UNIVERSITY OF CALIFORNIA SAN DIEGO

Modeling-based Optimization for Robotic Manipulation

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Zhiao Huang

Committee in charge:

Professor Hao Su, Chair
Professor Sicun Gao, Co-Chair
Professor Albert Ren-haur Chern
Professor Michael T Tolley
Professor Zhuowen Tu

2024

The Dissertation of Zhiao Huang is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

DEDICATION

To my family, who have always been there for me.

EPIGRAPH

Go to the roots of these calculations! Group the operations. Classify them according to their complexities rather than their appearances! This, I believe, is the mission of future mathematicians. This is the road on which I am embarking in this work.

Évariste Galois

At each level of complexity, entirely new properties appear, and the understanding of the new behaviors requires research which I think is as fundamental in its nature as any other.

P. W. Anderso

Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.

Ludwig Wittgenstein

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

PREFACE

When I first embarked on my journey in machine learning and ventured into the world of robotic manipulation, I was surprised by the power of gradients in training large neural networks. However, I soon became disillusioned by their limitations in exploring complex search spaces. While sufficiently large neural networks can easily memorize vast amounts of data, they often struggle when tasked with discovering unknown structures not present in the training data. For instance, when trying to solve a new mathematical problem or uncover abstract concepts, simple Stochastic Gradient Descent (SGD) frequently falls short.

From that moment, I became perplexed by the relationship between machine learning and optimization. In my personal statement before beginning my Ph.D. studies, I expressed my confusion as follows:

> *Learning can be considered as a mysterious memorization tool for efficient search, so the learning complexity is directly related to the size of search space. Interestingly, just like the graphical model, the complexity can be reduced by factorizing the search space, where human priors are introduced... SGD, our basic tool to build the deep learning world, seems to be not enough to explore those hidden spaces even if we know the factorization.*

From that point onward, I became captivated by the enigmatic relationship between learning and search. As I delved deeper into robotic manipulation and reinforcement learning, I discovered that these concepts are intricately interwoven across every aspect of robotics and machine learning. For example, optimization is used to train any neural network. In reinforcement learning (RL), we train a world model to optimize the policy while the policy itself models a certain distribution of actions. One can later combine the model and the learned policy to do the search.

This dissertation encapsulates my exploration in pursuit of answers to the questions that have fueled my curiosity. First, optimization is often more challenging than modeling, much like the distinction between P and NP. This principle generally holds true in learning as well. Memorizing the solution to an NP-hard problem is non-trivial. A good practice is to

transform the it into one of learning a model to verify solutions and then use search to find the solution. This approach helps explain why model-based methods with perception modules tend to generalize better than end-to-end neural networks. Second, policy learning itself can be seen as modeling the optimal solution or the current search space during exploration. Gradient descent, for example, is a rudimentary linear model of the environment, and more can be achieved through more sophisticated modeling of the policy or the search space. Third, by properly modeling the optimization problem, we can factorize the original problem into a series of sub-optimization problems, leveraging the concept of 'optimal substructure.'

As the famous Chinese saying goes, we should unite action and knowledge. Similarly, I believe that modeling and optimization are two sides of the same coin and should be unified. In my view, learning involves both the modeling of problems and the optimization for better solutions, and robotics is fundamentally about how to model and optimize the world. I hope this explanation, though superficial, can inspire readers and offer a preliminary answer to the questions I had before beginning my Ph.D. studies.

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to Professor Hao Su for his invaluable guidance and support as my Ph.D. advisor. He generously gave me the freedom to explore topics that sparked my interest. Through his mentorship, I have learned that curiosity, vision, and keen attention to detail are fundamental to great research.

I am also profoundly thankful to Professor Sicun Gao for his insightful contributions and unwavering support as my co-chair. Additionally, I extend my sincere thanks to Professor Albert Ren-haur Chern, Professor Michael Tolley, and Professor Zhuowen Tu, whose expertise and guidance have been instrumental in the completion of this dissertation.

I extend my heartfelt gratitude to my collaborators, including Professor Chuang Gan, who taught me the importance of impact and guided my exploration of soft-body research. His support has been invaluable to my research journey. I am deeply thankful to Sizhe Li, Yuanming Hu, Tao Du, Siyuan Zhou, Litian Liang, Ling Zhan, Xuanlin Li, Feng Chen, Yewen Pu, and Chunru Lin for their significant contributions to the research projects presented in this dissertation. I am also grateful for what I have learned from Professor Josh Tenenbaum, Professor Jiajun Wu, Professor David Held, and Professor Shuran Song during collaboration. I am incredibly thankful to all members of Su Lab for the beautiful years I have spent. I also thank Yunzhu Li, Xingyu Lin, Tao Chen, Zhenjia Xu, Driess Danny, Haochen Shi, Huazhe Xu, Chaoyi Zhang, Tiange Luo, Hao Tang, and Kaizhi Yang for their collaboration and support. I want to thank Yue Zhang for proofreading this dissertation and the people who attended and supported my thesis defense. If I have missed anyone, please accept my sincere apologies. I am grateful to all of you.

I am especially grateful to my family, who have always stood by my side, providing constant love and support. Their unwavering belief in me and encouragement have been the foundation of my resilience throughout this challenging academic journey. I am deeply thankful for their patience, understanding, and the strength they have given me every step of the way.

Chapter 2, in full, is a reprint of the material published in International Conference on Learning Representations 2021. PlasticineLab: A Soft-Body Manipulation Benchmark with

Differential Physics; Huang, Zhiao; Hu, Yuanming; Du, Tao; Zhou, Siyuan; Su, Hao; Tenenbaum, Joshua B.; Gan, Chuang. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material published in International Conference on Machine Learning 2023. Reparameterized Policy Learning for Multimodal Trajectory Optimization; Huang, Zhiao; Liang, Litian; Ling, Zhan; Li, Xuanlin; Gan, Chuang; Su, Hao. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, is a reprint of the material published in Advances in Neural Information Processing Systems 32 (2019). Mapping State Space using Landmarks for Universal Goal Reaching; Huang, Zhiao; Liu, Fangchen; Su, Hao. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in full, is a reprint of the material published in Advances in Neural Information Processing Systems 36 (2023). DiffVL: Scaling Up Soft Body Manipulation using Vision-Language Driven Differentiable Physics; Huang, Zhiao; Chen, Feng; Pu, Yewen; Lin, Chunru; Gan, Chuang; Su, Hao. The dissertation author was the primary investigator and author of this paper.

VITA

2014-2018 Bachelor of Engineering, Tsinghua University

2018-2024 Doctor of Philosophy, University of California San Diego

PUBLICATIONS

1. **Zhiao Huang***, Feng Chen*, Yewen Pu, Chunru Lin, Hao Su+, Chuang Gan+. DiffVL: Scaling Up Soft Body Manipulation using Vision-Language Driven Differentiable Physics, Neurips 2023

2. Zhan Ling, Yunhao Fang, Xuanlin Li, **Zhiao Huang**, Mingu Lee, Roland Memisevic, Hao Su. Deductive Verification of Chain-of-Thought Reasoning, Neurips 2023

3. **Zhiao Huang**, Litian Liang, Zhan Ling, Xuanlin Li, Chuang Gan, Hao Su. Reparameterized Policy Learning for Multimodal Trajectory Optimization, ICML 2023 oral

4. Stone Tao, Xiaochen Li, Tongzhou Mu, **Zhiao Huang**, Yuzhe Qin, Hao Su. Abstract-to-Executable Trajectory Translation for One-Shot Task Generalization, ICML 2023

5. Zhenjia Xu, Zhou Xian, Xingyu Lin, Cheng Chi, **Zhiao Huang**, Chuang Gan†, Shuran Song†. RoboNinja: Learning an Adaptive Cutting Policy for Multi-Material Objects, RSS 2023

6. Sizhe Li*, **Zhiao Huang***, Tao Chen, Tao Du, Hao Su, Joshua B Tenenbaum, Chuang Gan. DexDeform: Dexterous Deformable Object Manipulation with Human Demonstrations and Differentiable Physics, ICLR 2023

7. Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, **Zhiao Huang**, Rui Chen, Hao Su. ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills, ICLR 2023 Spotlight

8. Xingyu Lin, Carl Qi, Yunchu Zhang, Yunzhu Li, **Zhiao Huang**, Katerina Fragkiadaki, Chuang Gan, David Held, Planning with Spatial-Temporal Abstraction from Point Clouds for Deformable Object Manipulation, CORL 2022

9. Danny Driess, **Zhiao Huang**, Yunzhu Li, Russ Tedrake, Marc Toussaint, Learning Multi-Object Dynamics with Compositional Neural Radiance Fields, CORL 2022

10. Haochen Shi*, Huazhe Xu*, **Zhiao Huang**, Yunzhu Li, Jiajun Wu. RoboCraft: Learning to See, Simulate, and Shape Elasto-Plastic Objects with Graph Networks. RSS 2022

11. Xingyu Lin, **Zhiao Huang**, Yunzhu Li, Joshua B Tenenbaum, David Held, Chuang Gan. DiffSkill: Skill Abstraction from Differentiable Physics for Deformable Object Manipulations with Tools. ICLR 2021

12. Sizhe Li*, **Zhiao Huang***, Tao Du, Hao Su, Joshua B Tenenbaum, Chuang Gan. Contact Points Discovery for Soft-Body Manipulations with Differentiable Physics. ICLR 2021 Spotlight

13. Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, **Zhiao Huang**, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. NeurIPS 2021 Dataset Track

14. **Zhiao Huang**, Xiaochen Li and Hao Su, Efficient Hierarchical Navigation and Manipulation by Constraint-induced Option-Reward Design, RSS 2021 Workshop on Declarative and Neurosymbolic Representations in Robot Learning and Control

15. **Zhiao Huang**, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B. Tenenbaum, Chuang Gan. "PlasticineLab: A Soft-Body Manipulation Benchmark with Differential Physics". ICLR 2021 Spotlight

16. Hao Tang, **Zhiao Huang**, Jiayuan Gu, Bao-Liang Lu, Hao Su. "Towards Scale-Invariant Graph-related Problem Solving by Iterative Homogeneous Graph Neural Networks". Neurips 2020

17. Tiange Luo, Kaichun Mo, **Zhiao Huang**, Jiarui Xu, Siyu Hu, Liwei Wang, Hao Su. "Learning to Group: A Bottom-Up Framework for 3D Part Discovery in Unseen Categories". ICLR 2020

18. **Zhiao Huang***, Fangchen Liu* and Hao Su. "Mapping State Space using Landmarks for Universal Goal Reaching". NeurIPS 2019

19. Guangxiang Zhu, **Zhiao Huang** and Chongjie Zhang. "Object-Oriented Dynamics Predictor". NeurIPS 2018

20. Alejandro Newell, **Zhiao Huang** and Jia Deng. "Associative Embedding: End-to-End Learning for Joint Detection and Grouping", NIPS 2017

21. **Zhiao Huang**, Erjing Zhou, Zhimin Cao. "Coarse-to-Fine Face Alignment with Multi-Scale Local Patch Regression". arXiv:1511.04901

FIELDS OF STUDY

Major Field: Computer Science (Robotics, Reinforcement Learning and Simulation)

ABSTRACT OF THE DISSERTATION

Modeling-based Optimization for Robotic Manipulation

by

Zhiao Huang

Doctor of Philosophy in Computer Science

University of California San Diego, 2024

Professor Hao Su, Chair
Professor Sicun Gao, Co-Chair

This dissertation explores the intersection of modeling and optimization in robotics, focusing on the development of efficient and effective systems for robotic manipulation. The primary objective is to study how to integrate modeling techniques with optimization processes, a concept we term ”modeling-based optimization.”

We first introduce a differentiable physics simulator for soft-body manipulation, demonstrating the power of environment modeling in policy learning. By simulating elastoplastic materials such as plasticine, we benchmark reinforcement learning (RL) and gradient-based optimization methods, highlighting the strengths and limitations of each approach. The findings

reveal that while gradient-based methods excel in environments with well-modeled physics, they struggle with long-term planning and multi-stage tasks.

To address these challenges, we propose a reparameterized policy gradient method, which leverages latent variable models to facilitate exploration and avoid local minima. This approach integrates generative models to enhance policy expressiveness and improve performance in hard-exploration tasks. We further extend the concept of hierarchical policy modeling by introducing graph-based and vision-language-driven methods. These techniques enable robots to plan and execute long-horizon tasks by abstracting the search space and using human-like instructions to guide complex manipulations.

The contributions of this thesis include the development of novel algorithms for soft-body manipulation, hierarchical policy modeling, and the integration of generative models with reinforcement learning. These advancements offer new insights into the relationship between learning, modeling, and optimization in robotics.

Chapter 1

Introduction

1.1 Learning-based Robotic Manipulation

The early 2010s to 2020s marked a transformative era for artificial intelligence. Generative AI, particularly large language models, has had a significant impact on our daily lives. Meanwhile, robotics—or embodied AI—emerged as one of the most compelling applications, bridging the gap between digital intelligence and the real world, allowing us to fully harness the potential of artificial intelligence.

In robotics, we want to find policies to manipulate robots or agents in the complex physical world. To this end, we usually frame the problem as a policy learning problem within Markov decision processes (MDPs) [199, 182]. For example, reinforcement learning aims to find a parameterized policy π_θ that maximizes expected reward $E_{s_0\sim p(s_0)}[V^{\pi_\theta}(s_0)] = E_{\tau\sim\pi_\theta, s_0\sim p(s_0)}[R_\gamma(\tau)]$, where $V^{\pi_\theta}(s_0)$ is the value function of the initial state s_0. The trajectory $\tau = \{s_0, a_0, s_1, \ldots, s_t, a_t, \ldots\}$ is a sequence of states $\{s_t\}$ and actions $\{a_t\}$ at different time steps. The discounted reward of a trajectory $R_\gamma(\tau)$ is $\sum_t \gamma^t R(s_t, a_t)$ measure how well the trajectory is. In real world, the agent may only observe partial information, forming a partially observable Markov decision process (POMDP).

Embodied AI has been significantly fueled by deep learning and reinforcement learning breakthroughs, which have accelerated the development of increasingly sophisticated robotic algorithms and systems. The traditional "sense, plan, and act" can levearge advanced vision

models to identify object poses or determine optimal grasp positions for object picking [11]. However, these methods often face limitations when dealing with soft bodies or can only work limited categories of actions.

With the success in applications like Atari [168] and AlphaGo [225], as well as advancements in algorithms like SAC [74], PPO [216], deep reinforcement learning (RL) have gained traction in robotics. RL has succeeded in solving grasping and other challenging tasks [182, 125], allowing us to search policies given any black box environments. However, model-free RL algorithms are prone to getting stuck in local minima without proper reward engineering. Additionally, its high sample complexity limits its practical application, with most RL research still predominantly conducted in simulation.

While RL faces challenges, the rise of large language models and the success of generative AI have shifted attention back to data-driven approaches. One can incorporate human demonstrations to guide RL agents through imitation learning. Moreover, supervised learning have also shown significant success. Works, such as [43, 112, 33], by leveraging large-scale robot datasets, trained models that exhibit impressive generalizability. Diffusion policies or similar generative models have shown the potential to generate diverse and realistic samples from a limited number of human demonstrations [275, 29]. These data-driven methods require vast data. To overcome these challenges, researchers have worked on developing more cost-effective teleoperation systems and robots [275, 30], as well as exploring the potential of learning policies from large-scale web videos [220]. However, it is still unclear how to obtain sufficient data and even how much data is needed to train a generalizable robot policy.

How can we resolve the data dilemma in embodied AI? What is the most suitable approach for learning effective policies? I believe the answer lies in fully harnessing the power of generative models.

Instead of relying solely on collecting real world data, another approach is to use advanced environment modeling techniques (simulators) and policy optimization method to search for better policies or generate additional data within simulators. The learned policy can then be

deployed in the real world. Numerous simulation platforms have been developed [45, 14, 18, 117, 193, 260, 210, 262, 58, 59], and the feasibility of simulation to real world transfer (sim2real) has been demonstrated [72]. These achievements have demonstrated the potential of using simulators, or environment models, to reduce the data needed for training robot policies.

The deep entanglement of modeling and optimization in policy learning creates new opportunities for tackling complex manipulation tasks. In this thesis, I aim to explore the synergies between models and policies through "modeling-based optimization", presenting it as a promising direction for robotic manipulation.

1.2 Modeling-based Optimization in Robotics

1.2.1 Modeling and Optimization

What is modeling, and what is optimization? From a probabilistic perspective, "modeling" involves inferring the distributions of specific data. A "generative model," denoted as $p(X)$, represents a probabilistic distribution that allows for sampling and generating data. This is typically achieved using a parameterized neural network trained by maximizing the likelihood of the observed data, or a lower-bound approximation. In robotics, we model the distribution for various data types, such as the state-action trajectory τ, the environment model $p(s_{t+1} \mid s_t, a_t)$, or the policy $\pi(a \mid s)$.

Embodied AI is related to optimization, finding a policy distribution such that the sampled action will maximize the expected rewards. This is important especially when we need to find new solutions to new problems, when the solution is not covered by the data we collected.

Following the probabilistic perspective, optimization can be viewed as a problem of inferring $p(X)$ given $\log p(X)$ up to a scale. Let us consider the trajectory optimization problem to find a trajectory τ that maximizes the reward $R(\tau)$. The "reinforcement learning as inference framework" [242, 243, 245, 278, 108, 124] defines optimality as an additional variable $p(O \mid \tau) \propto e^{R(\tau)/\mathscr{T}}$, where $\mathscr{T}$ is a temperature scalar, turning the reward maximization problem

into a probabilistic inference framework of inferring the posterior of the optimal trajectories. Specifically, the prior distribution of the trajectory is: $p(\tau) = p(s_1)\prod_{t=1}^{T} p(a_t \mid s_t)p(s_{t+1} \mid s_t, a_t)$, where $p(a_t \mid s_t)$ is a known prior action distribution, e.g., a Gaussian distribution. Thus, one can compute the density of optimality:

$$p(O) = \int p(O \mid \tau)p(\tau)\, d\tau.$$

The posterior distribution of optimal trajectories becomes:

$$p(\tau \mid O) = \frac{p(O \mid \tau)p(\tau)}{\int p(O \mid \tau)p(\tau)\, d\tau}.$$

In maximum entropy framework [73], we can apply the evidence lower bound [114]:

$$\log p(O) \geq \mathbb{E}_{\tau\sim\pi}\left[\log p(O \mid \tau) + \log p(\tau) - \log \pi(\tau)\right],$$

leading to a practical algorithm for reinforcement learning (RL).

