $(N^{\text{chain}}, N^{\text{locks}}) \sim \{(1,1),(2,1),(3,1),(1,2),(1,3)\}$, where each setting has 10 different configurations (joint type, axis, and position).

In the 1-chain N-locks setups, the door is blocked by a number of sliding locks and revolving locks. To solve the task, an agent needs to figure out which locks are blocking the door and how to manipulate these locks to resolve the blocking dependency. In these 1-chain setups, we assume the locks can be solved independently. To further test whether an agent can optimally explore by only researching on task decency locks, we also include boxes with dummy locks, e.g., locks that are already open and are not blocking the doors. In the 2-chain and 3-chain setups, we test whether an agent can solve a chain of locks with dependency, i.e., the locks need to be operated in a specific order to fully unlock the door. In Fig. 4.4, we show some example design of the locks.

In both dataset, the 6-DoF action is implemented in the simulator by simulating a directed force on a 3D point, imitating actions from a suction gripper.

Table 4.2: Joint type estimation accuracy [%].

		Novel instances in training Categories						Testing categories			
		Box	Door	Microwave	Fridge	Cabinet	Mean	Safe	Table	Washing	Mean
Closed	PN2	**100.0**	43.3	97.4	72.9	69.2	76.5	55.7	56.5	45.2	54.0
	Ours	**100.0**	**85.4**	**100.0**	**98.6**	96.7	**96.1**	89.7	98.7	**100.0**	96.1
	PN2+Ours	**100.0**	80.5	90.9	**98.6**	**97.7**	93.4	**96.6**	**99.3**	93.8	**96.5**
Half-opened	PN2	**100.0**	87.5	**100.0**	99.1	**99.8**	**97.4**	**100.0**	92.2	77.4	89.9
	Ours	**100.0**	**97.6**	81.8	**100.0**	99.0	95.7	**100.0**	92.8	**100.0**	**97.6**
	PN2+Ours	92.3	90.2	**100.0**	98.6	98.6	96.0	**100.0**	**93.4**	93.8	95.7

4.3.1 Joint type estimation

We first evaluate how well H-SAUR can estimate the type, location, and limits of joints on an articulated object.

Settings. We test joint estimation performance using the PartNet-Mobility dataset with both the *closed* and *half-opened* settings. To measure the performance, we cast the problem into an eight-way classification problem where the model classifies the target joint as one of the following: four different revolute joints attached to the right, left, top, or bottom of the 3D bounding boxes for the object part, three different prismatic joints that moves along each of the X, Y, and Z axes, or a fixed joint (see Fig. 4.5).

Models. We initialize a uniform prior for H-SAUR with the eight possible joints, using 110 particles. The algorithm stops if one of the following conditions is satisfied: (1) the model has good confidence with more than 90% of the particles belong to a single class, or (2) the model interacts with the object 10 times. We compare our algorithm with a supervised learning baseline *PN2* [110], which uses PointNet++ [147] and implementation [189] as a feature extraction backbone to predict joint types given an input point cloud and segmentation masks of the object parts connected to the joint. We feed the extracted feature into a classification network which consists of two 256-dim fully connected layers and 8-dim fully connected layer that classifies the input into the eight possible joints.

We also test the combination of H-SAUR and PointNet++, which we denote *Ours + PN2*, where we initialize the hypothesis distribution of the proposed model with prediction from PN2. However, we found that PN2 can generates almost zero weight on a correct hypothesis when it becomes over-confident about a false configuration. This can cause the proposed model to fail since all the particles are initialized with the over-weighted false configuration. To handle the problem, we impose a minimum weight 1/16 on all the hypotheses to ensure all hypotheses are covered in the particle pool. We found this heuristic can significantly improve the overall performance (from 80.5% to 96.5% on *closed* setting with test categories).

Results and analysis. Table 4.2 shows the joint type estimation accuracy. Our model performs comparably with PN2 in the *half-opened* setting and significantly outperform PN2 on the *closed* setting where joint type is mostly ambiguous from vision alone. We also show, by integrating visual prior from PN2 with the proposed framework, we can improve in cases where visual prior helps significantly, e.g., in *half-opened* Microwave. We visualize the posterior over hypotheses in Fig. 4.5, where we can see that our model becomes more confident after a few interactions.

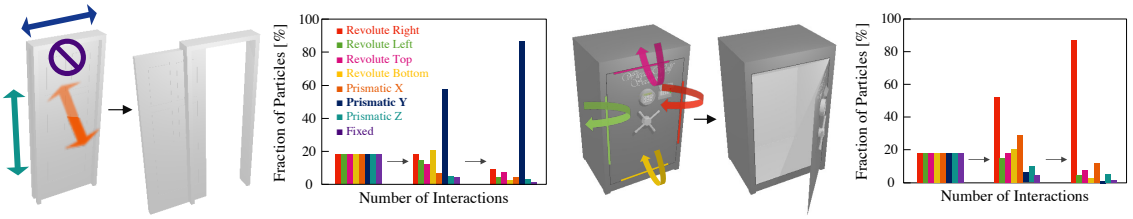


Figure 4.5: Percentage of particles representing different hypothetical configurations during interactions. The hypotheses start as a uniform distribution, then with an observed action, belief tends to aggregate on the correct joint type.

Table 4.3: Joint type estimation accuracy on noisy dynamics [%].

σ	Box	Door	Microwave	Fridge	Cabinet	Safe	Table	Washing	Mean
0.0	**100.0**	85.4	**100.0**	**98.6**	96.7	89.7	**98.7**	**100.0**	**96.1**
0.1	96.1	**94.8**	**100.0**	87.8	90.9	**95.9**	96.6	87.5	93.6
0.2	95.4	92.8	**100.0**	82.9	**100.0**	**95.9**	82.8	87.5	92.2
0.3	95.2	92.8	**100.0**	85.4	**100.0**	93.2	96.6	93.8	94.6

4.3.2 Joint type estimation under stochastic dynamics.

Settings. We further evaluate the performance of H-SAUR under stochastic dynamics by adding noise to action to imitate a stochastic dynamics. The action noise ϵ is uniformly sampled from $\boldsymbol{\epsilon} \sim [-\sigma, \sigma]^6$ thus the action on the stochastic dynamics will be $a_t^{\text{noise}} = a_t + \boldsymbol{\epsilon}$. We evaluate H-SAUR with $\sigma \in \{0, 0.1, 0.2, 0.3\}$, where $\sigma = 0$ corresponds to *Ours* in Table 4.2.

Results and analysis. Table 4.3 shows the results on different noise levels. At its most extreme, the noise is sampled from a uniform range of width 0.6 meters, which is the equivalent to the size of the articulated part in many cases, yet adding this noise has little effect on joint estimation performance. This is partly because our method is a probabilistic framework, thus it can handle any uncertainty including stochastic dynamics, part segmentation, action noises, etc.

4.3.3 Action Proposal and Affordance Map

We next measure how well the H-SAUR model can use its estimates of joint properties to estimate whether an action will be effective on the PartNet-Mobility dataset.

Settings. To evaluate all models, we collect 10,000 interactions on the *closed* setting by randomly sampling a point belonging on a movable part and applying a force with a uniformly distributed direction on the surface of the 3-d unit sphere. An action is labeled as "success" if it causes the joint to move more than 5% of its full range. We counterbalance "success" and "failure" interactions in the final test set. We use two metrics to evaluate the models: (1) *Binary classification Accuracy* which is the proportion of actions correctly predicted as success or failure, and (2) *Distance Prediction* which measure the ℓ_1 distance between the predicted point translation and the ground truth.

Baseline. We compare our model with the state-of-the-art articulated object manipulation algorithm, Where2Act (W2A) [129], which takes the pointcloud of an articulated

Table 4.4: The proportion of the part opened [%] with fifteen testing-time interactions.

	Novel instances in training categories						Testing categories			
	Box	Door	Microwave	Fridge	Cabinet	Mean	Safe	Table	Washing	Mean
W2A (100K)	65.0	47.3	49.6	50.7	54.8	53.5	42.6	43.9	70.0	52.2
W2A+HP (100K)	**96.2**	51.3	91.0	**98.4**	92.9	85.9	72.8	72.0	94.3	79.7
Ours (0.01K)	83.9	**90.2**	99.0	94.8	95.6	92.7	82.8	98.0	93.8	91.5
Ours + PN2	87.2	86.0	**100.0**	94.7	**97.1**	**93.0**	**85.6**	**99.0**	**97.7**	**94.1**

Setting	1-chain	2-chain	3-chain	1-chain 2-locks	1-chain 3-locks
Random	23.3	6.7	0.0	13.3	3.3
Heuristic	36.7	13.3	0.0	23.3	10.0
CURL	33.3	0.0	0.0	0.0	0.0
Ours	**96.7**	**86.7**	**80.0**	**93.3**	**86.7**

Table 4.5: Manipulation performance (%) for solving PuzzleBoxes averaged from 3 runs.

object as input to predict an effectiveness score for all points. To train the model, we collect $\{10\text{K}, 100\text{K}\}$ number of counter-balanced interactions using the same procedure as above. For a fair comparison, we collect both the testing and training data from only movable links by applying a segmentation mask when sampling the position to interact as our method assumes segmentation of the parts is given.

Results and analysis. We show the results in Table 4.6. Our method significantly outperforms the baseline, despite being 1000 times more sample efficient. We show qualitative results of distance prediction by H-SAUR in Fig. 4.6.

4.3.4 Manipulation

Next we evaluate the estimated joints for manipulation task on the PartNet-Mobility Dataset.

Settings. The task is to open the movable parts as much as possible from completely closed setting within $N^{\max} = 15$ interactions. Our method uses first $N^{\text{int}} = 10$ interactions to estimate the joint type, and the rest to manipulate the object while the baseline models use all $N^{\max}$ interactions to open the movable parts. For evaluation, we measure the proportion of the part opened $r = \max_{t \in \{1,\dots,N^{\max}\}}(\theta_t - \theta^{\text{init}})/(\theta^{\max} - \theta^{\text{init}})$, where $r = 1$ means the fully opened target part.

Baselines. We again use Where2Act as the baseline for this experiment. We also add Where2Act + HP, which employs an additional heuristic that filters out actions that has a larger than 90 deg angle with last-step action as done in [197]. This heuristic helps to avoid sequences of back-and-forth actions.

Results and analysis. Table 4.4 shows our method significantly outperforms Where2Act in all categories and performs better than Where2Act+HP in most settings except for boxes and fridge. We found these two categories are simpler in the sense that all boxes open in the same direction (upward), and so do the fridges (to the left), so it is easy for Where2Act to overfit to a single action. We can also see that the performance of our method, when

Table 4.6: Affordance prediction performance.

Method	Novel instances in training categories						Testing categories			
	Box	Door	Microwave	Fridge	Cabinet	Mean	Safe	Table	Washing	Mean
Binary classification accuracy [%] ↑										
W2A (10K)	68.6	59.7	70.6	70.1	69.7	67.7	68.5	63.9	60.3	64.2
W2A (100K)	75.9	60.2	81.1	71.3	70.0	71.7	74.1	53.5	66.1	64.6
Ours (0.01K)	**96.4**	**79.5**	**97.8**	**93.0**	**93.0**	**91.9**	**93.3**	**97.4**	**91.3**	**94.0**
Distance prediction error ↓										
W2A (10K)	0.051	0.074	0.040	0.068	0.062	0.059	0.057	0.053	0.076	0.062
W2A (100K)	0.049	0.072	0.032	0.063	0.057	0.055	**0.051**	0.061	0.067	0.059
Ours (0.01K)	**0.009**	**0.036**	**0.013**	**0.040**	**0.026**	**0.025**	0.055	**0.016**	**0.029**	**0.033**

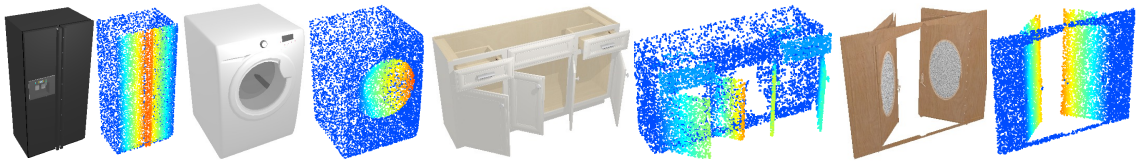


Figure 4.6: Visualizations of distance prediction. The warmer color shows larger deformations.

combined with the learned PN2 model, slightly improves. It shows H-SAUR alone is already robust to skewed prior, and one can easily improve its performance by incorporating good prior from vision models.

4.3.5 Ablation on different scoring methods to update hypotheses.

In this section, we compare two different scoring functions to update the hypotheses: the chamfer distance used in our method (described in Sec. 4.2.3) and the cosine similarity referring to [62]. The cosine similarity β_k measures the cosine of the angle between the displacement of the real object $\boldsymbol{d}$ and the displacement of a hypothetical object. The direction $\boldsymbol{d}$ is calculated by $\boldsymbol{d}_t = \boldsymbol{p}_{t+1} - \boldsymbol{p}_t$, where $\boldsymbol{p}_t$ is the center position of the observed point cloud O_t^j as $\boldsymbol{p}_t = \frac{1}{\|O_t\|}\sum_{\boldsymbol{x}\in O_t^j}\boldsymbol{x}$. The cosine similarity is computed using the direction as $\beta_k = \arccos\left(\frac{\boldsymbol{d}_t \cdot \hat{\boldsymbol{d}}_t^{(k)}}{|\boldsymbol{d}_t||\hat{\boldsymbol{d}}_t^{(k)}|}\right)$. We then use β_k for the likelihood of the k-th particle $s^{(k)}$ as $w_k = (\beta_k + 1)/2$. This formulation results in an increase of the likelihood when a hypothetical object successfully imitates the movement of the real object. Following [62], we assign $w_k = 1$ when both objects do not move, and $w_k = 0$ in the case where only one of the two objects moves, where we assume that such a situation can happen when the hypothetical configuration is wrong. To wrap up, the cosine similarity-based likelihood function can be formulated as follows:

$$w_k = \begin{cases} (\beta_k + 1)/2 & \text{if } \boldsymbol{d}_t \neq \boldsymbol{0} \text{ and } \hat{\boldsymbol{d}}_t^{(k)} \neq \boldsymbol{0} \\ 1 & \text{if } \boldsymbol{d}_t = \boldsymbol{0} \text{ and } \hat{\boldsymbol{d}}_t^{(k)} = \boldsymbol{0} \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

Table 4.7 shows the comparison of the two different scoring functions to update hy-

		Box	Door	Microwave	Fridge	Cabinet	Safe	Table	Washing	Mean
Closed	Chamfer distance	**100.0**	**85.4**	**100.0**	**98.6**	**96.7**	**89.7**	**98.7**	**100.0**	**96.1**
	Cosine similarity	78.4	83.0	69.2	73.2	72.7	73.5	74.0	82.8	76.0

Table 4.7: Comparison of different scoring methods to update hypotheses for joint type estimation accuracy [%].

potheses for joint type estimation experiments. It clearly shows that chamfer distance outperforms the cosine similarity. We found that cosine similarity works poorly especially when the joint state of the real object is close to the upper or lower limit. In such situations, only either hypothetical or real object moves and results in $w_k = 0$, and filtered out from the particle pool. Chamfer distance, however, is robust to the wrong joint state or joint upper/lower limit, because it still returns a reasonable value even if the estimated state or joint limits is wrong.

4.3.6 PuzzleBoxes

Finally, we evaluate H-SAUR on a novel benchmark *PuzzleBoxes*. The task is to open a door outward more than 60° within 100 interactions. However, opening this door requires first moving other "locks" that restrict the range of motion of the door, as shown in Fig. 4.4.

RL Baselines. To the best of our knowledge, none of the prior learning-based approaches can solve this long-term manipulation problem without exhaustively interacting with the objects before deployment time. To show this, we train an RL agent with CURL [102], a state-of-the-art image-based RL algorithm that uses contrastive learning to acquire a good image representation. The following summarize the settings for training the RL agent:

State: For a fair comparison to H-SAUR, which requires part segments and 3D information as point cloud, we define the state of the environment is two frames of RGBD image $I^{RGBD} \in \mathbb{R}^{100\times100\times8}$. The RGB image corresponds to part segments because all parts in PuzzleBoxes have unique color (see Fig. 4.3), and the depth image gives the 3D information to an RL agent.

Action: The action of the agent consists of an interaction position and direction. The position is defined by 2D continuous value $a^{\text{pos}} \in [-1, 1]^2$, and we find the closest pixel in the RGBD image and then find the corresponding 3D position to the pixel. This enables to make the action space smaller, and makes the RL agent easier to solve the task. The interaction direction is defined by 3D continuous values $a^{\text{dir}} \in [-1, 1]^3$, and normalized so that it will be a unit vector.

Rewards: Since training an RL agent entirely from a sparse reward is too hard to solve, we define a shaped reward to enhance exploration, which consists of changes in position of any joint $r^{\text{shape}} = \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|$, where $\boldsymbol{\theta}$ is a joint state vector that consists of all movable joints.

Terminal conditions: An episode terminates with the same condition with other methods as H-SARU and the other baselines; an agent opens the door outward for more than 60°, or the total step of an episode is over 100.

Training: The hyperparameters of the CURL [102] agent is the same used in the original paper. We use a random policy to collect 1K transitions to a replay buffer before training an RL agent, and then train the RL agent for 10K timesteps. The configurations of PuzzleBoxes for train and test are different; we use 7 configurations for training and 3 configurations for testing, and we train RL agents for each PuzzleBoxes type separately.

Naive Baselines. We also compare our algorithm to two learning-free baselines: (1) *Random:* a policy that uniformly sample a movable link and apply randomly sampled action on it at each time step, (2) *Heuristic:* a policy that selects an action in the same way as *Random*, but keeps applying the same action if the object moves at the previous step.

Results and analysis. We show the results in Table 4.5. We can see that the RL baseline trained with 10K timesteps, which corresponds to 100x more timesteps than ours, performs poorly on all but the simplest levels. Deep RL algorithms generally needs enormous amount of interactions to learn, and can fail drastically when the agent is allowed to have a limited number of interactions. Aside from poor sample efficiency, most learning-based policies would generate similar actions (drawn from a fixed distribution) given similar observations (as PuzzleBoxes are designed to look similar but have different joint axes), since most policies only take one or a few past observations as inputs and do not update action distributions from past failed interactions. Even baselines that use knowledge of the problem structure (both *Random* and *Heuristic*) perform poorly at all levels. In contrast, H-SAUR can solve even the most complex levels far above chance.

4.4 Conclusion

In this work, we propose a physics- and uncertainty-aware exploration framework, H-SAUR, that can manipulate diverse articulated objects in circumstances where visual inputs do not uniquely specify the state. We show H-SAUR can open complex puzzle boxes requiring several steps to solve by interactively updating hypotheses over different articulation structures. We also show the proposed model outperforms baselines by a large margin, highlighting the importance of quick behavior adaption through test-time exploration. Our model operates directly on pointcloud segments without the need of detailed tracking using AR tags or any other tracking system, which increases its chance to transfer to a real-world setup. Current results show the model is robust to mismatch between the object of interest and the reconstructed virtual objects from the pointcloud segments.

We note that more work is needed to extend H-SAUR to manipulate arbitrary real-world articulated objects. First, our current model cannot handle articulated joints with arbitrary joint axis, e.g., a door that rotates with a tilted joint, or joint types that have not been prespecified in its hypothesis space. This problem can potentially be addressed using motion-based kinematic prediction [85] to propose new hypothesis to include in the prior. Second, we assume part segments are given. In ongoing work, we are investigating models that jointly infers object parts and their articulations. Third, we assume a force can be applied to any point on an object from any direction in order to separate *reasoning* about joints from *manipulating* them. While this is roughly similar to using a suction gripper as in [197, 202], we plan to explore practical constraints imposed by a real robot's geometry,

gripper, etc. in future work.

Nonetheless, H-SAUR demonstrates a promising avenue for systems that can reason about articulated objects, manipulate them, and update beliefs in real time.

Chapter 5

Understanding Objects' Geometries through Interactions using Tactile Sensing

Humans rely on tactile sensing to monitor and control manipulation tasks during interactions with the environment. Tactile sensing allows us to observe and respond to contact forces, adapt to object slip during grasping, and perform various perception tasks to build models of the environment. Even in situations where visual observation is not possible, tactile sensing enables us to interpret different types of interactions with the environment. For instance, we can locate objects in cluttered environments even in the absence of visual cues, identify the correct key for a lock by feeling the different options, or determine the type of video port (e.g., HDMI) on a monitor screen by touch alone. It has been the long-standing goal of robotics to imitate such intelligent behavior during manipulation tasks. Motivated by this goal, we present an interactive perception method for robots that utilize vision-based tactile sensors to construct reliable models for part mating.

A lot of manufacturing tasks could be decomposed into a sequence of insertion tasks. Object insertion is a well-studied contact-rich manipulation task in robotics [41, 160]. However, the task becomes extremely challenging when the geometry of the mating objects is unknown. Also, this can make the task of part mating significantly complex as the uncertainty in geometry can limit the ability to understand any possible contact formation between the parts [41]. This complexity is further amplified in assembly tasks that require precise geometric information, as tolerances between mating parts become critical. Achieving the required level of precision in manufacturing tasks can be challenging when relying solely on vision-based algorithms.

To address these challenges, we propose a method that leverages vision-based tactile sensors located at the robot's gripper for precise perception in these tasks. We present an interactive perception method, *Tactile-Filter*, using vision-based tactile sensors for estimating part correspondence for part mating, i.e., to estimate which parts fit into each other using tactile sensors in the absence of any vision sensing. We train a deep learning model to predict correspondence between the correct mating parts, observed using a tactile sensor. In the presence of partial observation or non-unique contact patch, we

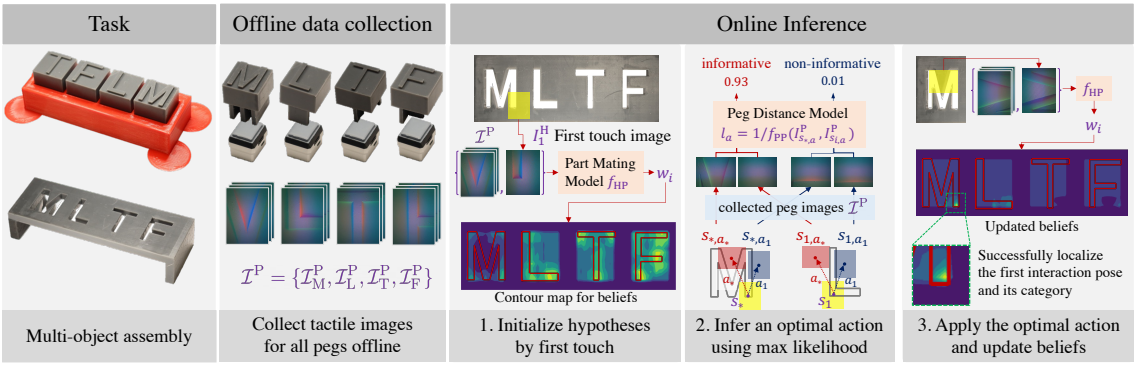


Figure 5.1: Overview of the proposed *Tactile-Filter*. As shown in the figure, we consider the task of part mating without any prior knowledge of 3D mesh of objects and which objects fit together. We assume that the robot has access to a collection of tactile images for the set of pegs as shown in figure (Offline data collection). During inference, the robot tries to identify which peg would fit in a given hole by the proposed *Tactile-Filter*. An initial set of hypotheses (denoted by $s \in \mathcal{S}$) is generated using the tactile image from the first touch and a trained part mating model, which predicts the correspondence between parts that fit together. We compute an optimal action for sampling the next image on the hole surface in order to minimize the uncertainty of the current estimate using a maximum likelihood approach. This is also illustrated in the figure, where, given an initial touch, we can select an optimal action that maximizes uncertainty reduction. This method allows us to find the peg for the right fit, as well as localize the hole (as we finally get the correct hypothesis) while minimizing the number of interactions during the task. (MLTF stands for Maximum Likelihood Tactile Filter). [Best viewed in color]

make use of an interactive perception approach, similar to that of Chapter 4, to aggregate the information from multiple touches and improve the estimate of the correct fit for a given hole. To minimize the number of interactions between the robot and the object, a maximum likelihood-based action selection method is used during the proposed interactive perception. The proposed method is tested on several different test environments with objects of different shapes and sizes that are not used in training the model to show the generalization of the proposed approach. Figure 5.1 shows the abstract idea of the proposed interactive perception method.

Contributions: This chapter has the following contributions:

1. We present a part mating problem that deals with objects of unknown shapes, aiming to identify and estimate the pose of mating parts using minimal number of interactions. To address this problem, we present a novel approach called *Tactile-Filter*, which combines contrastive learning, particle filter, and tactile sensing for part mating.

2. Through our experiments conducted on novel objects, we demonstrate that our proposed method effectively resolves uncertainty by iteratively updating its belief during interactions. We demonstrate the ability to generalize to objects not encountered during training. Furthermore, we introduce an action selection method within our approach, which leads to significant improvements in efficiency.

At its core, we combine the interactive perception approach proposed in Chapter 4 and tactile sensing to solve a real-world problem. It should be noted that for the sake of brevity

and clarity of presentation, we will use the word *peg* for the male part and *hole* for the female part in this chapter.

5.1 Related Work

Vision-based tactile sensors have attracted a lot of attention recently [173, 199]. These sensors provide a high-resolution capture of the contact patch during contact formation and thus can help in localization of contacts, detection of slip, etc [42, 118, 173]. Consequently, vision-based tactile sensors have been used for a lot of control and perception tasks [41, 75, 95, 157, 159]. Most of these methods make use of displacement of the contact patch to recover a signal indicating slip which can be used for control of the manipulation task. Similarly, there has been prior work that uses these sensors for object classification and slip stabilization using feedback from these sensors [24, 200], or for learning visuotactile servoing [25, 61].

Several studies have focused on estimating object pose using tactile sensors which can be broadly categorized into two directions. The first group of studies addresses pose estimation from a single touch [15, 49, 94, 108], employing regression [49, 108] and contrastive learning [15]. However, these methods exhibit limitations when applied to large objects or objects with non-unique contact patches. The second group of studies focuses on pose estimation of large objects, often utilizing particle filtering techniques to narrow down the distribution of possible poses [23, 94, 116, 170]. However, these methods face challenges when applied to real objects due to factors such as low-dimensional sensors [116], the requirement of a large number of interactions for training interaction policies [94] or collecting samples [170]. Furthermore, some approaches assume they have access to 3D models of objects that limits their applications to unknown agents [15, 23, 94, 170]. In contrast to these two sets of works, we assume that the shape of the objects is not known a priori, and allow only a limited number of interactions (a maximum of 10) by generating informative actions that effectively reduce ambiguity. Furthermore, our method not only estimates the pose of the object, but also identifies the type of object from multiple candidates, i.e., simultaneously solving classification and pose estimation problems, adding another layer of complexity to the problem.

There exists another category of work that integrates vision and tactile sensors to estimate object pose [25, 39, 49, 116]. By leveraging the capabilities of vision sensors, these methods can reduce the number of interactions required to estimate object pose by narrowing down the distribution of possible poses. While our study specifically focuses on object pose estimation using vision-based tactile sensors, our method is not mutually exclusive with the methods that combine vision and tactile sensors. By integrating both modalities, we can leverage the strengths of each sensor type and potentially improve the accuracy and efficiency of object pose estimation.

Our problem setup is relevant to the interactive perception literature, where the goal is also to interact with the objects and update iteratively on the estimation of the states [16, 165, 193]. While previous work focuses mostly on state estimation from raw visual perception [16, 62], point clouds [129, 141, 190], or 3D states of the objects [136]. In contrast to these

previous works, we consider state estimation using tactile sensors.

Our work is also related to perception during insertion or part mating. Vision is mostly insufficient to perform a lot of insertion tasks due to the precision required during these tasks [41, 104]. Consequently, there has been a lot of work making use of tactile and/or wrench measurements on contact formation between the mating parts. The idea behind most of these tasks is to make use of tactile measurements and a feedback mechanism to iteratively correct the pose error between the mating parts [41, 43, 84]. In all these methods, geometric information is not explicitly used. Furthermore, they do not explicitly update the uncertainty in measurements. In contrast, we present a method where the robot can make use of the geometric information upon contact formation using tactile sensors to iteratively estimate the part correspondence as well as precise localization.

5.2 Problem Statement

In this section, we present a formal statement of the problem which is studied in this chapter. We also motivate the problem by discussing some common scenarios in which the proposed problem could arise and the proposed method could be useful.

We consider the task of perception during part mating while performing automated assembly. We consider scenarios where the robot cannot use a vision sensor to perceive the target object to perform the desired mating task. Such situations could arise in tasks where a robot has to assemble a product where occlusions are created by other parts (e.g., consider the assembly of an electronic board). Apart from occlusion, these tasks could also require precision in pose estimates which might be very difficult to obtain using vision alone. To formalize the problem, we define the robot's task as identifying the correct peg from a known set of possible choices by multiple observations (touches) of the hole using the vision-based tactile sensor(s). The goal here is to design algorithms that can identify the correcSo,peg with minimum physical interaction with the hole using a tactile sensor. In the proposed study, we make the following assumptions:

1. The possible number of pegs for the part mating task is fixed and known as a priori.
2. The rough location of the target hole is known so that the robot can establish initial contact with the part, and it does not need to perform this rough localization using touch.
3. The robot can collect a dense set of tactile images for all the candidate pegs by touching each of the pegs at various different locations and orientations, prior to experiments.

The first assumption is not restrictive as we will generally have a limited number of parts to assemble. The second assumption is also very easily met using common vision methods with rough precision in localization. The third assumption would require that the robot has access to a detailed geometric model of the possible pegs observed using tactile sensors. This would require that the robot performs some exploration to collect this data. This

assumption is required as the size of the tactile sensor may be small compared to the size of the objects that the robot is interacting with.

We focus on the setup where the target objects are larger than the size of a tactile sensor such that the entire object can not be observed using a single touch, or a single touch can result in non-unique observations (consider when parts of the objects could be similar). Such problems would require multiple touches and a method to aggregate the data from multiple touches. For example, consider the task of inserting pegs into the shape of alphabet letters that are larger than the sensor size (see Fig. 5.4). If the target hole is the letter "A", whether we can be certain that it is indeed an "A" with a single touch depends on where we land our finger. For instance, if the first touch is made on the horizontal line segment of the letter "A," this feature may also be present in other alphabets such as "B," "D," "E" and so on. However, by making contact with features unique to the letter "A," such as the lower left intersection, the probability of other candidates can be dramatically reduced. So, the question is how do we aggregate information from multiple touches, and how do we select places to touch that will maximize the information so as to reduce required interactions.

5.3 Method

We propose *Tactile-Filter*, an uncertainty-aware interactive perception method to identify the correct peg from a candidate set that fits into a given hole for assembly using tactile sensors. At the core of the framework is a feature-matching model that computes the probability that a peg can pair with a hole by measuring the distance between the corresponding tactile images in a joint feature space. Using the feature matching model, we can construct the three critical steps in an interactive perception method, such as the one in Chapter 4: (1) we can initialize a set of possible hypotheses from the first touch by comparing the tactile image on the hole with the candidate images for the pegs collected before the experiment, (2) we then sample the most probable hypothesis and generate an action that maximizes uncertainty reduction, and (3) we apply the inferred action to the real system and update beliefs about the shape (class) and pose of the target hole. In this section, we detail each of these components.

5.3.1 Learning to find mating part

Given a pair of peg and hole images, we train a model that maximizes the similarity score if the image for the hole corresponds to the image for the peg. To this end, we use a contrastive learning framework [27, 63] to learn the feature space, similar to the work MoCo-v3 [29].

MoCo-v3 has two encoders, f_q and f_k, with output vectors q and k. The goal of learning is to find the key vectors k that correspond to the query vectors q. In our case, we consider the query vectors q to be the vectors from images of the holes and learn to maximize the similarity score between q and the vectors computed from the corresponding peg images k, while minimizing the similarity between the query vectors and a set of vectors from the negative peg images $\{k^-\}$. In MoCo-v3, this is formulated by minimizing the InfoNCE

loss [137]:

$$\mathcal{L} = -\log \frac{\exp\left(q \cdot k^{+}/\tau\right)}{\exp\left(q \cdot k^{+}/\tau\right) + \sum_{k^{-}} \exp\left(q \cdot k^{-}/\tau\right)}, \tag{5.1}$$

where τ is the hyperparameter. More details can be found in Chen *et al.* [29], implementations in [35], and the training procedure is shown in Fig. 5.3. We denote the model used for measuring similarities between the hole images and peg images as f_{HP}, and name it *part mating model*. Additionally, we train a model, denoted as f_{PP} and referred to as the *peg distance model*, to calculate the similarity between peg images. This f_{PP} will be utilized to produce informative actions (as detailed in Section 5.3.3).

5.3.2 Generating hypotheses

In cases where only a partial shape of a hole can be observed using a tactile sensor, multiple corresponding choices for peg may exist. Thus it might be hard or impossible to determine the object with a deterministic approach. To address this uncertainty, we explicitly generate and maintain a set of hypotheses representing potential candidate choices using a particle filter.

Since the class of the target hole and its orientation and location is unknown (Sec. 5.2), we generate a set of hypotheses $\mathcal{S}$ where each hypothesis s_k is a quadruple: $s_k = (c_k, x_k, y_k, \theta_k)$, where c_k represents the possible object categories $c_k \in \mathcal{C}$, and x_k, y_k, θ_k are the SE(2) relative planar displacements from the center of the object. An example is shown in Fig. 5.1 and is also explained in Fig. 5.2.

One can initialize the particle set by sampling from a uniform distribution within a reasonable range (e.g., c_k from candidate categories, x_k from $[-X_{\mathrm{MAX}}, X_{\mathrm{MAX}}]$, which is the lower and upper limits of reasonable sizes of the peg, etc.). However, it will be inefficient as the sampling space becomes huge as the target object becomes larger, and/or the number of candidate categories increases. As an alternative, we initialize the set of particles after obtaining the first tactile image of the target hole $I^{\mathrm{H}}_{t=1}$ (which we denote as I^{H}_{1}) by utilizing the previously collected set of peg images $\mathcal{I}^{\mathrm{P}}$ and the pre-trained part mating model f_{HP}. Specifically, we compute the similarities, w_i, between the observed initial tactile image for the hole I^{H}_{1} and each tactile image of the peg I^{P}_{i} from the set of peg images $\mathcal{I}^{\mathrm{P}}$ as:

$$w_i = f_{\mathrm{HP}}(I^{\mathrm{H}}_{1}, I^{\mathrm{P}}_{i}). \tag{5.2}$$

We sample a particle proportional to this likelihood. Therefore, the probability of a particle given the initial hole image can be written as:

$$p(s = i | I^{\mathrm{H}}_{1}) = \frac{w_i}{\sum_{i \in \{1,\ldots,|\mathcal{I}^{P}|\}} w_i} \tag{5.3}$$

where, i is the index of the set of previously collected peg images as $i \in \{1, ..., |\mathcal{I}^{\mathrm{P}}|\}$ (see Fig. 5.1). We then initialize the set of hypotheses by independently sampling K particles with the above distribution. It is noted that the above distribution is a categorical distribution over all peg images.