Directly sampling trajectories or using gradient descent to maximize this objective often encounters significant challenges, such as getting trapped in local minima and dealing with high sample complexity, calling for a better combination of search and optimization.

1.2.2 Environment Modeling

Modeling a low-level environment through a simulator is arguably the most straightforward way of leveraging generative models into the policy learning and optimization process. Model-based RL [75, 78, 47] and sim2real method [119, 182, 87] all fall in this realm. The advantages of environment modeling are two-fold. First, it decouples the policy learning problem into a verification problem and searching problem, enabling us to leverage the existing solver to solve the challenging non-convex optimization problem. Second, it is kind of widely believed it is easier to learn a model to verify a solution than to directly find the optimal solution; this leads

to the fact that a model-based approach often requires less amount of data than directly learning the policy if we properly inject inductive bias into the model, like human knowledge about the physical world. Properly leveraging the simulator can also help us reduce the data required in the real world.

These benefits make model-based optimization already widely used in robotics. For an audience familiar with optimization, you must know that a gradient, whether a policy gradient or an analytical gradient, actually represents a linear model of the loss function; that is why we can use descent to find optimal solutions. On the other hand, a physical simulator, widely used in the robot algorithm verifications and the sim2real paradigm, indeed serves as a real-world model. As we assume that the simulator is a model of the world, we can safely use it to optimize the policy and deploy the policy to the real world. Moreover, recent model-based RL or huge video models enable us to learn world models from data and use those models to either optimize the policy or plan for better solutions.

1.2.3 Hierarchical Policy Modeling

Hierarchical reinforcement learning (HRL) is considered one of the most efficient approaches to solving long-horizon robotics tasks. The idea originates in the fact that long-horizon tasks can be decomposed into small subtasks. Leveraging these optimal substructures, we can design the policy into a composition of multiple policies, where a low-level policy handles the subtasks and a high-level policy coordinates different policies to finish the whole task. When we have well-defined sub-skills, this approach has been rigorously tested and proven to be highly effective. However, the challenge arises when we need to find such a decomposition. What if we don't have this factorization and hope the optimization algorithm to discover the factorization? This remains an open question, but the proven effectiveness of the approach instills confidence in its potential.

1.3 Overview of Techniques and Contributions

This dissertation extends previous modeling techniques in robotic manipulation for building a scalable and general robot learning system. First, we can model the physical dynamics that are more suitable for policy optimization. Chapter 2 describes how we model soft bodies like Plasticine using a differentiable physical simulator, demonstrating the power of environment modeling and its gradients in policy learning. Second, we not only model the environment but also model the policy itself, as well as the process of searching for better policies. I introduce the "reparameterized policy gradient" in Chapter 3, where we build a theoretical framework for latent variable policies, provide empirical evidence for the necessities of the hierarchical policy modeling, and illustrate how the density model can guide exploration during the exploration. Finally, we illustrate two approaches to model hierarchical policies by leveraging graphs (Chapter 4) and vision-languages (Chapter 5). These approaches leverage the optimal substructure in the problem to decompose the optimization problems and build hierarchical policy models.

1.3.1 Environment Modeling for Soft Body Manipulation

In PlasticineLab [96], we built the first differentiable physics simulation for elastic and plastic soft body manipulation. This fully-featured differentiable physical engine supports elastic and plastic deformation, soft-rigid material interaction, and a tailored contact model for differentiability. We design multiple tasks to benchmark existing RL and gradient-based optimization methods and make thorough comparisons.

In this work, we not only incorporate deformable objects into the simulation and research of robotic manipulation but also demonstrate the feasibility of using the analytical model's gradient to optimize robot policy. This work inspires future research in soft body manipulation [148, 264, 130], differentiable physics optimization [261, 185, 178] and sim2real [145].

1.3.2 Learning Multimodal Policy with Reparameterized Policy Gradient

By modeling the environment, we can use gradient-based or sampling-based methods to search for optimal robot policies. However, object manipulation involves a vast search space, which presents challenges for both approaches: sampling-based methods struggle with inefficiency, while gradient-based methods often get trapped in local minima.

This naturally leads to the question: Can we achieve more efficient optimization and exploration beyond gradient descent while retaining the efficiency of gradient-based solvers? The answer is yes, and the key lies in developing a policy that can capture multimodal distributions, allowing us to sample and to optimize with gradients.

Our work in reparameterized policy learning [97] represents our efforts to explore policy optimization from a probabilistic modeling perspective. This idea originated from studying reinforcement learning in continuous optimization. We identified the unimodal Gaussian policy as a key factor causing gradient-based optimization and continuous RL methods to get stuck in local optima. The insight led us to focus on generative models. By following the common design of generative models with latent distributions, we derived a latent variable policy using variational inference, demonstrating that a hierarchical policy can naturally emerge to model the posterior of an optimal trajectory. This approach enables the reparameterized policy to combine global search with local gradient-based optimization, effectively bypassing local optima and partially validating the motivation for a hierarchical policy.

In practice, we utilize a learned world model to optimize our latent variable policy. Additionally, we found it crucial to model the exploration process by fitting a density model that measures the entropy of state visitation, rather than focusing solely on the actions of the policy. By integrating these elements, we establish a framework that models a multimodal policy, the environment, and the previously visited states, fully harnessing the potential of generative models in all aspects of policy learning.

1.3.3 Modeling Hierarchical Policy Using Graph and Language

While [97] demonstrates a powerful exploration algorithm for tackling challenging tasks, it is still not sufficient for real-world applications due to the large amount of data required for effective exploration and latent space learning. In the worst-case scenario, the policy may need to traverse the entire state space to find a meaningful reward signal. Additionally, constructing abstractions and hierarchical structures is not without cost; learning a useful latent space demands trajectories explored across different modalities. If we continue to treat policy learning as a black-box problem and only apply hierarchy in the latent space, there will be no way to further reduce the data requirements.

To address this issue, rather than solely learning the latent space, the best approach is to maintain the hierarchical structure we've developed for the latent variable policy while incorporating more model priors into the latent space. This allows us to explicitly factorize the optimization problem into simpler subproblems.

From a policy modeling perspective, we should carefully design the policy's latent or skill space to either reduce the low-level search space or enable the rapid learning of high-level skills. Fortunately, robotic manipulation in a 3D environment presents unique structures that we can exploit. We demonstrate that a subgoal-based graph planning method can effectively facilitate robot motion planning, while vision-language models can guide the robot in solving long-horizon manipulation tasks.

Landmark Map for Combining RL and Search

In our early work [98] (Chapter 4), we proposed a graph-based environment modeling method for long-horizon robot locomotion tasks in MDPs with sparse rewards, in which exploration and routing across remote states are both extremely challenging. Our method explicitly hierarchically models the environment, with a high-level dynamic landmark-based map abstracting the visited state space and a low-level value network to derive precise local decisions. We use the farthest point sampling to select landmark states from experience, which has improved

exploration compared with simple uniform sampling. This approach built an abstract model of the original environment, enabling the combination of high-level search and low-level policy optimization, which is a very effective policy in real-world robot tasks.

Vision-language Driven Trajectory Optimization

The graph-based approach is primarily limited to navigation. For object manipulation, which involves a diverse range of objects, materials, skills, and trajectories, finding a suitable low-dimensional space for planning is challenging. In Chapter 5, we describe vision-language task description as a more powerful sub-goal representation other than state/goal embedding.

The idea originates from exploring how language can be used to describe the process of soft body manipulation to enhance a differentiable physics solver. Challenges arise when the process involves complex geometrical information. For instance, while it is easy to state the goal of ”making a bun,” it is much harder to convey what the bun looks like without directly showing an image.

One potential solution to better describe such processes is to use vision-language representations, which integrate natural language and geometric shapes. Our DiffVL framework describes a long horizon soft body manipulation problem as a sequence of 3D scenes or keyframes. Natural language instructions describe the motion of the objects and the tool to use across adjacent keyframes. We built GUI tools to allow non-expert users to annotate soft body manipulation tasks inspired by real-life scenes, and we leverage LLM to translate those natural language instructions as well as the geometry subgoals into machine-interpretable optimization objectives, thus enabling us to use differentiable physics' solver to synthesize the trajectory of solving the tasks. In this work, we illustrated that language is a powerful tool for describing physical constraints and can serve as a bridge between humans and robots. Moreover, the task annotation and trajectory optimization scheme can be easily scaled up to collect extensive data for robot policy training. This, in turn, provides a way to solve the need for more data in robotics.

1.4 Additional Work Done During My Doctoral Career

My research primarily focuses on optimizing and modeling techniques for robotic manipulation. However, I have been fortunate to collaborate on various impactful projects that have expanded my expertise and enriched my understanding of the field.

Following our work on PlasticineLab, we investigated how demonstrations could significantly accelerate the dexterous manipulation of soft bodies [130]. By generating demonstrations using differentiable physics in simulation, we enhanced the real-world performance of robots in complex tasks [144, 145, 264]. Our efforts also led to introducing a geometric-aware contact loss [131], which markedly improved the manipulation of multi-stage soft bodies.