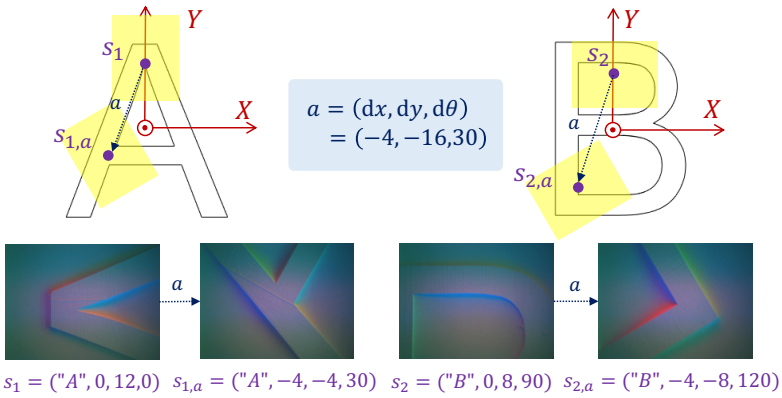


Figure 5.2: This picture defines particles and actions available to the robot. A particle s is defined as a tuple consisting of the class of the object and a pose in SE(2) w.r.t. a frame attached to the center of the object. Each particle s_i is associated with the corresponding image $I^{\mathrm{P}}_{s_i}$ observed via the tactile sensor at that location. An action a is equivalent to the transform in SE(2) applied to a particle s_i. The pose of a particle obtained by applying an action a can be obtained by applying the transform in SE(2) to the pose of the particle s, and we denote it as $s_{i,a}$. Thus, an action a will result in the observation of the contact patch $I^{\mathrm{P}}_{s_{i,a}}$ at the new pose corresponding to the particle $s_{i,a}$.

5.3.3 Selecting informative action

In order to efficiently determine the category and pose of the target hole, we aim to calculate an optimal action that can maximize uncertainty reduction. While it is possible to compute such an optimal action by maximizing information gain against all possible peg images, it necessitates the integration of all latent variables, making it computationally infeasible within a reasonable time frame. As an alternative, we utilize the existing hypothesis set $\mathcal{S}$ to enhance the sample efficiency, similarly to H-SAUR in Section 4.2.

We sample the most probable hypothesis from the current set of particles $s_* = \arg\max_{s_k \in \mathcal{S}} w_k$ and determine the optimal action by simulating it on a set of previously collected peg images $\mathcal{I}^{\mathrm{P}}$ (see Fig. 5.1). The action $a = (\mathrm{d}x, \mathrm{d}y, \mathrm{d}\theta)$ is represented as a transform in SE(2) to the pose of the particles, and we denote the particle s_k with the updated pose by applying the action a as $s_{k,a}$. With this updated pose and the set of peg images $\mathcal{I}^{\mathrm{P}}$, we can also obtain the peg image when applied to the action a, which we denote $I^{\mathrm{P}}_{s_{k,a}}$. If there is no corresponding pose in the collected set of images, we assign an empty image which is a tactile image without any contact $I_{\mathrm{NoContact}}$. We visually explain the definition of particles and actions in Fig. 5.2.

Given a current most-likely hypothesis, the next optimal action can be selected by finding the most informative action. To do that, we compute the distance between the tactile images obtained by applying any possible action to the most probable hypothesis and the remaining particles in set $\mathcal{S}$. The action that maximizes the sum of this distance over all the particles is selected as the optimal action. Such an action is favored only when the peg image of the sampled particle is close to the hole image, while other images have a greater distance when applying the same action to all the other particles. More concretely,

Algorithm 6 *Tactile-Filter*

Input Number of candidate pegs N^{P}. A set of dense tactile images for each peg categories $\mathcal{I}^{\mathrm{P}} = \{\mathcal{I}_1^{\mathrm{P}}, \cdots, \mathcal{I}_{N^{\mathrm{P}}}^{\mathrm{P}}\}$, number of particles K, maximum number of interactions $N^{\max}$, pre-trained part mating model f_{HP} and peg distance model f_{PP} (Sec. 5.3.1), threshold to stop iteration δ^{prob}

Output Mating peg category and its displacement in x, y, θ coordinate from the center of the peg

Touch and observe the first hole image I_1^{H}
Compute similarity score w_i between the hole image I_1^{H} and a peg image $I_i^{\mathrm{P}} \in \mathcal{I}^{\mathrm{P}}$ as $w_i = f_{\mathrm{HP}}(I_1^{\mathrm{H}}, I_i^{\mathrm{P}})$
Initialize a particle pool $\mathcal{S} = \emptyset$
for $k \leftarrow 1$ to K **do**
    Sample a particle s_k from the set of the peg images according to the categorical distribution defined in Eq.(5.3)
    Add the sampled particle $s_k = (c_k, x_k, y_k, \theta_k)$ to the particle pool: $\mathcal{S} \leftarrow \mathcal{S} \cup s_k$.
end for
for $t \leftarrow 2$ to $N^{\max}$ **do**
    Select the most probable particle $s_* = \arg\max_{s_k \in \mathcal{S}} w_k$
    Infer the optimal action a_* according to Eq. (5.5) using s_* and $\mathcal{S}$.
    Move the robot with the inferred action a_* and get the tactile image I_t^{H}
    for $k \leftarrow 1$ to K **do**
        Compute importance weight for the particle $w_k = f_{\mathrm{HP}}(I_t^{\mathrm{H}}, I_{s_k}^{\mathrm{P}})$
    end for
    Compute the posterior distribution defined in Eq.(5.6)
    Re-sample K particles from $\mathcal{S}$ using the posterior distribution
    Compute updated posterior $p_t(c|I_{1:t}^{\mathrm{H}})$ of each peg category from the particles as defined in Eq. (5.9)
    if $\max_{j \in \{1,\ldots,N^{\mathrm{P}}\}} p_t(c_j) > \delta^{\mathrm{prob}}$ **then**
        Break the current loop
    end if
end for
Select the most probable particle $s_* = (c_*, x_*, y_*, \theta_*) = \arg\max_{s_k \in \mathcal{S}} w_k$ (to get the localization estimate, i.e., x_*, y_*, θ_*)
return Object category c_* and its displacement x_*, y_*, θ_*.

we define the likelihood of an action as

$$l_a = \sum_{s_i \in \mathcal{S} \setminus \{s_*\}} 1/f_{\mathrm{PP}}(I_{s_{*,a}}^{\mathrm{P}}, I_{s_{i,a}}^{\mathrm{P}}). \tag{5.4}$$

The optimal action can be then selected by maximizing the likelihood as:

$$a^* = \arg\max_{a \in \mathcal{A}} l_a, \tag{5.5}$$

where $\mathcal{A}$ is a set of actions with which the tactile sensor can observe the peg after applying the action. This action set $\mathcal{A}$ can be obtained by calculating the L1 pixel distance between the tactile image of the peg after applying the action $I_{s_{*,a}}^{\mathrm{P}}$ and the peg image that does not have contact $I_{\mathrm{NoContact}}^{\mathrm{P}}$, and see if it exceeds a threshold as $\mathbb{I}(\|I_{s_{*,a}}^{\mathrm{P}} - I_{\mathrm{NoContact}}\|_1 < \delta^{\mathrm{act}})$.

This procedure is visually depicted in Fig. 5.1, where the optimal action a^* is shown to generate a more distinct contact patch (observable through touch) when applied to all particles (s_* and s_1 in the figure). On the other hand, the non-informative action a_1 produces similar contact patches that would not effectively disambiguate the current belief.

5.3.4 Update hypotheses

After obtaining the optimal action a_t at time step t, we apply it on the real robot and observe the tactile image of the hole I_t^{H}. We then update the probability of each hypothesis by comparing the observed hole image I_t^{H} and the peg images from the current hypotheses $I^{\mathrm{P}}_{s_{k,a_t}}$:

$$\begin{aligned} p(s_t|I^{\mathrm{H}}_{1:t}, a_{1:t-1}) &\propto p(I_t^{\mathrm{H}}|s_t)p(s_t|I^{\mathrm{H}}_{1:t-1}, a_{1:t-1}) \\ &\approx \sum_{k=1}^{K} w_k p(s_t|I^{\mathrm{H}}_{1:t-1}, a_{1:t-1}), \end{aligned} \tag{5.6}$$

where $w_k = \frac{f_{\mathrm{HP}}(I_t^{\mathrm{H}}, I^{\mathrm{P}}_{s_{k,a_t}})}{\sum_{k=1}^{K} f_{\mathrm{HP}}(I_t^{\mathrm{H}}, I^{\mathrm{P}}_{s_{k,a_t}})}$ is the likelihood (weight) for the hole image to match with the peg image of the k^{th} particle. The second term can be written as:

$$p(s_t|I^{H}_{1:t-1}, a_{1:t-1}) = \sum_{s_{t-1}\in\mathcal{S}} \underbrace{p(s_t|s_{t-1}, a_{t-1})}_{\text{forward dynamics}} \underbrace{p(s_{t-1}|I^{H}_{1:t-1}, a_{1:t-1})}_{\text{obtain through recursion}}, \tag{5.7}$$

where the first term represents the deterministic forward dynamics. Given the particle is in state s_k at time step t, the dynamics can be expressed as:

$$p(s_{t+1}|s_t, a_t) = \begin{cases} 1 & \text{if } s_{t+1} = s_{k,a_t} \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

The second term in Eq.(5.7) is initialized with the prior defined in Eq. (5.3) and can be obtained through recursion. We update the distribution of the particles by regenerating a new set of particles through weighted sampling based on w. The full algorithm is shown in Alg. 6.

5.3.5 Terminal condition

After updating the posterior of the particles with Eq. (5.6), we compute the posterior probability of each peg category as follows:

$$p_t(c|I^{H}_{1:t}) = \frac{\sum_{s_k\in\mathcal{S}} \mathbb{I}(s_k \in c)}{|\mathcal{S}|}, \tag{5.9}$$

where $\mathbb{I}(s_k \in c)$ is an indicator function that returns 1 only when the category of the particle c_k belongs to the category c. The algorithm terminates when the majority of particles belong to a specific class, indicated by $\max_{j\in\{1,\ldots,N^{\mathrm{P}}\}} p_t(c_j) > \delta^{\mathrm{prob}}$, which is a user-specified parameter for termination.

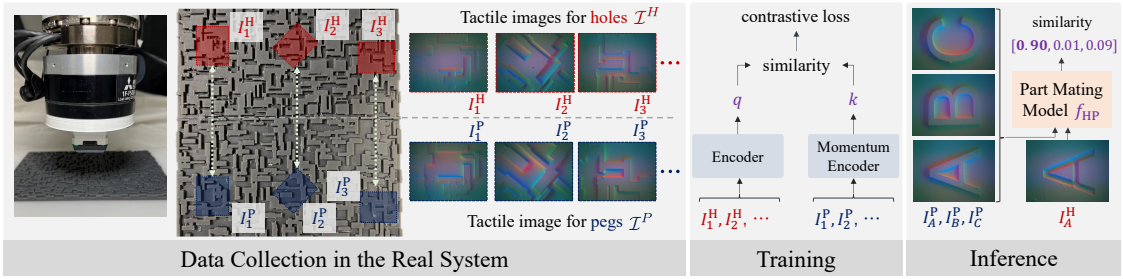


Figure 5.3: Data Collection, Training, and Inference of the part mating model: The left block depicts the data collection process using the *MAZE* board that features various shapes, including hole and peg shapes in the upper and lower halves, respectively. This board is placed on the robot platform and the robot arm equipped with a tactile sensor at the tip of the wrist makes contact with the board to collect data denoted as $\mathcal{I}^{\text{H}}$ and $\mathcal{I}^{\text{P}}$, which corresponds a set of images for pegs and holes, respectively. The middle block illustrates the training procedure for the part mating model. It is trained in a self-supervised manner using a contrastive loss that encourages the model to produce high scores only when images corresponding to true mating parts are provided. The right block demonstrates the model's generalization to different shapes after training. [Best viewed in color]

5.4 Experiments

In this section, we evaluate the performance of the *Tactile-Filter* algorithm in two different test scenarios. The first scenario, referred to as the **small objects** (Fig. 5.4 left), involves a collection of small objects that can be fully captured by a single touch of the tactile sensor, thereby making the estimation problem relatively simpler to solve. The second scenario, referred to as the **large objects** (Fig. 5.4 right), involves objects that are larger than the size of the tactile sensor, requiring multiple touch measurements to accurately estimate their shape. All the test objects used in the pose estimation experiments are novel and are not used for training the contrastive learning model.

5.4.1 Training the part mating model

Tactile sensor. We use a commercially available GelSight Mini [79] tactile sensor, which provides 320×240 compressed RGB images through the Robot Operating System (ROS) at a rate of approximately 25 Hz, with a field of view of 18.6×14.3 millimeters.

Robot platform. The MELFA ASSISTA robot [36], a collaborative robot with 6 DoF, is used in this study. The tactile sensor is mounted on the robot's wrist during data collection (see Fig. 5.3). It is noted that we do not use the force torque sensor mounted at the wrist of the robot as shown in the Fig. 5.3.

Data collection. In order to train a model that is capable of generalizing to a diverse set of shapes, we designed a board for data collection so that it features random polygonal shapes to simulate pegs and holes of arbitrary shapes. The shapes were generated through a process that involved creating a maze (we name it *MAZE* board), adding random perturbations to the position and size of the walls that make up the maze, and then exporting the result for 3D printing. This board was designed such that any arbitrary hole patch sampled from the upper half has a corresponding mating peg patch in the lower

half (see Fig. 5.3). To collect data for training, we sampled several different locations and orientations on the upper half MAZE board from a high-resolution grid to collect the hole images, and then collect the corresponding peg images from the lower half. This resulted in a total of approximately $23,000$ pairs of images of pegs and holes which perfectly fit with each other.

Preprocessing. In this study, the tactile sensor used has RGB LEDs with different colors on each of the three surfaces [79]. As a result, even when the same object is in contact, the color may differ depending on the position of the image captured. To mitigate the potential impact on generalization performance, we obtained an image of a non-contact situation $I_{\text{NoContact}}$ during data collection, reducing the impact by subtracting the image. Then, the average and variance of each RGB channel were calculated for all images, and the images were normalized before being input into the model.

Training. As described in Sec. 5.3.1, we use MoCo-v3 for our part mating model f_{HP} and peg distance model f_{PP}. We train the models with the collected images using the MAZE board for 500 epochs. To improve generalization capability, we augment the data by using random cropping and horizontal or vertical flips, which will be applied to the pairs of images inputted to the model during training.

5.4.2 Small objects

We first evaluate the performance of the TactileFilter when applied to objects that fit in the size of the sensor.

Baselines. To understand the challenges encountered when identifying objects that might not be fully captured through a single touch, we compare our method against two methods that only use the initial image. The first baseline, referred to as *Pixel*, computes the L1 distance between the peg and hole images and returns the index of the nearest neighbor image. The second baseline, *MoCo*, utilizes the pre-trained MoCo-v3 model to calculate the distance (negative of the MoCo-v3's output) based solely on the first tactile image and without incorporating any subsequent interactions. The results of our method are denoted as *Ours (n)*, where n indicates the number of interactions. It is important to note that the value of n includes the initial contact, therefore, *Ours ($n = 1$)* represents the results obtained without any additional interactions.

Settings. For this experiment, we have designed an evaluation board consisting of 12 alphabet characters (ranging from "A" to "L"), each with a maximum width of 16 mm and height of 12 mm, so the characters fit within a single touch. Since we would like to evaluate the model in situations where the pose of the object is unknown, resulting in only partial observation of the object and requiring multiple touches for accurate estimations, we collect data with displacements in $X, Y \in \{-8, -4, 0, 4, 8\}$ millimeter and $\theta \in \{-90, -60, ..., 90\}$ degree from their center position. This results in $12 \times 5 \times 5 \times 7 = 2100$ images. Figure 5.4 shows examples of the characters we used for the experiment. The hyperparameter we used for our algorithm is the number of particles $K = 100$, the maximum number of iterations $N^{\max} = 10$, and the threshold to stop the iteration $\delta^{\text{prob}} = 0.95$.

Metrics. The performance of the results is assessed through two metrics. Firstly, we evaluate the accuracy in classifying the objects. For the baseline calculation, we calculate

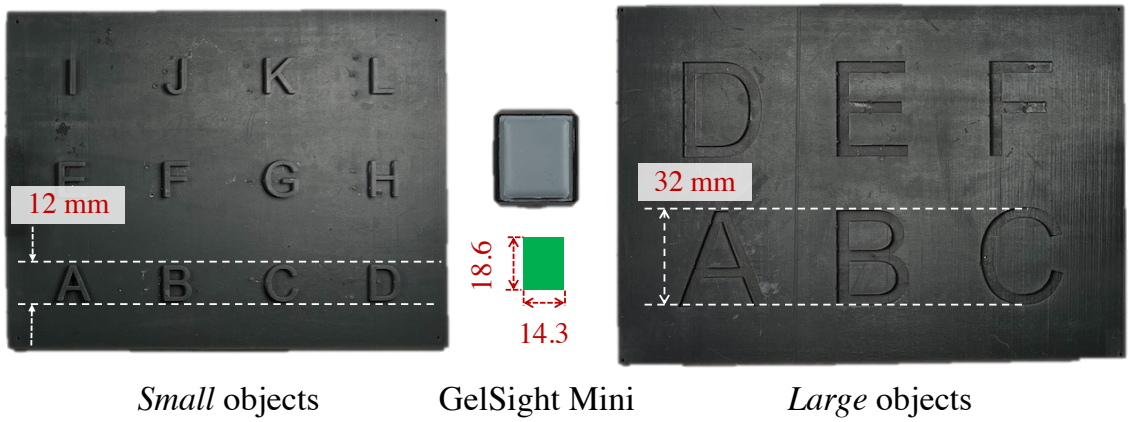


Figure 5.4: Alphabet boards for our experiments. The left board contains small characters, each with a length of 12 mm and a maximum width of 16 mm, to fit within the size of the sensor if the robot makes contact with the center position. The sensor size is shown in the middle image. The board on the right has large characters with a length of 32 mm and a maximum width of 40 mm, requiring multiple interactions with the tactile sensor to obtain complete geometry for the object.

the distance between a hole image and all previously gathered peg images, select the image with the minimum distance, and consider the prediction to be accurate when the predicted image's class matches the class of the inputted hole image. Additionally, the distance between the predicted pose and the ground truth pose is quantitatively measured.

With regard to the evaluation of the proposed method, we utilize the likelihood used to update the particles to weight the prediction. The object with the highest weighted probability is then evaluated with the target object. We also use weighted error between the particles and the ground-truth image to compute the quantitative error.

Results and Analysis. The results are presented in Table 5.1. A comparison between the two baselines, *Pixel* and *MoCo*, reveals that the correspondences between parts cannot be obtained simply by comparing pixel values. The contrastive framework captures the features of the mating parts, resulting in better performance. However, the results using only the first contact are still not sufficiently accurate as the tactile sensor only observes a partial view of the object. In contrast, our method demonstrates a gradual improvement in performance as interactions are added. Furthermore, as we can see from Table 5.1, our method is able to achieve good localization accuracy in both position and orientation. In particular, we are able to achieve a submillimeter average error in localization, which might be required for industrial insertion tasks.

5.4.3 Large objects

Settings. In the next set of experiments, we evaluate the performance of the proposed method when applied to objects larger than the sensor size. This scenario requires the robot to interact multiple times with the object to gain a comprehensive understanding of its shape. To this end, we have designed an evaluation board consisting of 12 alphabet characters (ranging from "A" to "L"), each with a maximum width of 40 mm and a height

Table 5.1: Quantitative evaluation of single touch experiments with **small** objects on the alphabet board.

	Pixel	MoCo	Ours		
			$n = 3$	$n = 5$	$n = 10$
Accuracy [%]	0.0	39.6	81.8	90.7	95.0
Error XY [mm]	-	0.7	0.2	0.1	0.1
Error θ [deg]	-	5.4	0.9	0.3	0.1

Table 5.2: Quantitative evaluation of multiple touch experiments with **large** objects on the alphabet board.

	Pixel	MoCo	Ours		
			$n = 3$	$n = 5$	$n = 10$
Accuracy [%]	11.9	41.9	58.7	72.3	85.0
Error XY [mm]	6.9	4.4	1.3	1.0	0.7
Error θ [deg]	16.6	15.5	4.4	2.9	1.5

of 32 mm. We tested the method in the location and orientation of the robot from $X, Y \in \{-20, -16, ..., 20\}$ mm and $\theta \in \{-90, -60, ..., 90\}$ with respect to the central position of each character. Figure 5.4 presents examples of the characters utilized in the experimental setup. Regarding the baselines, we compare the method with the same baselines as in the previous experiment on *small* objects.

Results and Analysis. Table 5.2 shows the results on the large objects. Similarly to the results obtained in the setting of small objects, our proposed model demonstrates improved performance compared to the baselines. However, it is also observed that a larger object size requires a greater number of interactions in order to achieve comparable accuracy. In Figure 5.5, we show the classification accuracy with respect to the number of interactions and the results with smaller sets of 4 and 8 characters to evaluate the performance with a smaller number of possible candidates. The bar plots demonstrate that the proposed method can quickly identify the correct class if the number of classes is small.

5.4.4 Ablation on action selection strategy

Settings. To assess the effectiveness of the proposed action selection strategy, we compare the proposed method with a random action selection method (which we call *Random*). We evaluate the two methods on the *Small* and *Large* objects settings described earlier.

Results and Analysis. The results in Fig. 5.6 indicate the proposed maximum likelihood action selection approach demonstrates significant improvement in comparison to the method with regard to classification accuracy.

5.4.5 Evaluation on industrial connectors

Settings. To further evaluate the performance of the trained part mating model in an industrial setting, we collect tactile images of connectors from a Raspberry-Pi board, as depicted in Fig. 5.7.

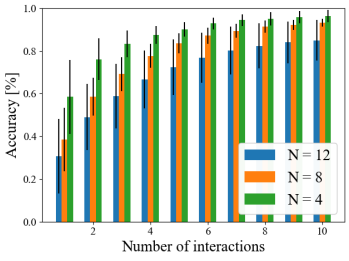
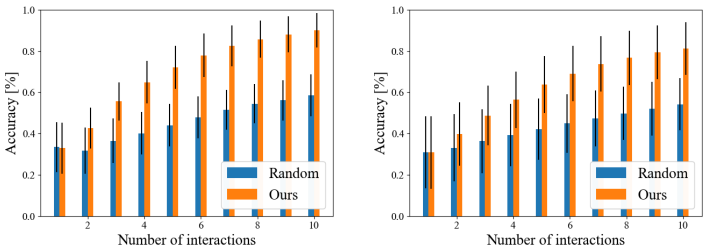


Figure 5.5: Classification accuracy of the proposed method with a different number of classes evaluated on the *Large* objects. The result shows that our method can quickly identify the correct class if the number of classes (shown by N) is limited.



(a) *Small* objects (b) *Large* objects

Figure 5.6: Classification accuracy for *Small* and *Large* objects with different action strategies. As can be seen in these bar graphs, our proposed method demonstrates a significant improvement compared to the selection of random actions with regard to classification accuracy.

Table 5.3: Classification accuracy on the Raspberry Pi Board.

	Pixel	MoCo
Accuracy [%]	50.0	83.3

Results and Analysis. The results of the evaluation of the Pixel baseline and our part mating model for the classification of connectors from the Raspberry-Pi board are presented in Table 5.3. The Pixel baseline demonstrates improved performance in comparison to the small and large object experiments, due to the reduced number of classes in this setting and the unique size of each connector, which simplifies the classification through the use of only L1 pixel distance. Although the part mating model outperforms the Pixel baseline, it misclassifies that the female HDMI connector fits the male USB-A connector. This is attributed to the significant distribution shift between the training set and the test set, where the pins on the surface of the male part are not present in the training data. To address this issue, future work can focus on enhancing the generalization capabilities of the part mating model.

5.4.6 Application to multi-object assembly

Settings. Once Tactile-Filter localizes the hole and identifies the corresponding peg, the robot can successfully insert the peg into the right hole. This is facilitated by the algorithm's ability to estimate the pose with high precision, as demonstrated by the submillimeter average prediction error (refer to Table 5.1 and Table 5.2). Consequently, we assess the proposed method in a real multi-object assembly scenario during the final experiment.

The task involves identifying and localizing four pegs and holes shaped like the alphabet characters "M", "L", "T", and "F" ("M"aximum "L"ikelihood "T"actile "F"ilter). Each peg has a length of 32 mm and a maximum width of 28 mm, as depicted in Fig. 5.1 (leftmost picture). Although the algorithm achieves accurate pose estimation for the holes, the robot

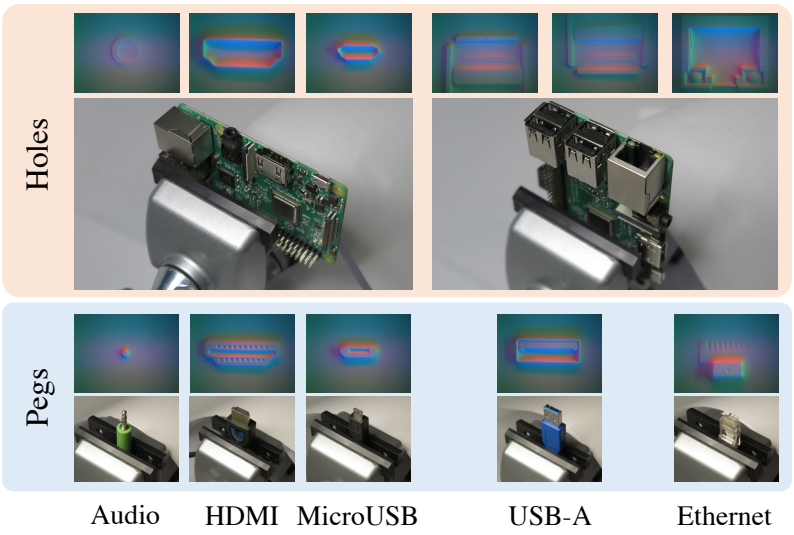


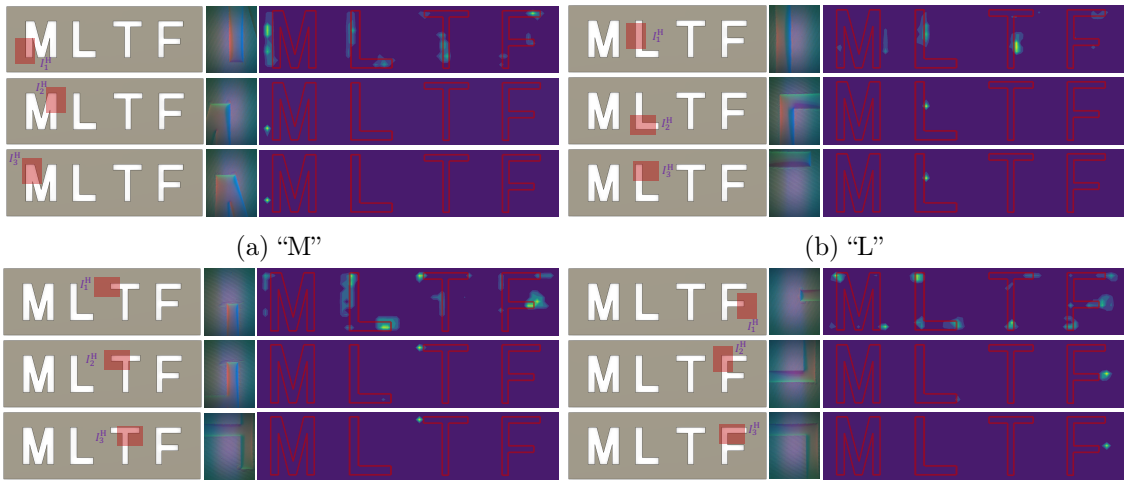
Figure 5.7: Experimental setup for industrial connector identification on a Raspberry-Pi board. This image shows the observations of the six pegs and holes using the GelSight Mini sensor. Table 5.3 shows the classification results obtained by our model. [Best viewed in color]

still fails in insertion due to errors during grasping. To account for this, we design holes with a tolerance of 2 mm and treat insertion as a simple pick-and-place operation using an impedance controller. For more precise assembly, we can combine our method with prior work, such as [41, 41].

Results and Analysis. The qualitative results are available in the video accessible at `https://www.youtube.com/watch?v=jMVBg_e3gLw`. The video demonstrates the algorithm successfully identifies the correct peg and pose of the hole, and the robot successfully inserts the pegs. Moreover, the visualization generated from this experiment as shown in Fig. 5.8 demonstrates that our method iteratively corrects its belief during the interactive perception. Finally, in terms of computational time, the most time-consuming steps of the algorithm involve updating importance weights using the part mating model f_{HP} in Eq. (5.2) and the peg distance model f_{PP} in Eq. (5.4). However, each of these steps takes approximately 0.3 seconds on a single GPU, which is significantly shorter than other robot operations. Thus, our algorithm is suitable for online control.

5.5 Discussion

Tactile sensing can allow robots to build reliable models of their environment to perform precise manipulation tasks. In this chapter, we presented a novel method called *Tactile-Filter*. We presented an interactive perception method where a robot can improve its estimate for the perception task using tactile sensors while minimizing the number of interactions required with its environment. We considered the design of the method in the context of the task of part mating. In the absence of any vision input, we described a method where the robot could incrementally improve its estimate of correspondence between parts within a fixed number of available choices. We also proposed a maximum



(a) “M” (b) “L”

(c) “T” (d) “F”

Figure 5.8: Visualization of belief maps for holes categorized as “M”, “L”, “T”, and “F”. In each figure, the red regions in the left column show the spatial and temporal region captured by the tactile sensor during interactions, represented as I_t^{H}, where t indicates the interaction number or timestep. The center image shows the hole image captured by the tactile sensor, and the right figure illustrates the belief map generated by the current particles. Across all hole types, the results indicate a rapid convergence of the distribution of the hole's initial contact pose and its corresponding category.

likelihood-based approach to select future actions to minimize the number of interactions during the perception task. The proposed method was verified using a vision-based tactile sensor and a physical robot on several tasks of part mating. The generalization of the proposed method to previously unseen scenarios was also illustrated.

As our method was trained and evaluated on a physical system, data collection was performed using a real robot. However, in future work, we aim to explore the possibility of utilizing an appropriate simulation environment [33, 185, 195, 196] to simulate contact patches for various object geometries. This approach would enable us to reduce reliance on physical robots for data collection and instead leverage simulations to acquire data. By doing so, we can learn and develop complex perception techniques with minimal usage of real-world data.

Another limitation of the proposed method is that we assume that tactile images consist of only the peg and hole parts. Therefore, the underlying deep learning model can get easily confused if distractors are present in the image (such as attachments to connectors in Fig. 5.7). We are working on this limitation by converting RGB images to pointclouds and finding mating pose in the poincloud space.

Part III

How can we solve a desired dexterous manipulation task?

Chapter 6

Solving Stable Stacking using Probabilistic Estimation and Deep RL

Humans can perform very complex and precise manipulation tasks effortlessly. Consider, for example, gently stacking two lightweight objects on top of each other without looking at them, as shown in Fig. 6.1. Successful execution of this kind of task requires the object not to fall upon release of the grasp. In these scenarios, stability is not directly observable; it must be implicitly inferred from tactile signals that entangle both *intrinsic* (direct) contact between the end effector and the grasped object and *extrinsic* (indirect) contact between the grasped object and the environment. For example, in Fig. 6.1, it is difficult to distinguish the stability of the configuration on the left from the right by looking at it visually. This work is motivated by how humans can disentangle a composite tactile signal to determine the nature of extrinsic contact; and can further predict whether a given stack configuration is stable. We present a closed-loop system that reasons about object stability using tactile signals that arise out of extrinsic contacts.

The stability of an object is governed by the relative location of the environmental contact and the center of mass location of the object, which could be estimated from the contact forces experienced by an object during placement. The forces observed by the force-torque (F/T) sensor mounted on the wrist of the robot, as well as the deformation observed by the tactile sensors co-located at the gripper fingers, depend on the contact patch between the object and its environment, as well as the geometric and physical properties of the object. Since these observed tactile signals are entangled, it will be very difficult to extract specific information, such as extrinsic contact patches. As a simplification, we assume that the geometry of the objects is fixed, so the robot works with known pieces. Under this assumption, the problem of estimating the stability of placement from tactile observations is simplified. With this understanding, we try to estimate the contact patch between the object and the environment using tactile signals. However, estimating contact patches from a single tactile observation is a partially observable problem, as totally different contact patches can generate similar tactile signals. Thus, a perfect estimate of the contact from a single interaction is impossible.