Beyond physical simulation, I collaborated on projects employing neural networks to learn physical models from simulated environments and real-world data. A highlight of this work was our exploration of the dynamics of real-world soft bodies, where we leveraged various 3D representations to capture its behavior accurately [42, 222].

Moreover, I was privileged to contribute to the ManiSkill project [70]. I also engaged in a range of research projects beyond robotics, including computer vision [151, 266], the study of graph neural networks [239], and the development of the multi-view dataset [141]. These collaborations have been instrumental in shaping my research.

Chapter 2

PlasticineLab: A Soft-Body Manipulation Benchmark with Differentiable Physics

Simulated virtual environments serve as one of the main driving forces behind developing and evaluating skill learning algorithms. However, previous environments typically only simulate rigid body physics. Additionally, the simulation process usually does not provide gradients that might be useful for planning and control optimizations. We introduce a new differentiable physics benchmark called *PasticineLab*, which includes a diverse collection of soft body manipulation tasks. In each task, the agent uses manipulators to deform the plasticine into a desired configuration. The underlying physics engine supports *differentiable elastic and plastic deformation* using the DiffTaichi system, posing many under-explored challenges to robotic agents. We evaluate several reinforcement learning (RL) methods and gradient-based methods on this benchmark. Experimental results suggest that 1) RL-based approaches struggle to solve most of the tasks efficiently; 2) gradient-based approaches, by optimizing open-loop control sequences with the built-in differentiable physics engine, can rapidly find a solution within tens of iterations, but still fall short on multi-stage tasks that require long-term planning. We expect that PlasticineLab will encourage the development of novel algorithms that combine differentiable physics and RL for more complex physics-based skill learning tasks. PlasticineLab is publicly available [1].

[1] Project page: http://plasticinelab.csail.mit.edu

2.1 Introduction

Virtual environments, such as Arcade Learning Environment (ALE) [15], MuJoCo [244], and OpenAI Gym [18] have significantly benefited the development and evaluation of learning algorithms on intelligent agent control and planning. However, existing virtual environments for skill learning typically involves *rigid-body* dynamics only. Research on establishing standard *soft-body* environments and benchmarks is sparse, despite the wide range of applications of soft bodies in multiple research fields, e.g., simulating virtual surgery in healthcare, modeling humanoid characters in computer graphics, developing biomimetic actuators in robotics, and analyzing fracture and tearing in material science.

Compared to its rigid-body counterpart, soft-body dynamics is much more intricate to simulate, control, and analyze. One of the biggest challenges comes from its infinite degrees of freedom (DoFs) and the corresponding high-dimensional governing equations. The intrinsic complexity of soft-body dynamics invalidates the direct application of many successful robotics algorithms designed for rigid bodies only and inhibits the development of a simulation benchmark for evaluating novel algorithms tackling soft-body tasks.

In this work, we aim to address this problem by proposing PlasticineLab, a novel benchmark for running and evaluating 10 soft-body manipulation tasks with 50 configurations in total. These tasks have to be performed by complex operations, including pinching, rolling, chopping, molding, and carving. Our benchmark is highlighted by the adoption of *differentiable physics* in the simulation environment, providing for the first time analytical gradient information in a soft-body benchmark, making it possible to conduct supervised learning with gradient-based optimization. In terms of the soft-body model, we choose to study plasticine (Figure 2.1, left), a versatile elastoplastic material for sculpturing. Plasticine deforms elastically under small deformation, and plastically under large deformation. Compared to regular elastic soft bodies, plasticine establishes more diverse and realistic behaviors and brings challenges unexplored in previous research, making it a representative medium to test soft-body manipulation algorithms

(Figure 2.1, right).

We implement PlasticineLab, its gradient support, and its elastoplastic material model using Taichi [93], whose CUDA backend leverages massive parallelism on GPUs to simulate a diverse collection of 3D soft-bodies in real time. We model the elastoplastic material using the Moving Least Squares Material Point Method [92] and the von Mises yield criterion. We use Taichi's two-scale reverse-mode differentiation system [91] to automatically compute gradients, including the numerically challenging SVD gradients brought by the plastic material model. With full gradients at hand, we evaluated gradient-based planning algorithms on all soft-robot manipulation tasks in PlasticineLab and compared its efficiency to RL-based methods. Our experiments revealed that gradient-based planning algorithms could find a more precious solution within tens of iterations with the extra knowledge of the physical model. At the same time, RL methods may fail even after 10K episodes. However, gradient-based methods lack enough momentum to resolve long-term planning, especially on multi-stage tasks. These findings have deepened our understanding of RL and gradient-based planning algorithms. Additionally, it suggests a promising direction of combining both families of methods' benefits to advance complex planning tasks involving soft-body dynamics. In summary, we contribute in this work the following:

- We introduce, to the best of our knowledge, the first skill learning benchmark involving elastic and plastic soft bodies.
- We develop a fully-featured differentiable physical engine, which supports elastic and plastic deformation, soft-rigid material interaction, and a tailored contact model for differentiability.
- The broad task coverage in the benchmark enables a systematic evaluation and analysis of representative RL and gradient-based planning algorithms. We hope such a benchmark can inspire future research to combine differentiable physics with imitation learning and RL.

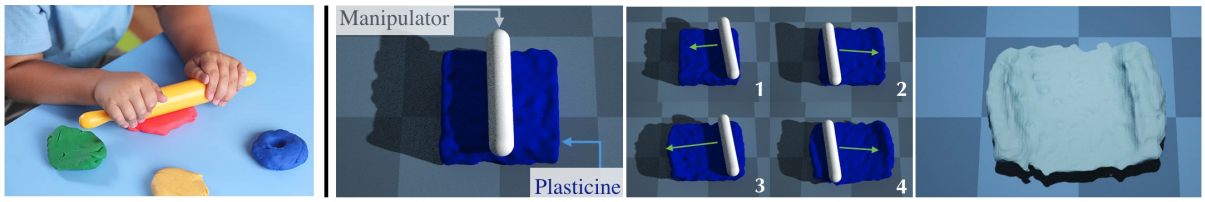


Figure 2.1. Left: A child deforming a piece of plasticine into a thin pie using a rolling pin. **Right:** The challenging **RollingPin** scene in PlasticineLab. The agent needs to flatten the material by rolling the pin back and forth, so that the plasticine deforms into the target shape.

2.2 Related Work

Learning in virtual environments Recently, several simulation platforms and datasets have been developed to facilitate the research and development of new algorithms on RL and robotics. An incomplete list includes RL Benchmark [45], DeepMind Lab [14], OpenAI Gym [18], AI2-THOR [117], VirtualHome [193], Gibson [260], Habitat [210], SAPIEN [262], and TDW [58, 59]. We observe a tendency to use full-physics simulators with realistic dynamics. However, most of these virtual environments are based on rigid-body physical engines, such as MuJoCo [244] and PyBullet [34]. While some support soft-body dynamics in theory (e.g., TDW and SAPIEN is based on NVIDIA PhysX [1] that supports particle simulation), none has provided the assets and tasks for soft-body manipulation. Differentiable information is also missing in these engines. We fill in this gap with our PlasticineLab benchmark.

Differentiable physics engines Differentiable physics engines for machine learning have gained increasing popularity. One family of approaches *approximates* physical simulators using neural networks, which are naturally differentiable [13, 25, 172, 138]. A more direct and accurate approach is to implement physics-based simulators using differentiable programming systems, e.g., standard deep learning frameworks equipped with automatic differentiation tools [38, 36, 213, 82]. These systems are typically redistricted to explicit time integration. Other approaches of evaluating simulation gradient computation include using the adjoint methods to differentiate implicit time integrators [16, 65], LCP [36] and leveraging QR decompositions [140, 194].

Closely related to our work is ChainQueen [94], a differentiable simulator for *elastic* bodies, and DiffTaichi [91], a system to automatically generate high-performance simulation gradient kernels. Our simulator is originated from ChainQueen but with significant modifications in order to add our novel support for *plasticity and contact* gradients.

Trajectory optimization Our usage of differentiable simulation in planning soft-body manipulation is closely related to trajectory optimization, a topic that has been extensively studied in robotics for years and has been applied to terrestrial robots [192, 48, 36], aerial robots [54, 240, 228], and, closest to examples in our work, robotic manipulators [158, 134]. Both trajectory optimization and differentiable physics formulate planning as an optimization problem and derive gradients from governing equations of the dynamics [241]. Still, the problem of motion planning for soft-body manipulation is under exploration in both communities because of two challenges: first, the high DoFs in soft-body dynamics make traditional trajectory optimization methods computationally prohibitive. Second, and more importantly, contacts between soft bodies are intricate to formulate in a concise manner. Our differentiable physics simulator addresses both issues with the recent development of DiffTaichi [91], unlocking gradient-based optimization techniques on planning for soft-body manipulation with high DoFs ($> 10,000$) and complex contact.

Learning-based soft body manipulation Finally, our work is also relevant to prior methods that propose learning-based techniques for manipulating physics systems with high degrees of freedom, e.g. cloth [140, 259], fluids [153, 88], and rope [265, 259]. Compared to our work, all of these prior papers focused on providing solutions to specific robot instances, while the goal of our work is to propose a comprehensive benchmark for evaluating and developing novel algorithms in soft-body research. There are also considerable works on soft manipulators [66, 39]. Different from them, we study soft body manipulation with rigid manipulators.