To deal with the partial observability problem, we employ the probabilistic estimation framework developed in Chapter 4 and 5 which allows the robot to update the contact

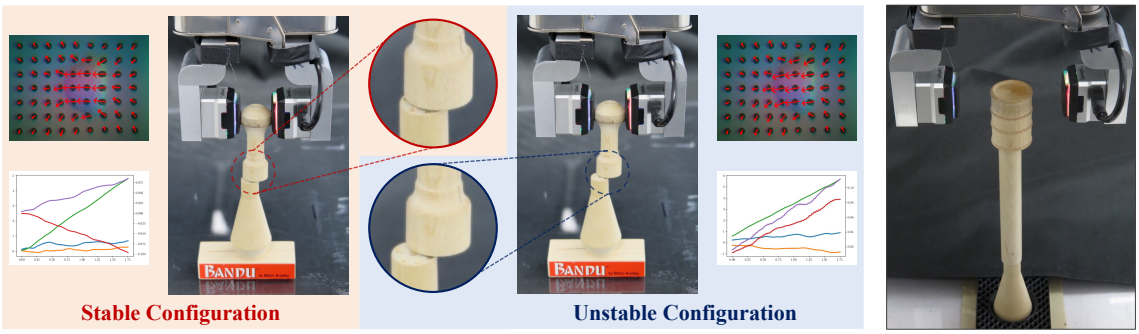


Figure 6.1: **Estimating extrinsic contact from tactile sensing**: This work studies how *extrinsic* contact (indirect contact between a manipulated object and an environment) can be estimated in the context of stable placement of an object in the environment with partial support. The figure above shows an object stacking scenario using two lightweight wooden game pieces (from the popular *Bandu* puzzle). (Left) The contact area between the two objects being stacked is critical to the success of the stack. Vision-based tactile sensors mounted on the end effector and force-torque sensor provide us with a composite signal that includes both intrinsic (direct) and (partially observable) extrinsic contacts. Our key innovation is to propose a learning-based method to estimate the extrinsic contact patch using only the composite tactile signal and knowledge of the force applied by the end effector. (Right) This enables the robot to stack a highly irregularly shaped object on top of a very unstable tower.

patch estimation from multiple tactile observations through interactions. Furthermore, the estimated contact patch from the aggregated observations is then inputted to the RL framework developed in Chapter 2 and 3 to train a policy that navigates the grasped object toward a more stable configuration so that it can be released in a stable pose. We demonstrate this behavior using several pairs of objects from a popular board game where the objective is to incorporate a new block on an existing tower without destabilizing it. We also perform ablations to understand which sensing modality, the F/T sensor or the vision-based tactile sensor is helpful in understanding the phenomena during the considered contact formation.

Contributions: In summary, the contributions of this chapter are the following.

1. We present a method for estimating extrinsic contact patches from end-effector tactile signals that compose both a force-torque sensor and vision-based tactile sensors.

2. We demonstrate the application of the probabilistic estimation approach developed in Chapter 4 and 5 successfully reconstructs the surface of the bottom object, and allows the robot to stably stack a set of extremely challenging real-world objects using solely tactile sensing with combination of the large RL framework developed in Chapter 2 and 3.

6.1 Related Work

Block stacking. Block stacking is one of the most widely studied problems in robotics. Several studies have addressed the problem of robot stacking through various approaches. These include learning to schedule auxiliary tasks for reinforcement learning (RL) [152],

combining demonstrations and RL [22, 209], employing sim-to-real transfer [69, 83, 209], and using task-and-motion planning [135]. The focus of these works primarily revolves around stacking simple cubes. Lee *et al.* [103] propose a benchmark that introduces relatively irregular rectangles generated by deforming cubes. However, these objects still maintain convexity and simplicity, allowing robots to employ relatively simple pick-and-place stratergy. Furrer *et al.* [53] and Liu *et al.* [112] have explored the stacking of irregular stones. Another related work discussing vision-based contact support could be found in [100]. Nevertheless, these studies make assumptions regarding knowledge of geometry and assume that objects possess wide support and high friction, simplifying the problem and, again, enabling basic pick-and-place strategies. Most importantly, these works do not reason about stability using contact information, but rather perform placement using open-loop controllers. These pick-and-place stackings would not work if there is ambiguity in the location of the environment and / or the supporting contact surface is limited (for example, the scenario shown in Fig. 6.1). To address this problem, our proposed method considers the local contact phenomenon in which the object can topple and fall if it is not placed with the proper support. Moreover, we remove assumptions regarding the geometry of the underlying objects, necessitating the estimation of stability through interactions.

External contact localization Prior works represent contacts as a set of points [95, 121], lines [96, 118], and patches [73]. Although line contacts give us more information compared to point contacts, they require active exploration involving changes in gripper orientation [96, 118], making it difficult to apply them in our setting, where the tower is very unstable. The closest work to ours is the neural contact fields (NCF) of Higuera et al. [73], where the authors estimate the contact patch between a grasped object and its environment. While NCF is evaluated on a simulation and a limited number of objects, we tested our method on unknown geometries of the environment, which can be used for an appropriate downstream task in a real system.

6.2 Problem Statement

We are interested in performing a stable placement in environments where the object might have partial support for placement. Consider, for example, the scenario shown in Fig. 6.1, where just establishing contact with the bottom piece is not enough for stable placement but rather the object's stability needs to be estimated in the resulting contact formation. Thus, we consider the problem of estimating the stability of an object in contact with its environment in an attempt to release and place the object in a stable pose. This is a partially observable task, as we cannot observe the full state of the system, and thus stability needs to be estimated from sensor observations. We assume that the robot has access to tactile sensors co-located at the gripper fingers and a F/T sensor at the wrist. A certain contact formation is stable if the object can remain stable after being released from the grasp.

The stability of a contact formation depends on the relative position of the center of mass and the (extrinsic) contact patch between the object and the environment. However, this cannot be directly observed during a contact formation, and thus leads to partial

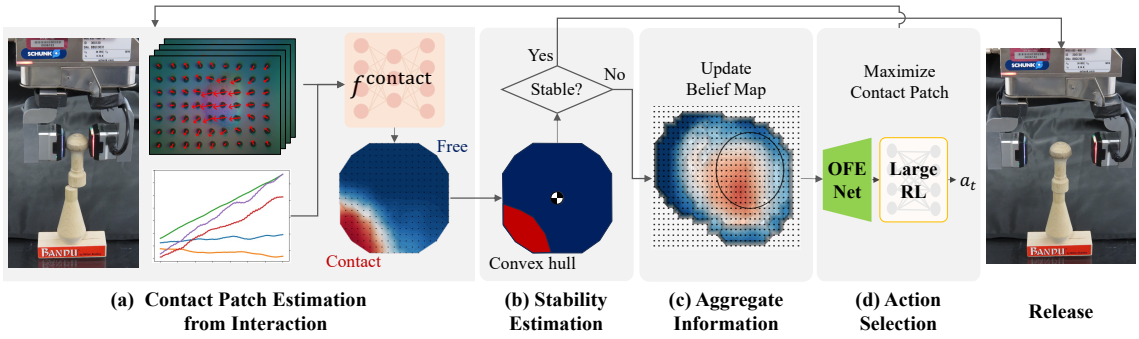


Figure 6.2: **Pipeline**: Our method comprises four components. First, a robot probes the environment to establish contact between the grasped object and the target object upon which it must be stacked. During this probing phase, we acquire a sequence of force/torque measurements and tactile images. We then estimate the extrinsic contact patch and, in turn, the potential stability of the resultant configuration. Subsequently, we aggregate the information from multiple interactions to update the belief map of the contact state. We choose the action that maximizes the contact patch between the objects.

observability. A robot can usually observe force-torque signals and/or tactile images during interaction. The observed signals depend not only on the contact formation but also on the geometry and physical parameters of the grasped object. Thus, although these data have a lot of information, these are all entangled, and thus it is very difficult to extract specific information, e.g., extrinsic contact patch. The stability estimation problem in its full scope requires reasoning about the sensor observations while considering the geometric information of the objects. To simplify the estimation problem, we make the following assumptions to limit the scope of the current study.

1. Geometry and physical parameters of the grasped objects are fixed.

2. All objects are rigid and have flat surfaces.

It is important to emphasize that the robot is unfamiliar with the shape of the underlying objects and needs to explore a stable configuration through several probing attempts. These assumptions restrict the use of our proposed objects to known grasped objects. A full and in-depth study of the problem is left as a future exercise.

6.3 Method

This work addresses the primary challenge of estimating stability during placing irregular objects. Since the contact formation between a grasped object and the environment generates sensor observations, we estimate the contact patch between the two objects from force and tactile measurements. We propose a framework consisting of four key components. First, the robot estimates the contact patch between the grasped object and its environment from an observation obtained by interacting with the environment. Then, it assesses stability based on the estimated contact patch; and releases the grasped object if it believes the current configuration is stable. Otherwise, it aggregates information from multiple estimated contact patches to predict a belief map using the probabilistic estimation

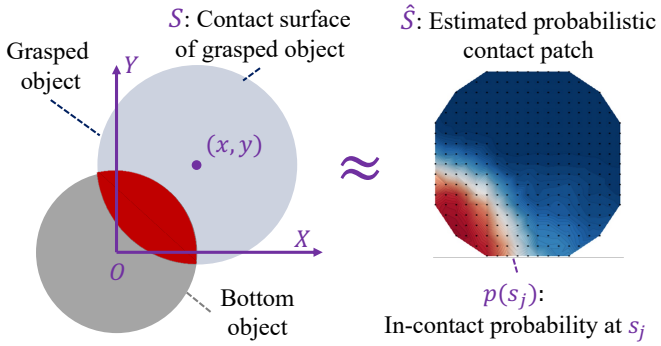


Figure 6.3: Definition of the **probabilistic contact patch**. (Left) (x, y) denotes the displacements from the origin of the bottom object O during data collection. This displacement and known contact surfaces of the two objects give the ground-truth contact patch S. (Right) The discretized contact patch $\hat{S}$ consists of a set of probabilities $p(s_j)$ that represents whether a specific position s_j of the contact surface of the grasped object is in contact or not.

framework developed in Chapter 4 and 5, which gives us a sense of the contact surface of the environment. Finally, the robot selects an action that moves the grasped object to a position that it believes can improve stability using a policy trained by the RL framework developed in Chapter 2 and 3. In this section, we describe these four modules in more detail.

6.3.1 Contact Patch Estimation

Given the observed sequence of tactile images o^{Tac} and F/T measurements o^{FT}, our objective is to learn a model that generates a probabilistic contact patch $\hat{S}$, which consists of a set of probabilities indicating which part of the grasped object is in contact.

Contact representation. We represent the contact patch by discretizing the contact surface of the grasped object S into N points as $S \approx \{s_1, ..., s_N\}$ each of which corresponds to a specific location on the contact surface of the grasped object (see Fig. 6.3 right). For each point s_j, we predict the probability of being in contact or remaining uncontacted $p(s_j)$. Consequently, we represent the probabilistic contact patch $\hat{S}$ as a set of probabilities $\hat{S} = \{p(s_1), ..., p(s_N)\}$.

Data collection by interaction. During a duration of T seconds, the robot applies a downward force along the negative Z axis for d mm, while collecting $o^{\text{Tac}}, o^{\text{FT}}$ from tactile and force-torque sensors at a frequency of 10 Hz. Specifically, $o^{\text{Tac}} = \{o_t^{\text{Tac}}\}_{t=0}^{T}$, where $o_t^{\text{Tac}} \in \mathbb{R}^{252}$ with $252 = 2 \times 2 \times 7 \times 9$, where we use two tactile sensors mounted on each finger and measure marker displacements on the XY axis in the tactile image, and 7×9 is the number of markers in column and row (see Fig. 6.2 (a)), which can be obtained by post-processing the tactile image I_t^{Tac}. Similarly, $o^{\text{FT}} = \{o_t^{\text{FT}}\}_{t=0}^{T}$, $o_t^{\text{FT}} \in \mathbb{R}^6$ is the 6-axis F/T measurement. We use a suitable impedance control to prevent the object from falling by using excessive force. In the data collection process, we sample random displacements in the XY plane from uniform distributions $x \sim U(x_{\min}, x_{\max})$ and $y \sim U(y_{\min}, y_{\max})$ whose origin is the center position of the contact surface of the lower object O (see Fig. 6.3), and the minimum and maximum ranges are defined to ensure contact between the flat surfaces

of the upper and lower objects. We use known geometries and displacements to generate ground-truth contact patches for training a model. The data collection process for each interaction takes approximately 10 seconds. This process consists of four distinct steps: (1) transitioning to the next interaction position, (2) gradually going down the robot until contact is established based on a predetermined force threshold, (3) recording data during the interaction, and (4) gradually moving the robot up to prevent the bottom object from toppling down.

Training. Finally, we train a contact patch estimation model f^{contact} that takes the observation $o^{\text{Tac}}, o^{\text{FT}}$ and learns to generate a probabilistic contact surface $\hat{S}$ as:

$$\hat{S} = f^{\text{contact}}(o^{\text{Tac}}, o^{\text{FT}}). \tag{6.1}$$

This model is trained by minimizing the binary cross-entropy loss for each data point s_j. To allow robots to understand contact phenomena, such as slip inside the fingers, we use LSTM [74] with two layers, each having 256 units, to build the model to capture patterns in time-series data.

6.3.2 Stability Estimation

We utilize the estimated contact patch $\hat{S}$ to estimate the stability of the current configuration. To do that, we first construct a convex hull $C \in \text{Convex}(\hat{S})$ (see Fig. 6.2 (b)) using points whose associated probability exceeds a predefined threshold denoted by δ, which we use $\delta = 0.9$ for our experiments. Subsequently, we check that the convex hull includes the position of the center of mass of the grasped object. In the affirmative case, the gripper releases and stacks the grasped object. Otherwise, the gripper aggregates information and moves towards a stable position by an action selection strategy described in the following sections.

6.3.3 Aggregating Information from Multiple Interactions

Since the estimation of the contact patch from tactile signals is a partially observable task, that is, multiple different contact patches can yield similar tactile signals, it is difficult to reliably estimate the contact patch from a single interaction. Therefore, we aggregate information from multiple interactions to disambiguate the estimate.

We denote the aggregated contact patch at the time step i as $\hat{S}_i^B$, again representing a probabilistic contact surface of the *bottom* object $\hat{S}_i^B = \{p(s_{1,i}^B), ..., p(s_{M,i}^B)\}$, where M is the number of discrete points. Following the formulation of the Tactile-Filter in Section 5.3, the probabilistic formulation of the contact $p(s_i^B)$ (note that we remove lowercase m for simplification) given past observation and action can be formulated as

$$p(s_i^B | o_{1:i-1}, a_{1:i-1}) = \int p(s_i^B | s_{i-1}^B, a_{i-1}) p(s_{i-1}^B | o_{1:i-1}, a_{1:i-1}), \tag{6.2}$$

where the first term is 1 as we assume deterministic dynamics, and the second term is initialized with the prior distribution and can be obtained through recursion. The posterior

can be computed as:

$$p(s_i^B|o_{1:i}, a_{1:i-1}) \propto p(o_i|s_i^B)p(s_i^B|o_{1:i-1}, a_{1:i-1}), \tag{6.3}$$

where the first term is given by the contact patch estimation model f^{contact} and the second term can be computed from Eq.(6.2). Specifically, we initialize the probability with $p(s_0^B) = \text{Bernoulli}(0.5)$ since we do not know whether the specific point is in contact or not before interaction.

6.3.4 Action Selection

To achieve a stable configuration, we employ an RL agent that learns to maximize the contact surface area in subsequent actions. As detailed in Section 6.3.1, a single interaction takes approximately 10 seconds, necessitating a sample-efficient algorithm to avoid a long training time in the real system. We address this by utilizing the framework introduced in Chapter 3, which enables RL agent training with significantly fewer samples, as empirically demonstrated in Section 3.4.5. Furthermore, to address the potential partial observability problem, we define the observation of the RL agent as the global contact surface $\hat{S}^B$ rather than the raw tactile signals o^{Tac} and $o^{\text{F/T}}$. Consequently, at timestep i, the policy takes the global contact patch $\hat{S}_i^B$ as input and learns to predict which direction the robot should move in the next step $a_i = \{\text{d}x_i, \text{d}y_i\}$. The RL agent is trained with a reward function that measures the distance from the center of mass of the grasped object and the bottom object (thereby allowing the agent to move towards a stable configuration), $r_i = -\sqrt{x_i^2 + y_i^2}$, as shown in Fig. 6.3.

6.4 Experiments

6.4.1 Settings

Tactile sensor. We use a commercially available GelSight Mini tactile sensor [79], which provides 320×240 compressed RGB images at a rate of approximately 25 Hz, with a field of view of 18.6 × 14.3 millimeters. We use gels that have 63 tracking markers.

Robot platform. The MELFA RV-5AS-D Assista robot, a collaborative robot with 6 DoF, is used in this study. The tactile sensor is mounted on the WSG-32 gripper (see Fig. 6.2). We use a Force-Torque (F/T) sensor mounted on the robot's wrist and used two-fold. First, we collect force observations used as input to the contact patch estimation model f^{contact}. Second, the stiffness control of the position-controlled robot.

Bandu. We use pieces from *Bandu* for our experiment. Bandu is a toy game that involves stacking light-weight wood pieces onto a base plate. The players take turns stacking these objects and compete to see who can stack the most. Each piece has a highly irregular shape and very limited contact surface (the average size of most pieces is several centimeters, which is much smaller than, for example, rgb-stacking benchmark [103]), requiring robots to estimate stable placements based on the shape of the objects. Figure 6.4 illustrates the Bandu pieces used in our experiments.

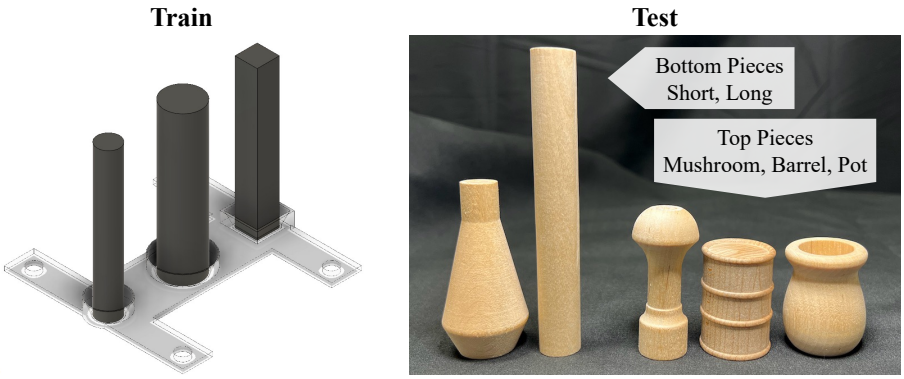


Figure 6.4: The 3D printed objects and Bandu pieces used in our experiments. (a) We use the 3D printed objects for training data collection. The objects consist of small and large cylinders with diameters of 15 and 25 mm and a rectangular prism whose length of its top surface is 15 mm. (b) The first two pieces on the left serve as the bottom objects (or the environment), while the subsequent three on the right are designated as the grasped (top) objects. These pieces have been assigned the following names: *Short*, *Long*, *Mushroom*, *Barrel*, and *Pot* from left to right.

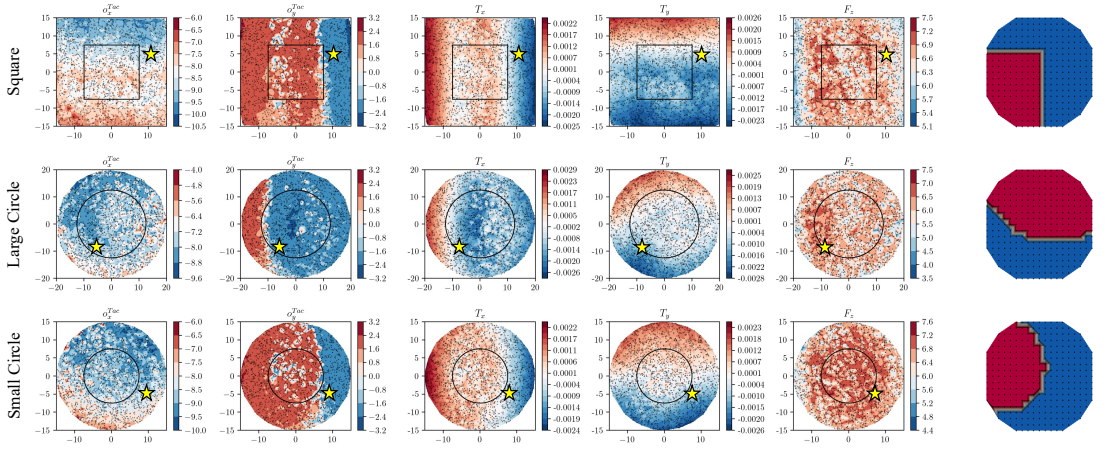
6.4.2 Data Collection

Settings. We first show the distribution of the observed tactile signals to understand the difficulties of the task. We collect 2000 tactile signals for each pair of top-bottom objects to train the contact patch estimation model f^{contact} by interacting with three objects in the 3D printed objects as shown in Fig. 6.4 (a), resulting in 6000 training samples. During data collection, we add random displacements on the XY axes as defined in Fig. 6.3, and let the robot go down for $d = 1.5$ mm after establishing contact with the bottom object for $T = 2$ seconds using the stiffness controller whose gain parameter is $(K_x, K_y, K_z) = (30, 30, 5)$ [N/mm]. We use the grasping force of 10 N.

Results and Analysis. Figure 6.5 shows the data distribution (left) and example contact patches (right). From the first to the fourth columns, we can observe the inherent difficulties of the estimation task. In many cases, we do not observe any symmetric distribution of o_x^{Tac}, o_y^{Tac} and the moment measurements T_x, T_y about $X = 0$ or $Y = 0$ while the shapes of bottom objects are symmetric (circle or rectangle). This could possibly be attributed to the inaccuracy in the manufacturing of the wood Bandu pieces or the slip of the object in the grasp during the contact interaction. Fig. 6.5 (b) shows three contact patches sampled from the star positions in each row. While tactile signals near the star positions are very similar, the resulting contact patches are quite different. This highlights the partial observability of the underlying contact formation, indicating that a single tactile observation may not be sufficient to localize the contact formation. This ambiguity makes training of a machine learning model very difficult because similar inputs (i.e., tactile observations) can lead to totally different outputs (i.e., contact patches).

6.4.3 Contact Patch Estimation

Settings. Next, we compare the performance of the contact patch estimation on different input modalities. We train the model f^{contact} for each top object using the dataset collected



(a) Training data distribution collected on 3D printed board (b) Contact patch

Figure 6.5: **Distribution of contact patches**: (a) Training data distribution with *Pot* as the grasped object and three different 3D printed shapes as the bottom objects (see Fig. 6.4). Each row shows the data obtained from different primitive shapes and each column shows the distribution of different data types: tactile displacements on the XY axes (only shows the maximum absolute values from all 63 tracking markers), moments on the XY axes and force F_z. The horizontal and vertical axes show the displacements randomly added during data collection (see Fig. 6.3), and the black circle or rectangle in each graph shows the contour of the bottom object. (b) Example contact patch sampled from the star points (★) in the left distributions. Although these contact patches are very different, the tactile signals look quite similar as seen in the data around the star point, showing the difficulty of the task; i.e., similar tactile signals can lead to very different contact patches.

in Section 6.4.2, and we evaluate the model using the intersection-over-union (IoU) and binary classification accuracy metrics. We compare the performance with three different input modalities, F/T sensor, tactile sensors, and the combination of the two denoted as *FT*, *Tac*, and *FT+Tac*, respectively. The evaluation is carried out using *unseen* two Bandu pieces (see Fig. 6.4), which we denote as *Short* and *Long*. We used a 3D-printed jig to ensure that the robot always grasps the same position of the top object and collected 400 interactions with random displacements for the two Bandu pieces.

Results and Analysis. The results are presented in Table 6.1. Comparing the three modalities, we can clearly see that the combination of the F/T sensor and tactile sensors (*FT+Tac*) yields the best performance. Consequently, for our subsequent experiments, we will utilize both of these modalities. However, it should be noted that the model is not confident enough to estimate the contact patch. This is because the same tactile signals can lead to different contact patches, as discussed in Sec. 6.4.2. Therefore, in the next experiment, we will aggregate multiple contact patch estimations from interactions and compare performance in stability estimation.

6.4.4 Stability Estimation

Next, we assess the stability estimation performance of the proposed method.

Settings. We reuse the same data as used in the previous experiments with an additional binary label indicating whether the current configuration is stable by checking whether

Table 6.1: Comparison of the contact patch estimation performance on different input modalities measured by IoU and binary classification accuracy. **Bold** numbers show the best results among the three different input modalities. The S and L of the bottom objects correspond to the *Short* and *Long* objects, respectively (see Fig. 6.4).

		Mushroom		Barrel		Pot	
		S	L	S	L	S	L
IoU	FT	27.4	37.7	29.6	**44.5**	23.8	53.2
	Tac	33.6	22.2	31.5	42.6	16.5	37.5
	FT+Tac	**38.4**	**50.7**	**31.9**	41.2	**24.8**	**54.8**
Acc	FT	67.4	65.5	75.3	73.1	**68.0**	71.9
	Tac	72.9	60.4	76.5	**77.5**	66.9	71.7
	FT+Tac	**77.9**	**73.8**	**77.0**	75.4	67.3	**74.9**

Table 6.2: Stability estimation performance measured by binary accuracy. n indicates the number of interactions and bold numbers show the best results.

		Mushroom		Barrel		Pot	
		S	L	S	L	S	L
Implicit		73.7	73.0	63.9	69.8	64.2	66.0
Ours	$n = 1$	69.8	81.0	63.7	71.4	61.7	69.4
	$n = 2$	88.4	93.0	92.7	88.9	83.2	83.9
	$n = 3$	**93.0**	**93.2**	**97.3**	**92.1**	**91.1**	**85.7**

the geometric center of the bottom surface of the grasped object (i.e., the projection of the center of mass of the grasped object on the bottom surface) lies inside the contact patch. We compare our method with a baseline model that directly produces the stability probability by replacing the final layer of f^{contact} with a fully connected layer with a single unit and sigmoid activation. We name it *Implicit* because it implicitly estimates stability from an *intrinsic* contact, while our framework explicitly predicts the stability through estimating the *extrinsic* contact patches.

Results and Analysis. Table 6.2 shows the qualitative results. Single interaction leads to poor performance, as seen in the results of the baseline (*Implicit*) as well as our method with single interaction (*Ours* $n = 1$). However, by aggregating the estimates of multiple interactions, the stability estimation performance improves significantly, leading to an average accuracy of around 90%. Figure 6.6 shows how the probability of a contact patch changes during interactions. It shows that the method corrects the initial inaccurate estimate and improves stability prediction accuracy with additional interactions, and the method finally reconstructs the contact surface of the bottom object with reasonable accuracy.

6.4.5 Stable Stacking

Finally, we evaluate the stable stacking performance of the method.

Settings. For training the RL agent, we use the same architecture as the one used in Section 3.4.2, that is, the framework consisting of the OFENet and the SAC algorithm [60]

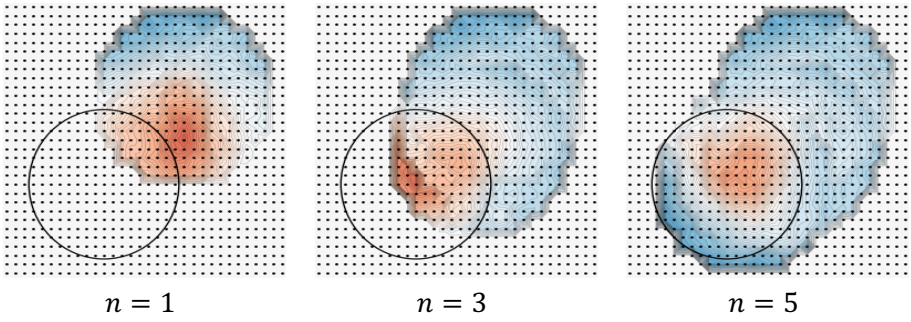


Figure 6.6: An example of how the proposed method aggregates multiple estimations and updates contact probability map. The circle in a solid line shows the ground-truth contour of the bottom object. Although the initial estimate ($n = 1$) is incorrect, the estimation accuracy monotonically improves with multiple interactions ($n = 3, 5$).

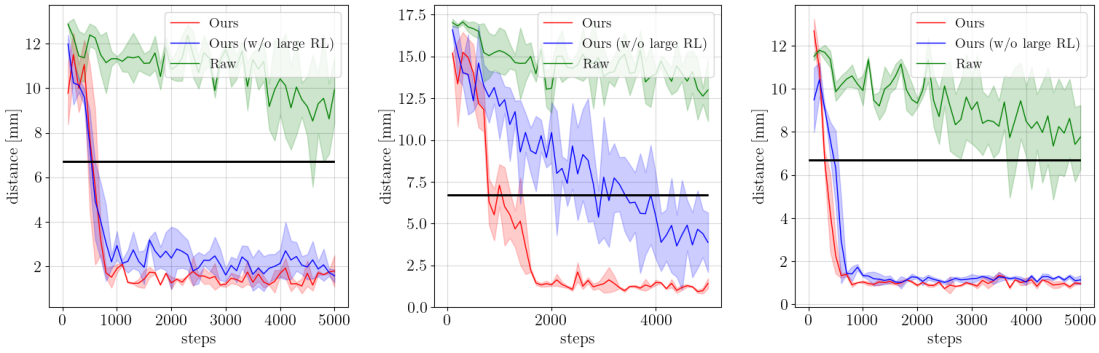
with DenseNet architecture. Among many design choices, we specifically use the largest networks used in Section 3.4.2, which were shown to achieve the best control performance and sample efficiency. To demonstrate the effectiveness of the large network architecture, we compare our method with the network consisting of the original SAC implementation, i.e., without OFENet, DenseNet architecture, and large networks, and denote it as *Ours (w/o large RL)* [1]. In addition, to demonstrate the effectiveness of our representation of the probabilistic extrinsic contact patch (estimation of the contact surface of the bottom object denoted as $\hat{S}^B$), we compare our method with another SAC agent that takes raw tactile signals o^{Tac} and o^{FT}, which we denote as *Raw* . Similar to f^{contact}, we replace the first fully-connected layer with LSTM [74] to capture patterns in time-series data.

We train all RL agents for 5000 steps for each grasped object (see Fig. 6.4), with a single training takes approximately 14 hours in the real system using the 3D-printed objects as the bottom ones. We define an episode length as 10 steps. To evaluate the methods, we always initialize the first interaction from an unstable contact state (i.e., the object would topple upon release of grasp). We run the method 20 times for each piece and evaluate the distance d to the center of the bottom object, that is, $d = \sqrt{x^2 + y^2}$ as shown in Fig. 6.3.

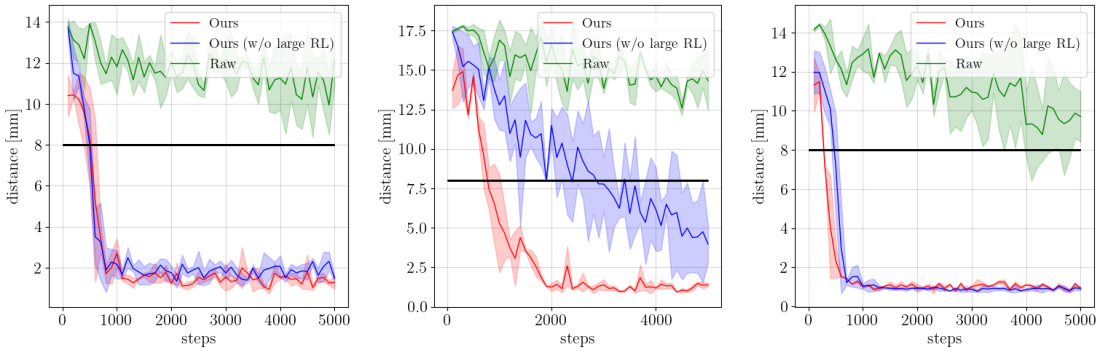
Results and Analysis. Figure 6.7 shows the distance to the center of the contact surface of the bottom object, with the horizontal solid line on each plot showing the half-size of the contact surface of the bottom object, indicating the distance below this line meaning the robot can stably stack the grasped object upon release. When comparing *Ours (w/o large RL)* with *Raw* , the better performance of using the estimated probabilistic contact patch shows the importance of extracting effective representations for the task by aggregating information from multiple interactions compared to directly learning from raw tactile signals. Again, this is because the tactile signals from a single interaction do not have sufficient information to reason about the indirect (extrinsic) contact interaction. This is also indicated in the poor binary classification accuracies in Section 6.4.3 and in the data analysis in Section 6.4.2.

Next, comparing *Ours* and *Ours (w/o large RL)* , we can clearly see that *Ours* converges

[1] We still name the method "Ours" for this method because this baseline (*Ours (w/o large RL)*) takes the global contact patch estimation as its input, which is a contribution of this work, while it does not use the large RL developed in Chapter 3.



(a) *Mushroom*, *Short* (b) *Barrel*, *Short* (c) *Pot*, *Short*



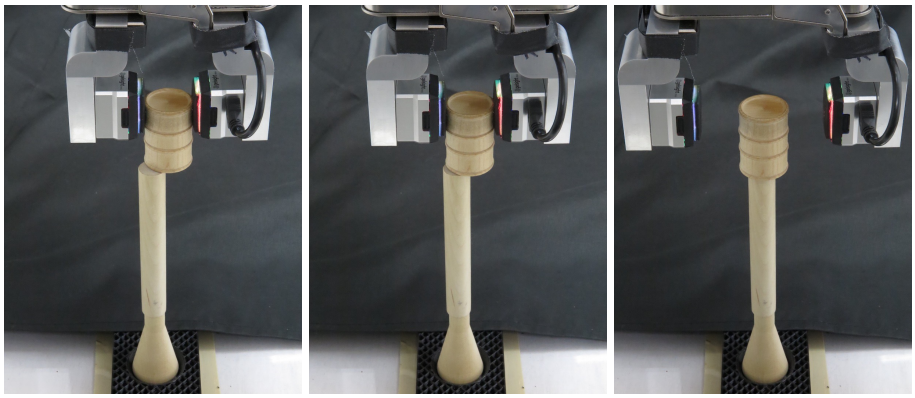
(d) *Mushroom*, *Long* (e) *Barrel*, *Long* (f) *Pot*, *Long*

Figure 6.7: Training curves on the combination of three different grasped objects and two different *unknown* bottom objects with different methods. The black horizontal lines show the half sizes [mm] of the surface of the grasped objects, indicating that the distance below the black line allows the robots to stably stack the grasped object. The proposed method denoted as *Ours* converges faster than the other baselines. The average and ±1 standard deviation results are shown as solid lines and shaded regions in the figures.

much faster with around 1000 interactions (approximately three hours in the real system). This result shows the effectiveness of the large RL framework proposed in Chapter 3 even for a real-world dexterous manipulation task. The framework allows the RL agent to converge much faster and to a better local minima, which reaches closer to the center position of the surface of bottom objects, thus more stable configurations. Figure 6.8 shows a qualitative result of how it moves to the more stable position.

6.5 Discussion

The key to precise and fine manipulation is to design systems that can interpret and disentangle useful contact information from observed tactile measurements. We proposed a framework for estimating extrinsic contact patches from tactile and force-torque measurements. Contact patch estimation allows us to explicitly estimate the stability of the placement of several different objects in novel and unstable environments. We tested the proposed approach for the placement of several pieces of the game of Bandu, which is known to be a difficult stacking task. Importantly, this system is built on top of the probabilistic estimation framework proposed in Chapter 4 and 5 and the RL framework developed in



$n = 1$ $n = 3$ $n = 5$

Figure 6.8: The robot moves towards a stable configuration and successfully stacks the *Barrel* piece on top of an already built tower consisting of *Short* and *Long*.