2.3 The PlasticineLab Learning Environment

PlasticineLab is a collection of challenging soft-body manipulation tasks powered by a differentiable physics simulator. All tasks in PlasticineLab require an agent to deform one or more pieces of 3D plasticine with rigid-body manipulators. The underlying simulator in PlasticineLab allows users to execute complex operations on soft bodies, including pinching, rolling, chopping, molding, and carving. We introduce the high-level design of the learning environment in this section and leave the technical details of the underlying differentiable simulator in Section 2.4.

2.3.1 Task representation

PlasticineLab presents 10 tasks with the focus on soft-body manipulation. Each task contains one or more soft bodies and a kinematic manipulator, and the end goal is to deform the soft body into a target shape with the planned motion of the manipulator. Following the standard reinforcement learning framework [18], the agent is modeled with the Markov Decision Process (MDP), and the design of each task is defined by its state and observation, its action representation, its goal definition, and its reward function.

Markov Decision Process An MDP contains a state space $\mathscr{S}$, an action space $\mathscr{A}$, a reward function $\mathscr{R} : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \to \mathbb{R}$, and a transition function $\mathscr{T} : \mathscr{S} \times \mathscr{A} \to \mathscr{S}$. In PlasticineLab, the physics simulator determines the transition between states. The goal of the agent is to find a stochastic policy $\pi(a|s)$ to sample action $a \in \mathscr{A}$ given state $s \in \mathscr{S}$, that maximizes the expected cumulative future return $E_\pi\left[\sum_{t=0}^{\infty} \gamma^t \mathscr{R}(s_t, a_t)\right]$ where $0 < \gamma < 1$ is the discount factor.

State The state of a task includes a proper representation of soft bodies and the end effector of the kinematic manipulator. Following the widely used particle-based simulation methodology in previous work, we represent soft-body objects as a particle system whose state includes its particles' positions, velocities, and strain and stress information. Specifically, the particle state is encoded as a matrix of size $N_p \times d_p$ where N_p is the number of particles. Each

row in the matrix consists of information from a single particle: two 3D vectors for position and velocities and two 3D matrices for deformation gradients and affine velocity fields [103], all of which are stacked together and flattened into a d_p-dimensional vector.

Being a kinematic rigid body, the manipulator's end effector is compactly represented by a 7D vector consisting of its 3D position and orientation represented by a 4D quaternion, although some DoFs may be disabled in certain scenes. For each task, this representation results in an $N_m \times d_m$ matrix encoding the full states of manipulators, where N_m is the number of manipulators needed in the task and $d_m = 3$ or 7 depending on whether rotation is needed. Regarding the interaction between soft bodies and manipulators, we implement one-way coupling between rigid objects and soft bodies and fix all other physical parameters such as particle's mass and manipulator's friction.

Observation While the particle states fully characterize the soft-body dynamics, its high DoFs are hardly tractable for any planning and control algorithm to work with directly. We thus downsample N_k particles as landmarks and stack their positions and velocities (6D for each landmark) into a matrix of size $N_k \times 6$, which serves as the observation of the particle system. Note that landmarks in the same task have fixed relative locations in the plasticine's initial configuration, leading to a consistent particle observation across different configurations of the task. Combining the particle observation with the manipulator state, we end up having $N_k \times 6 + N_m \times d_m$ elements in the observation vector.

Action At each time step, the agent is instructed to update the linear (and angular when necessary) velocities of the manipulators in a kinematic manner, resulting in an action of size $N_m \times d_a$ where $d_a = 3$ or 6 depending on whether rotations of the manipulators are enabled in the task. For each task, we provide global $A_{\min}, A_{\max} \in \mathbb{R}^{d_a}$, the lower and upper bounds on the action, to stabilize the physics simulation.

Goal and Reward Each task is equipped with a target shape represented by its mass tensor, which is essentially its density field discretized into a regular grid of size N_{grid}^3. At each time step t, we compute the mass tensor of the current soft body S_t. Discretizing both

target and current shapes into a grid representation allows us to define their similarity by comparing densities at the same locations, avoiding the challenging problem of matching particle systems or point clouds. The complete definition of our reward function includes a similarity metric as well as two regularizers on the high-level motion of the manipulator: $\mathscr{R} = -c_1\mathscr{R}_{\text{mass}} - c_2\mathscr{R}_{\text{dist}} - c_3\mathscr{R}_{\text{grasp}} + c_4$, where $\mathscr{R}_{\text{mass}}$ measures the L_1 distance between the two shapes' mass tensors as described above, $\mathscr{R}_{\text{dist}}$ is the dot product of the signed distance field (SDF) of the target shape and the current shape's mass tensor, and $\mathscr{R}_{\text{grasp}}$ encourages the manipulators to be closer to the soft bodies. Positive weights c_1, c_2, c_3 are constant for all tasks. The bias c_4 is selected for each environment to ensure the reward is nonnegative initially.

2.3.2 Evaluation Suite

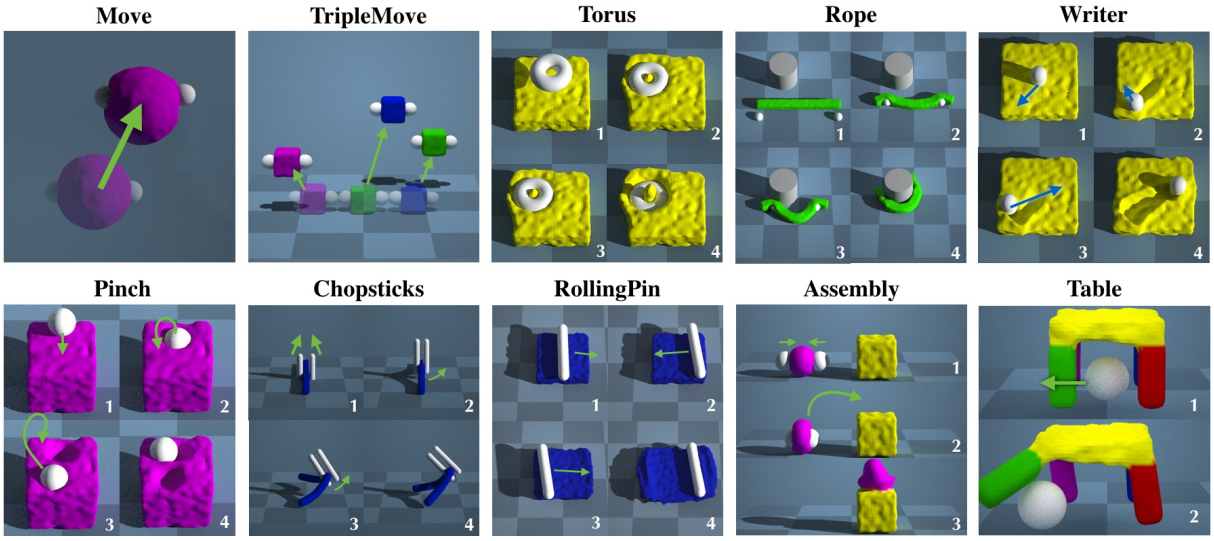


Figure 2.2. Tasks and reference solutions of PlasticineLab. Certain tasks require multi-stage planning.

PlasticineLab has a diverse collection of 10 tasks (Figure 2.2). We describe four representative tasks here, and the remaining six tasks are detailed in Section 2.6.2.

These tasks, along with their variants in different configurations, form an evaluation suite for benchmarking performance of soft-body manipulation algorithms. Each task has 5 variants (50 configurations in total) generated by perturbing the initial and target shapes and the initial

locations of manipulators.

Rope The agent needs to wind a rope, modeled as a long plasticine piece, around a rigid pillar with two spherical manipulators. The pillar's position varies in different configurations.

Writer The agent manipulates a "pen" (represented using a vertical capsule), to sculpt a target scribble on cubic plasticine. For each configuration, we generate the scribble by drawing random 2D lines on the plasticine surface. The three-dimensional action controls the tip of the pen.

Chopsticks The agent uses a pair of chopsticks, modeled as two parallel capsules, to pick up the rope on the ground and rotate it into the target location. The manipulator has 7 DoFs: 6 for moving and rotating the pair of chopsticks and 1 for controlling the distance between them.

RollingPin The agent learns to flatten a "pizza dough", which is modeled as a plasticine box, with a rigid rolling pin. We simulate the rolling pin with a 3-DoF capsule: 1) the pin can descend vertically to press the dough; 2) the pin can rotate along the vertical axis to change its orientation; 3) the agent can also roll the pin over the plasticine to flatten it.

2.4 Differentiable Elastoplasticity Simulation

The simulator is implemented using Taichi [93] and runs on CUDA. Continuum mechanics is discretized using the Moving Least Squares Material Point Method (MLS-MPM) [92], a simpler and more efficient variant of the B-spline Material Point Method (MPM) in computer graphics [230]. Both Lagrangian particles and Eulerian background grids are used in the simulator. Material properties, include position, velocity, mass, density, and deformation gradients, are stored on Lagrangian particles that move along with the material, while particle interactions and collisions with rigid bodies are handled on the background Eulerian grid. We refer the reader to ChainQueen [94] and DiffTaichi [91] for more details on differentiable MPM with elastic materials. Here we focus on extending the material model with *(differentiable) plasticity*, a defining feature of plasticine. We leverage Taichi's reverse-mode automatic differentiation

system [91] for most of the gradient evaluations.

von Mises yield criterion We use a simple von Mises yield criterion for modeling plasticity, following the work of [60]. According to the von Mises yield criterion, a plasticine particle yields (i.e., deforms plastically) when its second invariant of the deviatoric stress exceeds a certain threshold, and a projection on the deformation gradient is needed since the material "forgets" its rest state. This process is typically called *return mapping* in MPM literature.

Return mapping and its gradients Following [116] and [60], we implement the return mapping as a 3D projection process on the singular values of the deformation gradients of each particle. This means we need a singular value decomposition (SVD) process on the particles' deformation gradients, and we provide the pseudocode of this process in Section 2.6.1. For backpropagation, we need to evaluate *gradients of SVD*. Taichi's internal SVD algorithm [161] is iterative, which is numerically unstable when automatically differentiated in a brute-force manner. We use the approach in [248] to differentiate the SVD. For zeros appearing in the denominator when singular values are not distinct, we follow [104] to push the absolute value of the denominators to be greater than 10^{-6}.

Contact model and its softened version for differentiability We follow standard MPM practices and use grid-base contact treatment with Coulomb friction (see, for example, [230]) to handle soft body collision with the floor and the rigid body obstacles/manipulators. Rigid bodies are represented as time-varying SDFs. In classical MPM, contact treatments induce a drastic non-smooth change of velocities along the rigid-soft interface. To improve reward smoothness and gradient quality, we use a *softened* contact model during backpropagation. For any grid point, the simulator computes its signed distance d to the rigid bodies. We then compute a smooth *collision strength factor* $s = \min\{\exp(-\alpha d), 1\}$, which increases exponentially when d decays until $d = 0$. Intuitively, collision effects get stronger when rigid bodies get closer to the grid point. The positive parameter α determines the sharpness of the softened contact model. We linearly blend the grid point velocity before and after collision projection using factor s, leading to a smooth transition zone around the boundary and improved contact gradients.

2.5 Experiments

2.5.1 Evaluation Metrics

We first generate five configurations for each task, resulting in 50 different reinforcement learning configurations. We compute the normalized incremental IoU score to measure if the state reaches the goal. We apply the soft IoU [198] to estimate the distance between a state and the goal. We first extract the grid mass tensor S, the masses on all grids. Each value S^{xyz} stores how many materials are located in the grid point (x,y,z), which is always nonnegative. Let two states' 3D mass tensors be S_1 and S_2. We first divide each tensor by their maximum magnitude to normalize its values to $[0,1]$: $\bar{S_1} = \frac{S_1}{\max_{ijk} S_1^{ijk}}$ and $\bar{S_2} = \frac{S_2}{\max_{ijk} S_2^{ijk}}$. Then the softened IoU of the two state is calculated as $IoU(S_1,S_2) = \frac{\sum_{ijk} \bar{S_1}\bar{S_2}}{\sum_{ijk} \bar{S_1}+\bar{S_2}-\bar{S_1}\bar{S_2}}$. We refer readers to Section 2.6.6 for a better explanation for the soft IoU. The final normalized incremental IoU score measures how much IoU increases at the end of the episode than the initial state. For the initial state S_0, the last state S_t at the end of the episode, and the goal state S_g, the normalized incremental IoU score is defined as $\frac{IoU(S_t,S_g)-IoU(S_0,S_g)}{1-IoU(S_0,S_g)}$. For each task, we evaluate the algorithms on five configurations and report an algebraic average score.

2.5.2 Evaluations on Reinforcement Learning

We evaluate the performance of the existing RL algorithms on our tasks. We use three SOTA model-free reinforcement learning algorithms: Soft Actor-Critic (SAC) [73], Twin Delayed DDPG (TD3) [56], and Policy Proximal Optimization (PPO) [216]. We train each algorithm on each configuration for 10000 episodes, with 50 environment steps per episode.

Figure 2.3 shows the normalized incremental IoU scores of the tested reinforcement learning algorithms on each scene. Most RL algorithms can learn reasonable policies for **Move**. However, RL algorithms can hardly match the goal shape exactly, which causes a small defect in the final shape matching. We notice that it is common for the RL agent to release the objects during exploration, leading to a free-fall of plasticine under gravity. Then it becomes challenging

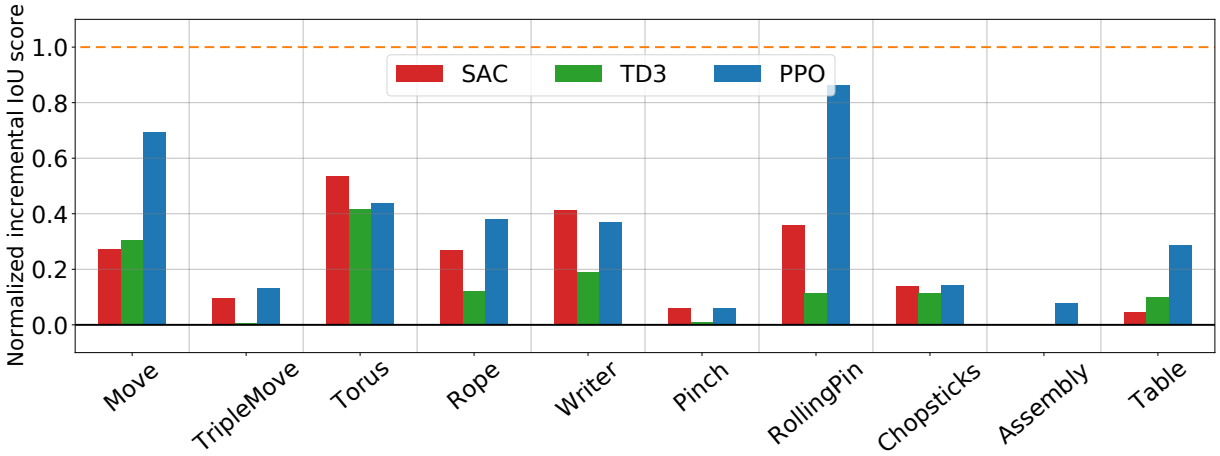


Figure 2.3. The final normalized incremental IoU score achieved by RL methods within 10^4 epochs. Scores lower than 0 are clamped. The dashed orange line indicates the theoretical upper limit.

for the agent to regrasp the plasticine, leading to training instability and produces unsatisfactory results. The same in **Rope**, agents can push the rope towards the pillar and gain partial rewards, but they fail to move the rope around the pillar in the end. Increasing the numbers of manipulators and plasticine boxes causes significant difficulties in **TripleMove** for RL algorithms, revealing their deficiency in scaling to high dimensional tasks. In **Torus**, the performance seems to depend on the initial policy. They could sometimes find a proper direction to press manipulators, but occasionally, they fail as manipulators never touch the plasticine, generating significant final score variance. Generally, we find that PPO performs better than the other two. In **RollingPin**, both SAC and PPO agents find the policy to go back and forth to flatten the dough, but PPO generates a more accurate shape, resulting in a higher normalized incremental IoU score. We speculate that our environment favors PPO over algorithms dependent on MLP critic networks. We suspect it is because PPO benefits from on-policy samples while MPL critic networks might not capture the detailed shape variations well.

In some harder tasks, like **Chopsticks** that requires the agent to carefully handle the 3D rotation, and **Writer** that requires the agent to plan for complex trajectories for carving the traces, the tested algorithm seldom finds a reasonable solution within the limited time (10^4 episodes). In

Assembly, all agents are stuck in local minima easily. They usually move the spherical plasticine closer to the destination but fail to lift it up to achieve an ideal IoU. We expect that a carefully designed reward shaping, better network architectures, and fine-grained parameter tuning might be beneficial in our environments. In summary, plasticity, together with the soft bodies' high DoFs, poses new challenges for RL algorithms.

2.5.3 Evaluations on Differentiable Physics for Trajectory Optimization

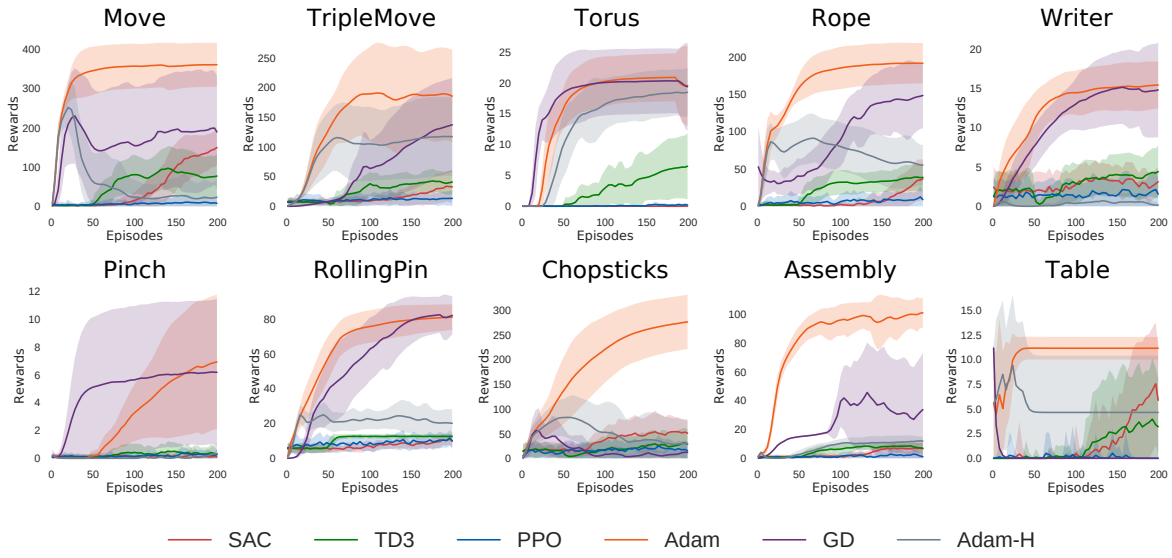


Figure 2.4. Rewards and their variances in each task w.r.t. the number of episodes spent on training. We clamp the reward to be greater than 0 for a better illustration.