Chapter 2 and 3. This work proved that adopting an interpretable and essential abstraction, the extrinsic contact patch, as an intermediate representation, allows an RL agent to efficiently learn a useful policy while dealing with the partial observability by aggregating information.

In the future, we would like to improve the performance by training on a wider variety of objects and relaxing the assumption of the known geometry so that the trained model can be used for the stacking task with arbitrary objects.

Chapter 7

Conclusion

This thesis began with the question: *How can we enable robots to accomplish complex manipulation tasks that involve physical contact interactions?*, and has explored the challenges surrounding this fundamental question.

In summary, we developed an RL framework that achieves 2 times higher control performance and 100 times better sample efficiency in Part I. We then developed a probabilistic estimation framework that allows a robot to interactively estimate an essential object's representation for solving a given task in Part II. In Part III, we combine the two frameworks and demonstrate that it enables the robot to achieve a stable stacking task that involves complex physical interactions.

In this chapter, we elaborate the key findings and conclude the thesis by addressing limitations and suggesting possible future directions to further investigate and address the question.

7.1 Key Findings

At the beginning of the thesis, we considered two major challenges (high-control-performance and sample-efficient RL and effective objects' representations) in Section 1.2. We then formulated questions to address these challenges. In this section, we summarize our answers to these questions.

- **How can we train high-control-performance and sample-efficient RL policies?→ Make networks larger!**

  To address the challenge of sample efficiency and control performance of RL, we proposed a framework consisting of three key components: 1) decoupling representation learning and RL, 2) DenseNet architecture with large networks, and 3) distributed training to mitigate overfitting and distribution mismatch. While it has been shown that naively increasing the network capacity fails RL training, our proposed framework achieves much higher performance and sample efficiency than baselines.

- **How can we estimate an effective object's representation for performing manipulation?→ Probabilistic estimation using interaction!**

Probabilistic estimation using interaction serves as a more efficient, interpretable, and widely applicable way to reason about the essential representation of environments through interactions. We prove this in two use cases: articulated object manipulation and general connector mating task, both showing state-of-the-art performance in understanding objects' structures.

- **How can we solve a desired dexterous manipulation task?→ Integrate probabilistic estimation and RL!**

  To achieve dexterous manipulation, we incorporated the probabilistic estimation and the RL framework developed in Part I and II, into one framework. We apply it to a stable stacking task, where a robot has to stack a highly irregular object on top of an unstable tower. This integration enables the robot to estimate the extrinsic contact patch that cannot be directly observed, and solve the task by training an RL policy, which takes the estimated contact patch instead of raw signals, within $1,000$ interactions, corresponding to three hours of training in the real system.

7.2 Limitation and Future Directions

Although the work in this thesis has made progress towards answering the questions posed in Section 1.2, we are still far away from a robotic system that can manipulate complex objects. Getting there will require advances in several areas.

7.2.1 Generalization over diverse objects

While the proposed methods showcased the effectiveness in solving a general connector part-mating in Chapter 5 and stable stacking in Chapter 6, the challenge lies in generalization to unseen objects. This section discusses how to improve generalization capability to unseen objects.

In the case of part-mating, as shown in the experiments with industrial connectors in Section 5.4, the learned part mating model does not generalize to, for example, a male HDMI connector that has 19 tiny pins on the contact surface, which are not included in training set, and thus the trained model failed to find the mating parts. This distribution mismatch problem is not avoidable if we use any learning-based method. Therefore, we are currently replacing this vision-based model with a 3D pointcloud-based method, and removing any learning-based method in the pipeline to improve performance on unknown objects that may have complex geometries.

For the stable stacking, we assumed that the grasped objects were fixed, which limits its applicability to unknown objects. Again, since this model includes a learning-based method for estimating extrinsic contact patches, the model does not generalize due to the distribution mismatch. As described in Section 6.2, the stability of a tower is governed by the geometry, physical parameters, and contact patches between the two objects. Covering all possible sets of these parameters in the real system would be exceedingly difficult, making it challenging to train such an *object-agnostic* contact patch estimation model. While this is an open problem, one potential solution could involve using a simulator to

collect extensive data with various geometries and physical parameters, and applying it to the real system. Although evaluated only in simulation, Higuera *et al.* [73] train a model that takes a sequence of tactile images, robot joint angles, and, importantly, the point cloud of the object to improve generalizability toward diverse objects. This is a promising approach; however, simulating tactile signals remains an open problem, as discussed in the next section.

7.2.2 Sim2Real: simulation to real-world transfer

There are few, if any, reliable simulators capable of accurately replicating tactile observations. Although significant efforts have been made toward simulating tactile signals, force/torque sensing from simulators remains unreliable. Moreover, simulating tactile signals for vision-based tactile sensors (e.g., GelSight [199], used in our experiments) is even more challenging as simulating RGB images involves 1) estimating the force distribution on a gel surface given contacts, 2) simulating the deformation of the gel by the force, and 3) conducting optical simulations to generate RGB pixel values produced by LEDs inside the sensor. This requires us to know accurate physical parameters of objects, which is typically very difficult to obtain.

Some problems are too difficult to simulate accurately without substantial engineering expertise and effort. The engineering of simulation setups for complex manipulation skills is particularly challenging. For example, in the stable stacking task discussed in Chapter 6, reliable results are difficult to achieve without detailed information on the physical properties of the objects involved, such as the mass and inertia of both the grasped and bottom objects, the friction parameters of the fingers (gels of GelSight) and objects, and the geometries of the grasped object and its environment. Since physical interactions are simulated on the basis of these parameters, understanding these values is essential to obtain reliable results.

Nevertheless, obtaining a reliable simulator would pave the way for manipulation with more complex interactions that might require a vast number of interactions, which could be challenging to train solely in the real system. We can begin with a quasistatic setting, where the simulator does not need to simulate the dynamics of objects, making the simulation significantly easier. Additionally, using depth information instead of RGB pixel values can lead to more reliable estimates. Under these assumptions, simulators can be beneficial for solving, for example, part-mating tasks discussed in Chapter 5, where we can collect peg images in simulation instead of physically interacting with pegs (see the leftmost data collection in Fig. 5.1). In any case, developing high-fidelity simulators, particularly for simulating tactile signals, remains an open problem.

7.2.3 Integration of vision and tactile

In the real experiments described in Chapter 5 and 6, we focused on using force/torque (F/T) and/or tactile sensors to estimate the object's pose and extrinsic contact patches during manipulation. However, tactile sensors can only capture local contact information. Especially for the part-mating task in Chapter 5, this limitation (small sensor area) necessitates multiple interactions when the contact region does not contain distinguishable

geometric features.

In contrast, vision can capture the global information of the object, though it suffers from occlusion and lower accuracy. Recent advances in vision foundation models [97, 138] and vision-language models (VLMs) [107] provide functionalities such as depth estimation, instance segmentation, and dense matching. This information would be very useful when designing a system that solves tasks such as stable stacking comprehensively, where the robot must first estimate how to grasp an object for stacking and determine where to make initial contact with an already built tower.

Acknowledgment

I would like to take this opportunity to express my gratitude to my advisor, Prof. *Asako Kanezaki*, for her continuous support and guidance. I first met her approximately six years ago at the domestic conference (MIRU), where I was seeking a collaborator with expertise in robotics and computer vision (CV). Immediately after hearing her presentation, I approached her to collaborate with my company. Since then, I have learned a lot from her, not only related to research but also through her invaluable advice on my career, family matters, and more.

I also extend my gratitude to the distinguished committee members, Prof. *Koichi Shinoda*, Prof. *Takamitsu Matsubara*, Prof. *Rio Yokota*, Prof. *Masamichi Shimosaka*, and Prof. *Naoaki Okazaki*, for their time and effort in reviewing my work and providing constructive feedback. Their invaluable insights have significantly broadened my perspective, allowing me to rethink the structure of my thesis and shape my future research direction. Prof. *Shinoda* offered me the opportunity to participate in a paper reading meeting in his lab when I was a master's student in 2015. At that time, I was working entirely alone on satellite image classification and had no one to rely on, so his acceptance and advice were immensely helpful.

I also thank my lab members, especially *Haruyuki Nakagawa* and *Hao Zheng* for being the first Ph.D. students together in our lab, and *Jiawei Jiang*, *Keigo Kamiyama*, and *Ryosuke Takanami* for working on robotics research together. I wish them a fruitful and rewarding remainder of their student life and future careers. Also, I would like to thank Secretary *Akiko Fukuda* for receiving such great support.

I am also grateful to my colleagues at MERL, especially *Devesh Jha*, *Diego Romeres*, *Pedro Miraldo*, *Radu Corcodel*, *Siddarth Jain*, *Yuki Shirai*, and *Anthnoy Vetro*. *Devesh* has been my mentor, closest collaborator, and dear friend since my first visit to MERL in May 2018. Without meeting him, I would not have started my research on robot manipulation. *Diego* and *Pedro* always provided excellent advice and welcomed me as a family. Thanks to them, my stay in lonely Boston became very pleasant. *Anthony* introduced me to the taste of real beer, which will become an important part of my life.

I am extremely grateful to my family – *Shin Ota*, *Momoyo Ota*, and *Suguru Ota* – and extended family for their endless support. I especially want to acknowledge my parents and brother for allowing me to pursue my interests since childhood and for working hard to provide me with every opportunity to succeed. Their support and encouragement have truly shaped who I am today. I hope they stay healthy and happy for a very very long time.

Last but not least, I am deeply grateful to my partner *Eri Ota*, for all her support.

She has always prioritized my career, shared and amplified my joys during good times, and helped me contemplate my future direction. Her visit to Boston brought color to my research-focused life and created many wonderful memories together. I look forward to the future we will build together.

References

[1] Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

[2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

[3] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. {OPAL}: Offline primitive discovery for accelerating offline reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021.

[4] Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020.

[5] Samuel Alvernaz and Julian Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2017.

[6] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021.

[7] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

[8] Alice M I Auersperg, Alex Kacelnik, and Auguste M. P. von Bayern. Explorative learning and functional inferences on a five-step means-means-end problem in goffin's cockatoos (cacatua goffini). *PLoS ONE*, 8, 2013.

[9] Dibya Ghosh Aviral Kumar, Rishabh Agarwal and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021.

[10] Jeroen van Baar, Alan Sullivan, Radu Cordorel, Devesh Jha, Diego Romeres, and Daniel Nikovski. Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6001–6007, 2019.

[11] Patrick R. Barragän, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Interactive bayesian identification of kinematic mechanisms. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2013–2020, 2014.

[12] Manuel Baum, Matthew Bernstein, Roberto Martin-Martin, Sebastian Höfer, Johannes Kulick, Marc Toussaint, Alex Kacelnik, and Oliver Brock. Opening a lockbox through physical exploration. In *Proceedings of IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, pages 461–467, 11 2017.

[13] Manuel Baum, Matthew Bernstein, Roberto Martin-Martin, Sebastian Höfer, Johannes Kulick, Marc Toussaint, Alex Kacelnik, and Oliver Brock. Opening a lockbox through physical exploration. In *Proceedings of IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, pages 461–467, 2017.

[14] Maria Bauza, Antonia Bronars, Yifan Hou, Ian Taylor, Nikhil Chavan-Dafle, and Alberto Rodriguez. simple: a visuotactile method learned in simulation to precisely pick, localize, regrasp, and place objects. *arXiv preprint arXiv:2307.13133*, 2023.

[15] Maria Bauza, Antonia Bronars, and Alberto Rodriguez. Tac2pose: Tactile object pose estimation from the first touch. *The International Journal of Robotics Research (IJRR)*, 42(13):1185–1209, 2023.

[16] Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S. Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

[17] Eli S. Bridge, Kasper Thorup, Melissa S. Bowlin, Phillip B. Chilson, Robert H. Diehl, René W. Fléron, Phillip Hartl, Roland Kays, Jeffrey F. Kelly, W. Douglas Robinson, and Martin Wikelski. Technology on the Move: Recent and Forthcoming Innovations for Tracking Migratory Birds. *BioScience*, 61(9):689–698, 09 2011.

[18] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan,

Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*, 2023.

[19] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S Ryoo, Grecia Salazar, Pannag R Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan H Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale. In *Robotics Science and System (RSS)*, Daegu, Republic of Korea, July 2023.

[20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[21] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, volume 31. Curran Associates, Inc., 2018.

[22] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, Oleg Sushkov, David Barker, Jonathan Scholz, Misha Denil, Nando de Freitas, and Ziyu Wang. Scaling data-driven robotics with reward sketching and batch reinforcement learning, 2020.

[23] Gabriele M. Caddeo, Nicola A. Piga, Fabrizio Bottarel, and Lorenzo Natale. Collision-aware in-hand 6d object pose estimation using multiple vision-based tactile sensors. In

Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2023.

[24] Roberto Calandra, Andrew Owens, Dinesh Jayaraman, Justin Lin, Wenzhen Yuan, Jitendra Malik, Edward H Adelson, and Sergey Levine. More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robotics and Automation Letters (RAL)*, 3(4):3300–3307, 2018.

[25] Arkadeep Narayan Chaudhury, Timothy Man, Wenzhen Yuan, and Christopher G. Atkeson. Using collocated vision and tactile sensors for visual servoing and localization. *IEEE Robotics and Automation Letters (RAL)*, 7(2):3427–3434, 2022.

[26] Nikhil Chavan-Dafle, Rachel Holladay, and Alberto Rodriguez. Planar in-hand manipulation via motion cones. *The International Journal of Robotics Research (IJRR)*, 39(2-3):163–182, 2020.

[27] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020.

[28] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[29] X. Chen, S. Xie, and K. He. An empirical study of training self-supervised vision transformers. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 9620–9629, Los Alamitos, CA, USA, oct 2021. IEEE Computer Society.

[30] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 2172–2180, 2016.

[31] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, page 7–10, New York, NY, USA, 2016. Association for Computing Machinery.

[32] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[33] Alex Church, John Lloyd, Raia Hadsell, and Nathan F. Lepora. Tactile sim-to-real policy transfer via real-to-sim image translation. In *Proceedings of the 5th Conference*

on Robot Learning (CoRL), volume 164 of *Proceedings of Machine Learning Research*. PMLR, 08–11 Nov 2022.

[34] Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Mart'in-Mart'in, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shuran Song, Sichun Xu, Siddhant

Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, Zipeng Fu, and Zipeng Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. `https://arxiv.org/abs/2310.08864`, 2023.

[35] MMSelfSup Contributors. MMSelfSup: Openmmlab self-supervised learning toolbox and benchmark. `https://github.com/open-mmlab/mmselfsup`, 2021.

[36] Mitsubishi Electric Corporation. MELFA ASSISTA. `https://www.mitsubishielectric.com/fa/products/rbt/robot/items/assista/index.html`. Accessed: 2023-01-16.

[37] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

[39] Snehal Dikhale, Karankumar Patel, Daksh Dhingra, Itoshi Naramura, Akinobu Hayashi, Soshi Iba, and Nawid Jamali. Visuotactile 6d pose estimation of an in-hand object using vision and tactile sensor data. *IEEE Robotics and Automation Letters (RAL)*, 7(2):2148–2155, 2022.

[40] Rostam Dinyari, Pierre Sermanet, and Corey Lynch. Learning to play by imitating humans. *arXiv preprint arXiv:2006.06874*, 2020.

[41] Siyuan Dong, Devesh K Jha, Diego Romeres, Sangwoon Kim, Daniel Nikovski, and Alberto Rodriguez. Tactile-rl for insertion: Generalization to objects of unknown geometry. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 6437–6443. IEEE, 2021.

[42] Siyuan Dong, Daolin Ma, Elliott Donlon, and Alberto Rodriguez. Maintaining grasps within slipping bounds by monitoring incipient slip. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3818–3824. IEEE, 2019.

[43] Siyuan Dong and Alberto Rodriguez. Tactile-based insertion for dense box-packing. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7953–7960, 2019.

[44] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1329–1338, 2016.

[45] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, Feb 2021.

[46] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 3061–3071, 2020.

[47] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of International Conference on Machine Learning (ICML)*, ICML'17, page 1126–1135. JMLR.org, 2017.

[48] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *Proceedings of International Conference on Machine Learning (ICML)*, 2019.

[49] Letian Fu, Huang Huang, Lars Berscheid, Hui Li, Ken Goldberg, and Sachin Chitta. Safely learning visuo-tactile feedback policies in real for industrial insertion. *arXiv preprint arXiv:2210.01340*, 2022.

[50] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *arXiv*, 2024.

[51] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters (RAL)*, 6(3):4257–4264, 2021.

[52] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of International Conference on Machine Learning (ICML)*, 2018.

[53] Fadri Furrer, Martin Wermelinger, Hironori Yoshida, Fabio Gramazio, Matthias Kohler, Roland Siegwart, and Marco Hutter. Autonomous robotic stone stacking with online next best object target pose planning. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2350–2356, 2017.

[54] Cristina Garcia Cifuentes, Jan Issac, Manuel Wüthrich, Stefan Schaal, and Jeannette Bohg. Probabilistic articulated real-time tracking for robot manipulation. *IEEE Robotics and Automation Letters (RAL)*, 2(2):577–584, 2017.