Thanks to the built-in differentiable physics engine in PlasticineLab, we can apply gradient-based optimization to plan open-loop action sequences for our tasks. In gradient-based optimization, for a certain configuration starting at state s, we initialize a random action sequence $\{a_1, \ldots, a_T\}$. The simulator will simulate the whole trajectory, accumulate the reward at each time step, and do back-propagation to compute the gradients of all actions. We then apply a gradient-based optimization method to maximize the sum of rewards. We assume all information of the environment is known. This approach's goal is not to find a controller that can be executed in the real world. Instead, we hope that differentiable physics can help find a solution efficiently and pave roads for other control or reinforcement/imitation learning algorithms.

Table 2.1. The averaged normalized incremental IoU scores and the standard deviations of each method. Adam-H stands for optimizing on the hard contact model with Adam optimizer. We train RL agent for 10000 episodes and optimizing for 200 episodes for gradient-based approaches.

Env	Move	Tri. Move	Torus	Rope	Writer
SAC	0.27 ± 0.27	0.12 ± 0.11	$\mathbf{0.53 \pm 0.42}$	0.29 ± 0.20	$\mathbf{0.41 \pm 0.23}$
TD3	0.31 ± 0.20	0.00 ± 0.02	0.42 ± 0.42	0.12 ± 0.15	0.19 ± 0.18
PPO	$\mathbf{0.69 \pm 0.15}$	$\mathbf{0.13 \pm 0.09}$	0.44 ± 0.38	$\mathbf{0.38 \pm 0.19}$	0.37 ± 0.17
Adam	$\mathbf{0.90 \pm 0.12}$	$\mathbf{0.35 \pm 0.20}$	0.77 ± 0.39	$\mathbf{0.59 \pm 0.13}$	0.62 ± 0.27
GD	0.51 ± 0.39	0.24 ± 0.17	$\mathbf{0.79 \pm 0.37}$	0.41 ± 0.17	$\mathbf{0.69 \pm 0.29}$
Adam-H	0.05 ± 0.15	0.26 ± 0.15	0.72 ± 0.23	0.21 ± 0.09	0.00 ± 0.00
Env	Pinch	RollingPin	Chopsticks	Assembly	Table
SAC	0.05 ± 0.08	0.36 ± 0.30	0.13 ± 0.08	0.00 ± 0.00	0.04 ± 0.12
TD3	0.01 ± 0.02	0.11 ± 0.02	0.11 ± 0.07	0.00 ± 0.00	0.10 ± 0.16
PPO	$\mathbf{0.06 \pm 0.09}$	$\mathbf{0.86 \pm 0.10}$	$\mathbf{0.14 \pm 0.09}$	$\mathbf{0.06 \pm 0.17}$	$\mathbf{0.29 \pm 0.28}$
Adam	$\mathbf{0.08 \pm 0.08}$	$\mathbf{0.93 \pm 0.04}$	$\mathbf{0.88 \pm 0.08}$	$\mathbf{0.90 \pm 0.10}$	$\mathbf{0.01 \pm 0.01}$
GD	0.03 ± 0.05	0.89 ± 0.11	0.03 ± 0.04	0.27 ± 0.36	0.00 ± 0.00
Adam-H	0.00 ± 0.02	0.26 ± 0.12	0.02 ± 0.06	0.03 ± 0.03	$\mathbf{0.00 \pm 0.01}$

In Figure 2.4, we demonstrate the optimization efficiency of differentiable physics by plotting the reward curve w.r.t. the number of environment episodes and compare different variants of gradient descent. We test the Adam optimizer (Adam) and gradient descent with momentum (GD). We use the soft contact model to compute the gradients. We compare the Adam optimizer with a hard contact model (Adam-H). For each optimizer, we modestly choose a learning rate of 0.1 or 0.01 per task to handle the different reward scales across tasks. Notice that we only use the soft contact model for computing the gradients and search for a solution. We evaluate all solutions in environments with hard contacts. In Figure 2.4, we additionally plot the training curve of reinforcement learning algorithms to demonstrate the efficiency of gradient-based optimization. Results show that optimization-based methods can find a solution for challenging tasks within tens of iterations. Adam outperforms GD in most tasks. This may be attributed to Adam's adaptive learning rate scaling property, which fits better for the complex loss surface of the high-dimensional physical process. The hard contact model (Adam-H) performs worse than the soft version (Adam) in most tasks, which validates the intuition that a soft model

is generally easier to optimize.

Table 2.1 lists the normalized incremental IoU scores, together with the standard deviations of all approaches. The full knowledge of the model provides differentiable physics a chance to achieve more precious results. Gradient descent with Adam can find the way to move the rope around the pillar in **Rope**, jump over the sub-optimal solution in **Assembly** to put the sphere above the box, and use the chopsticks to pick up the rope. Even for **Move**, it often achieves better performance by better aligning with the target shape and a more stable optimization process.

Some tasks are still challenging for gradient-based approaches. In **TripleMove**, the optimizer minimizes the particles' distance to the closet target shape, which usually causes two or three plasticines to crowd together into one of the target locations. It is not easy for the gradient-based approaches, which have no exploration, to jump out such local minima. The optimizer also fails on the tasks that require multistage policies, e.g., **Pinch** and **Writer**. In **Pinch**, the manipulator needs to press the objects, release them, and press again. However, after the first touch of the manipulator and the plasticine, any local perturbation of the spherical manipulator doesn't increase the reward immediately, and the optimizer idles at the end. We also notice that gradient-based methods are sensitive to initialization. Our experiments initialize the action sequences around 0, which gives a good performance in most environments.

2.6 Additional Details

2.6.1 Simulator Implementation Details

von Mises plasticity return mapping pseudo code

Here we list the implementation of the forward return mapping [60]. Note the SVD in the beginning leads to gradient issues that need special treatments during backpropagation.

```
def von_Mises_return_mapping(F):
    # F is the deformation gradient before return mapping
    U, sig, V = ti.svd(F)
```

```
    epsilon = ti.Vector([ti.log(sig[0, 0]), ti.log(sig[1, 1])])
    epsilon_hat = epsilon - (epsilon.sum() / 2)
    epsilon_hat_norm = epsilon_hat.norm()
    delta_gamma = epsilon_hat_norm - yield_stress / (2 * mu)

    if delta_gamma > 0: # Yields!
        epsilon -= (delta_gamma / epsilon_hat_norm) * epsilon_hat
        sig = make_matrix_from_diag(ti.exp(epsilon))
        F = U @ sig @ V.transpose()
    return F
```

Parameters

We use a yield stress of 50 for plasticine in all tasks except **Rope**, where we use 200 as the yield stress to prevent the rope from fracturing. We use $\alpha = 666.7$ in the soft contact model.

Parallelism and performance

Our parallel mechanism is based on ChainQueen (Hu et al.,2019b). A single MPM substep involves three stages: particle to grid transform (p2g), grid boundary conditions (gridop), and grid to particle transform (g2p). In each stage, we use a parallel for-loop to loop over all particles (p2g) or grids (for gridop and g2p) to do physical computations and ap-ply atomic operators to write results back to particles/grids. Gradient computation, which needs to reverse the three stages, is automatically done by DiffTaich (Hu et al., 2020).

We benchmark our simulator's performance on each scene in Table 2.2. Note one step of our environment has 19 MPM substeps.

2.6.2 More Details on the Evaluation Suite

Move The agent uses two spherical manipulators to grasp the plasticine and move it to the target location. Each manipulator has 3 DoFs controlling its position only, resulting in a 6D action space.

Table 2.2. Performance on an NVIDIA GTX 1080 Ti GPU. We show the average running time for a single forward or forward + backpropagation step for each scene.

Env	Forward	Forward + Backward
Move	14.26 ms (70 FPS)	35.62 ms (28 FPS)
Tri.Move	17.81 ms (56 FPS)	41.88 ms (24 FPS)
Torus	13.77 ms (73 FPS)	35.69 ms (28 FPS)
Rope	15.05 ms (66 FPS)	38.70 ms (26 FPS)
Writer	14.00 ms (71 FPS)	36.04 ms (28 FPS)
Pinch	12.07 ms (83 FPS)	27.03 ms (37 FPS)
RollingPin	14.14 ms (71 FPS)	36.45 ms (27 FPS)
Chopsticks	14.24 ms (70 FPS)	35.68 ms (28 FPS)
Assembly	14.43 ms (69 FPS)	36.51 ms (27 FPS)
Table	14.00 ms (71 FPS)	35.49 ms (28 FPS)

TripleMove The agent operates three pairs of spherical grippers to relocate three plasticine boxes into the target positions. The action space has a dimension of 18. This task is challenging to both RL and gradient-based methods.