[55] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Artificial Intelligence and Statistics (AISTATS)*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[56] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.

[57] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[58] Huifeng Guo, Ruiming TANG, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1725–1731, 2017.

[59] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[60] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

[61] Johanna Hansen, Francois Hogan, Dmitriy Rivkin, David Meger, Michael Jenkin, and Gregory Dudek. Visuotactile-rl: Learning multimodal manipulation policies with deep reinforcement learning. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 8298–8304, 2022.

[62] Karol Hausman, Scott Niekum, Sarah Osentoski, and Gaurav S. Sukhatme. Active articulation model estimation through interactive perception. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3305–3312, 2015.

[63] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016.

[66] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 4182–4192. PMLR, 13–18 Jul 2020.

[67] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning That Matters. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, pages 3207–3214, 2018.

[68] Nick Heppert, Toki Migimatsu, Brent Yi, Claire Chen, and Jeannette Bohg. Category-independent articulated object tracking with factor graphs. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3800–3807, 2022.

[69] Lukas Hermann, Max Argus, Andreas Eitel, Artemij Amiranashvili, Wolfram Burgard, and Thomas Brox. Adaptive curriculum generation from demonstrations for sim-to-real visuomotor control. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 6498–6505, 2020.

[70] Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

[71] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Agent modeling as auxiliary task for deep reinforcement learning. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[72] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[73] Carolina Higuera, Siyuan Dong, Byron Boots, and Mustafa Mukadam. Neural contact fields: Tracking extrinsic contact with tactile sensing. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 12576–12582. IEEE, 2023.

[74] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[75] Francois R. Hogan, Jose Ballester, Siyuan Dong, and Alberto Rodriguez. Tactile dexterity: Manipulation primitives with tactile feedback. In *Proceedings of IEEE*

International Conference on Robotics and Automation (ICRA), pages 8863–8869, 2020.

[76] Rachel Holladay, Tomás Lozano-Pérez, and Alberto Rodriguez. Robust planning for multi-stage forceful manipulation. *The International Journal of Robotics Research (IJRR)*, 43(3):330–353, 2024.

[77] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[78] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[79] GelSight Inc. GelSight Mini. `https://www.gelsight.com/gelsightmini/`. Accessed: 2023-01-16.

[80] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[81] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[82] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[83] Rae Jeong, Yusuf Aytar, David Khosid, Yuxiang Zhou, Jackie Kay, Thomas Lampe, Konstantinos Bousmalis, and Francesco Nori. Self-supervised sim-to-real adaptation for visual robotic manipulation. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2718–2724, 2020.

[84] Devesh K. Jha, Diego Romeres, William Yerazunis, and Daniel Nikovski. Imitation and supervised learning of compliance for robotic assembly. In *European Control Conference (ECC)*, pages 1882–1889, 2022.

[85] Zhenyu Jiang, Cheng-Chun Hsu, and Yuke Zhu. Ditto: Building digital twins of articulated objects from interaction. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5606–5616, 2022.

[86] Rico Jonschkowski and Oliver Brock. State Representation Learning in Robotics: Using Prior Knowledge about Physical Interaction. In *Robotics Science and System (RSS)*, 2014.

[87] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

[88] Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, pages 195–206, 2017.

[89] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[90] Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2019.

[91] Dov Katz, Moslem Kazemi, J. Andrew (Drew) Bagnell, and Anthony (Tony) Stentz. Go-explore: a new approach for hard-exploration problemsation, tracking, and kinematic modeling of unknown 3d articulated objects. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 5003 – 5010, May 2013.

[92] Dov Katz, Andreas Orthey, and Oliver Brock. *Interactive Perception of Articulated Objects*, pages 301–315. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[93] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, Aug 2023.

[94] Tarik Kelestemur, Robert Platt, and Taskin Padir. Tactile pose estimation and policy learning for unknown object manipulation. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, AAMAS '22, page 742–750, Richland, SC, 2022. International Foundation for Autonomous Agents and Multiagent Systems.

[95] Sangwoon Kim, Devesh K. Jha, Diego Romeres, Parag Patre, and Alberto Rodriguez. Simultaneous tactile estimation and control of extrinsic contact. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 12563–12569, 2023.

[96] Sangwoon Kim and Alberto Rodriguez. Active extrinsic contact sensing: Application to general peg-in-hole insertion. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 10241–10247. IEEE, 2022.

[97] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 4015–4026, October 2023.

[98] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-Normalizing Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 971–980, 2017.

[99] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, volume 25. Curran Associates, Inc., 2012.

[100] Oliver Kroemer, Simon Leischnig, Stefan Luettgen, and Jan Peters. A kernel-based approach to learning contact distributions for robot manipulation tasks. *Autonomous Robots*, 42:581–600, 2018.

[101] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[102] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR, 13–18 Jul 2020.

[103] Alex X. Lee, Coline Manon Devin, Yuxiang Zhou, Thomas Lampe, Konstantinos Bousmalis, Jost Tobias Springenberg, Arunkumar Byravan, Abbas Abdolmaleki, Nimrod Gileadi, David Khosid, Claudio Fantacci, Jose Enrique Chen, Akhil Raju, Rae Jeong, Michael Neunert, Antoine Laurens, Stefano Saliceti, Federico Casarini, Martin Riedmiller, raia hadsell, and Francesco Nori. Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, volume 164 of *Proceedings of Machine Learning Research*, pages 1089–1131. PMLR, 08–11 Nov 2022.

[104] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019.

[105] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

[106] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 6391–6401, 2018.

[107] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 19730–19742. PMLR, 2023.

[108] Rui Li, Robert Platt, Wenzhen Yuan, Andreas ten Pas, Nathan Roscup, Mandayam A. Srinivasan, and Edward Adelson. Localization and manipulation of small parts using gelsight tactile sensing. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3988–3993, 2014.

[109] Xiang Li, Jinghuan Shang, Srijan Das, and Michael Ryoo. Does self-supervised learning really improve reinforcement learning from pixels? *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 35:30865–30881, 2022.

[110] Xiaolong Li, He Wang, Li Yi, Leonidas J Guibas, A Lynn Abbott, and Shuran Song. Category-level articulated object pose estimation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3706–3715, 2020.

[111] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[112] Yifang Liu, Jiwon Choi, and Nils Napp. Planning for robotic dry stacking with irregular stones. In *Field and Service Robotics*, pages 321–335, Singapore, 2021. Springer Singapore.

[113] Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021.

[114] Tomás Lozano-Pérez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research (IJRR)*, 3(1):3–24, 1984.

[115] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: a view from the width. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 6232–6240, 2017.

[116] Shan Luo, Wenxuan Mou, Kaspar Althoefer, and Hongbin Liu. Localizing the object contact through matching tactile features with visual map. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3903–3908, 2015.

[117] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, volume 100 of *Proceedings of Machine Learning Research*, pages 1113–1132. PMLR, 30 Oct–01 Nov 2020.

[118] Daolin Ma, Siyuan Dong, and Alberto Rodriguez. Extrinsic contact sensing with relative-motion tracking from distributed tactile measurements. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 11262–11268. IEEE, 2021.

[119] Jeffrey Mahler, Florian T. Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964, 2016.

[120] Zhao Mandi, Yijia Weng, Dominik Bauer, and Shuran Song. Real2code: Reconstruct articulated objects via code generation. *arXiv preprint arXiv:2406.08474*, 2024.

[121] Lucas Manuelli and Russ Tedrake. Localizing external contact using proprioceptive sensors: The contact particle filter. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5062–5069, 2016.

[122] Roberto Martín Martín and Oliver Brock. Online interactive perception of articulated objects with multi-level recursive estimation based on task-specific priors. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2494–2501, 2014.

[123] Matthew T. Mason. Toward robotic manipulation. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(Volume 1, 2018):1–28, 2018.

[124] Matthew T. Mason and J. Kenneth Salisbury. *Robot hands and the mechanics of manipulation*. MIT Press, Cambridge, MA, USA, 1985.

[125] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[126] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.

[127] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[128] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[129] Kaichun Mo, Leonidas J. Guibas, Mustafa Mukadam, Abhinav Gupta, and Shubham Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 6813–6823, October 2021.

[130] Aditya Mohan, Carolin Benjamins, Konrad Wienecke, Alexander Dockhorn, and Marius Lindauer. AutoRL hyperparameter landscapes. In *AutoML Conference 2023*, 2023.

[131] J Munk, J Kober, and R Babuška. Learning state representation for deep actor-critic control. In *IEEE Conference on Decision and Control (CDC)*, pages 4667–4673, 2016.

[132] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 1054–1062, 2016.

[133] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2019.

[134] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 2603–2612, 2017.

[135] Michael D. Noseworthy, Caris Moses, Isaiah Brand, Sebastian Castro, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Nicholas Roy. Active learning of abstract plan feasibility. *Robotics Science and System (RSS)*, 2021.

[136] Tonci Novkovic, Remi Pautrat, Fadri Furrer, Michel Breyer, Roland Siegwart, and Juan Nieto. Object finding in cluttered scenes using interactive perception. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 8338–8344, 2020.

[137] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[138] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski.

DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

[139] Kei Ota, Siddarth Jain, Mengchao Zhang, and Devesh K Jha. Tactile pose feedback for closed-loop manipulation tasks. In *Robotics: Science and Systems, Learning Dexterous Manipulation Workshop*, 2023.

[140] Kei Ota, Tomoaki Oiki, Devesh Jha, Toshisada Mariyama, and Daniel Nikovski. Can increasing input dimensionality improve deep reinforcement learning? In *Proceedings of International Conference on Machine Learning (ICML)*, pages 7424–7433, 2020.

[141] Kei Ota, Hsiao-Yu Tung, Kevin A. Smith, Anoop Cherian, Tim K. Marks, Alan Sullivan, Asako Kanezaki, and Joshua B. Tenenbaum. H-saur: Hypothesize, simulate, act, update, and repeat for understanding object articulations from interactions. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 7272–7278, 2023.

[142] Stefan Otte, Johannes Kulick, Marc Toussaint, and Oliver Brock. Entropy-based strategies for physical exploration of the environment's degrees of freedom. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 615–622, 2014.

[143] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, et al. Automated reinforcement learning (autorl): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, 2022.

[144] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J. Lim. Demonstration-guided reinforcement learning with learned skills. *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[145] Sudeep Pillai, Matthew Walter, and Seth Teller. Learning articulated motions from visual demonstration. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[146] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[147] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 5099–5108, 2017.

[148] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[149] Aravind Rajeswaran, Kendall Lowrey, Emanuel V. Todorov, and Sham M Kakade. Towards generalization and simplicity in continuous control. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 6550–6561, 2017.

[150] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for Activation Functions. *CoRR*, 2017.

[151] Hassan Ramchoun, M A Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil. New modeling of multilayer perceptron architecture optimization with regularization: an application to pattern classification. *IAENG International Journal of Computer Science*, 44(3):261–269, 2017.

[152] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom van de Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4344–4353. PMLR, 10–15 Jul 2018.

[153] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

[154] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec 2020.

[155] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[156] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[157] Yu She, Shaoxiong Wang, Siyuan Dong, Neha Sunil, Alberto Rodriguez, and Edward Adelson. Cable manipulation with a tactile-reactive gripper. *The International Journal of Robotics Research (IJRR)*, 40(12-14):1385–1401, 2021.

[158] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[159] Yuki Shirai, Devesh K. Jha, Arvind U. Raghunathan, and Dennis Hong. Tactile tool manipulation. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 12597–12603, 2023.

[160] Bruno Siciliano, Oussama Khatib, and Torsten Kröger. *Springer handbook of robotics*, volume 200. Springer, 2008.

[161] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[162] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[163] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017.

[164] Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS), Deep Reinforcement Learning Workshop*, 2020.

[165] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, volume 164 of *Proceedings of Machine Learning Research*, pages 477–490. PMLR, 08–11 Nov 2022.

[166] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

[167] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 9870–9879. PMLR, 18–24 Jul 2021.

[168] Jürgen Sturm, Cyrill Stachniss, and Wolfram Burgard. A probabilistic framework for learning kinematic models of articulated objects. *Journal of Artificial Intelligence Research*, 41:477–526, 2011.

[169] Entong Su, Chengzhe Jia, Yuzhe Qin, Wenxuan Zhou, Annabella Macaluso, Binghao Huang, and Xiaolong Wang. Sim2real manipulation on unknown objects with tactile-based reinforcement learning. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[170] Sudharshan Suresh, Zilin Si, Stuart Anderson, Michael Kaess, and Mustafa Mukadam. Midastouch: Monte-carlo inference over distributions across sliding touch. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2022.

[171] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 6105–6114. PMLR, 2019.

[172] Bingjie Tang, Michael A Lin, Iretiayo A Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj S Narang. IndustReal: Transferring Contact-Rich Assembly Tasks from Simulation to Reality. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.

[173] Ian H. Taylor, Siyuan Dong, and Alberto Rodriguez. Gelslim 3.0: High-resolution measurement of shape, force and slip in a compact tactile-sensing finger. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 10781–10787, 2022.

[174] Edward L. Thorndike. Animal intelligence. *Psych Revmonog*, 8(2):207–208, 1911.

[175] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[176] Yusuke Urakami, Alec Hodgkinson, Casey Carlin, Randall Leu, Luca Rigazio, and Pieter Abbeel. Doorgym: A scalable door opening environment and baseline agent. In *Advances in Neural Information Processing Systems (NeurIPS), Deep Reinforcement Learning Workshop*, 2019.

[177] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *CoRR*, 2018.

[178] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[179] Herke Van Hoof, Nutan Chen, Maximilian Karl, Patrick Van Der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016-Novem:3928–3934, 2016.

[180] Herke van Hoof, Tucker Hermans, Gerhard Neumann, and Jan Peters. Learning robot in-hand manipulation with tactile features. In *Proceedings of IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, pages 121–127, 2015.

[181] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, volume 30. Curran Associates, Inc., 2017.

[182] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 550–558, 2016.

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

[184] Xingchen Wan, Cong Lu, Jack Parker-Holder, Philip J Ball, Vu Nguyen, Binxin Ru, and Michael Osborne. Bayesian generational population-based training. In *International Conference on Automated Machine Learning*, pages 14–1. PMLR, 2022.

[185] Shaoxiong Wang, Mike Lambeta, Po-Wei Chou, and Roberto Calandra. Tacto: A fast, flexible, and open-source simulator for high-resolution vision-based tactile sensors. *IEEE Robotics and Automation Letters (RAL)*, 7(2):3930–3937, 2022.

[186] Xiaogang Wang, Bin Zhou, Yahao Shi, Xiaowu Chen, Qinping Zhao, and Kai Xu. Shape2motion: Joint analysis of motion parts and attributes from 3d shapes. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[187] Yian Wang, Ruihai Wu, Kaichun Mo, Jiaqi Ke, Qingnan Fan, Leonidas Guibas, and Hao Dong. AdaAfford: Learning to adapt manipulation affordance for 3d articulated objects via few-shot interactions. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2022.

[188] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, NIPS'15, page 2746–2754, Cambridge, MA, USA, 2015. MIT Press.

[189] Erik Wijmans. Pointnet++ pytorch. `https://github.com/erikwijmans/Pointnet2_PyTorch`, 2018.

[190] Ruihai Wu, Yan Zhao, Kaichun Mo, Zizheng Guo, Yian Wang, Tianhao Wu, Qingnan Fan, Xuelin Chen, Leonidas Guibas, and Hao Dong. VAT-Mart: Learning visual action trajectory proposals for manipulating 3d articulated objects. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2022.

[191] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90:119–133, 2019.

[192] Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, Feb 2022.

[193] Fei Xia, William B. Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters (RAL)*, 5(2):713–720, 2020.

[194] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. Sapien: A simulated part-based interactive environment. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11094–11104, 2020.

[195] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Robotics Science and System (RSS)*, Virtual, July 2021.

[196] Jie Xu, Sangwoon Kim, Tao Chen, Alberto Rodriguez Garcia, Pulkit Agrawal, Wojciech Matusik, and Shinjiro Sueda. Efficient tactile simulation with differentiability for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2022.

[197] Zhenjia Xu, Zhanpeng He, and Shuran Song. Universal manipulation policy network for articulated objects. *IEEE Robotics and Automation Letters (RAL)*, 7(2):2447–2454, 2022.

[198] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2019.

[199] Wenzhen Yuan, Siyuan Dong, and Edward H. Adelson. Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors*, 17(12), 2017.

[200] Wenzhen Yuan, Yuchen Mo, Shaoxiong Wang, and Edward H Adelson. Active clothing material perception using tactile sensing and deep learning. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 4842–4849. IEEE, 2018.

[201] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of British Machine Vision Conference (BMVC)*, 2016.

[202] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2020.

[203] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R. Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan Dafle, Rachel Holladay, Isabella Morona, Prem Qu Nair, Druck Green, Ian Taylor, Weber Liu, Thomas Funkhouser, and Alberto Rodriguez. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *The International Journal of Robotics Research (IJRR)*, 41(7):690–705, 2022.

[204] Amy Zhang, Harsh Satija, and Joelle Pineau. Decoupling Dynamics and Reward for Transfer Learning. *CoRR*, 2018.

[205] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[206] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.

[207] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.

[208] Tao Zhou and Bertram E. Shi. Simultaneous learning of the structure and kinematic model of an articulated body from point clouds. In *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, pages 5248–5255, 2016.

[209] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and imitation learning for diverse visuomotor skills. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.