Torus A piece of cubic plasticine is fixed on the ground. In each configuration of the task, we generate the target shape by randomly relocating the plasticine and push a torus mold towards it. The agent needs to figure out the correct location to push down the mold.

Pinch In this task, the agent manipulates a rigid sphere to create dents on the plasticine box. The target shape is generated by colliding the sphere into the plasticine from random angles. To solve this task, the agent needs to discover the random motion of the sphere.

Assembly A spherical piece of plasticine is placed on the ground. The agent first deforms the sphere into a target shape and then moves it onto a block of plasticine. The manipulators are two spheres.

Table This task comes with a plasticine table with four legs. The agent pushes one of the table legs towards a target position using a spherical manipulator.

2.6.3 Reinforcement Learning Setup

We use the open-source implementation of SAC, PPO and TD3 in our environments. We list part of the hyperparameters in Table 2.3 for SAC, Table 2.5 for TD3 and Table 2.4 for PPO. We fix $c_1 = 10, c_2 = 10$ and $c_3 = 1$ for all environments' reward.

Table 2.3. SAC Parameters

gamma	0.99
policy lr	0.0003
entropy lr	0.0003
target update coef	0.0003
batch size	256
memory size	1000000
start steps	1000

Table 2.4. PPO Parameters

update steps	2048
lr	0.0003
entropy coef	0
value loss coef	0.5
batch size	32
horizon	2500

Table 2.5. TD3 Parameters

start timesteps	1000
batch size	256
gamma	0.99
tau	0.005
policy noise	0.2
noise clip	0.5

2.6.4 Ablation Study on Yield Stress

To study the effects of yield stress, we run experiments on a simple **Move** configuration (where SAC can solve it well) with different yield stress. We vary the yield stress from 10 to 1000 to generate 6 environments and train SAC on them. Figure 2.5 plots the agents' performances w.r.t. the number of training episodes. The agents achieve higher reward as the yield stress increase, especially in the beginning. Agents in high yield stress environments learn faster than those in lower yield stress environments. We attribute this to the smaller plastic deformation in higher yield stress environments. If we train the agents for more iterations, those in environments with yield stress larger than 100 usually converge to a same performance level, close to solving the problem. However, materials in environments with yield stress smaller than 100 tend to deform plastically, making it hard to grasp and move an object while not destroying its structure. This demonstrates a correlation between yield stress and task difficulty.

2.6.5 Optimization Efficiency of Differentiable Physics

We show the optimization efficiency of differentiable physics in Figure 2.6.

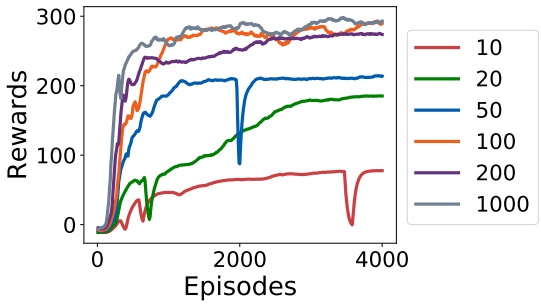


Figure 2.5. Rewards w.r.t. the number of training episode on 6 environments. Their yield stresses are 10, 20, 50, 100, 200, and 1000.

2.6.6 Explanation of Soft IoU

Let a, b be two numbers and can only be 0 or 1, then $ab = 1$ if and only if $a = b = 1$; $a + b - ab = 1$ if and only if at least one of a, b is 1. If the two mass tensors only contain value of 0 or 1, $\sum S_1 S_2$ equals the number of grids that have value 1 in both tensors, i.e., the "Intersection". For the same reason $\sum S_1 + S_2 - S_1 S_2$ counts the grids that $S_1 = 1$ or $S_2 = 1$, the "Union". Thus, the formula $\sum S_1 S_2 / \sum S_1 + S_2 - S_1 S_2$ computes the standard Intersection over Union (IoU). In our case, we assume the normalized mass tensor S_1 and S_2 are two tensors with positive values. Therefore, we first normalize them so that their values are between 0 and 1, then apply the previous formula to compute a "soft IoU," which approximately describes if two 3D shapes match.

2.7 Potential Research Problems to Study using PlasticineLab

Our environment opens ample research opportunities for learning-based soft-body manipulation. Our experiments show that differential physics allows gradient-based trajectory optimization to solve simple planning tasks extremely fast, because gradients provide strong and clear guidance to improve the policy. However, gradients will vanish if the tasks involve detachment and reattachment between the manipulators and the plasticine. When we fail to use gradient-based optimization that is based on local perturbation analysis, we may consider

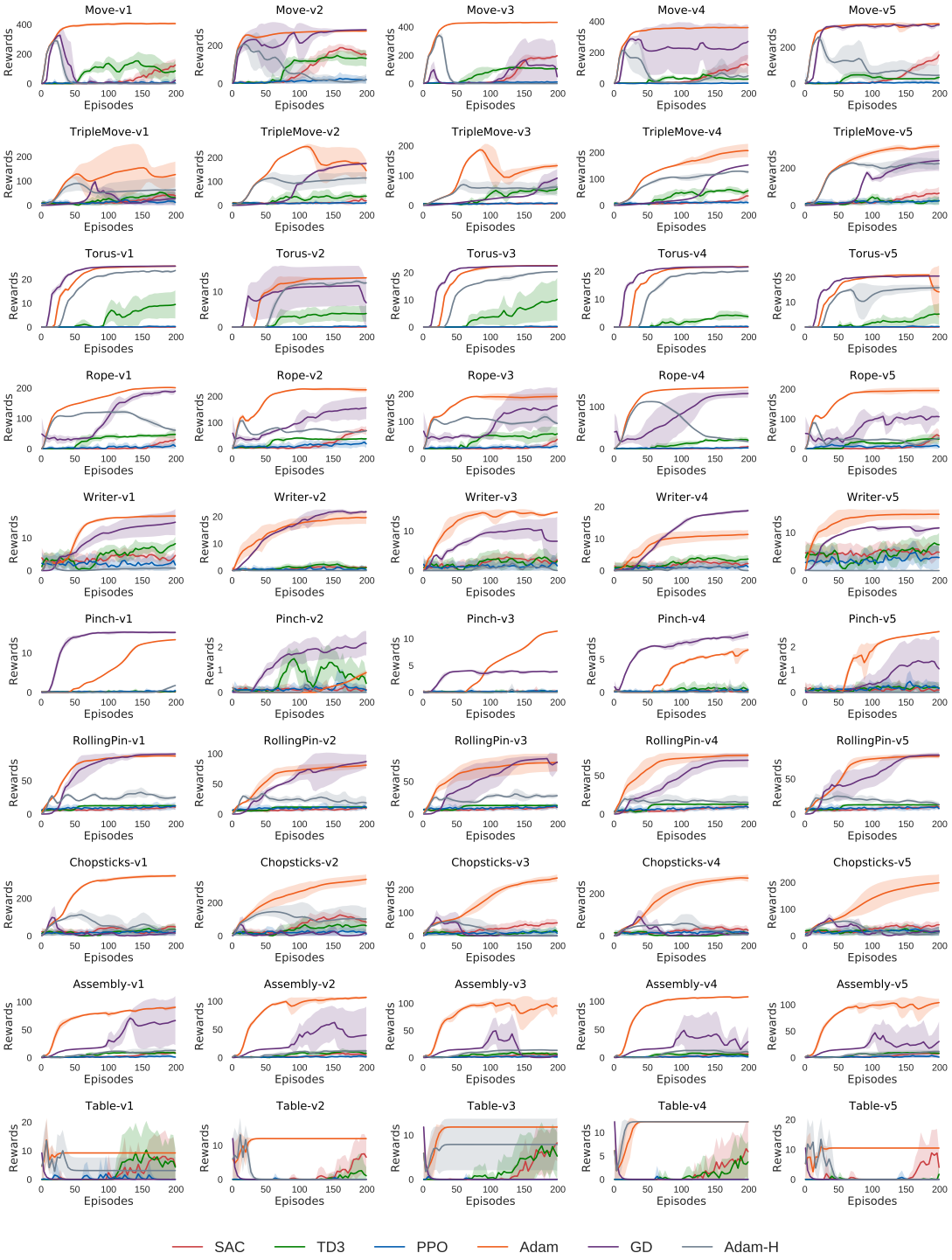


Figure 2.6. Rewards and variances in each configuration w.r.t. the number of episodes spent on training. We clamp the reward to be greater than 0 for a better illustration.

those methods that allow multi-step exploration and collect cumulative rewards, e.g., random search and reinforcement learning. Therefore, it is interesting to study how differentiable physics may be combined with these sampling-based methods to solve planning problems for soft-body manipulation.

Beyond the planning problem, it is also interesting to study how we shall design and learn effective controllers for soft-body manipulation in this environment. Experimental results (Section 2.5.2) indicate that there is adequate room for improved controller design and optimization. Possible directions include designing better reward functions for RL and investigating proper 3D deep neural network structures to capture soft-body dynamics.

A third interesting direction is to transfer the trained policy in PlasticineLab to the real world. While this problem is largely unexplored, we believe our simulator can help in various ways: 1. As shown in [63], MPM simulation results can accurately match the real world. In the future, we may use our simulator to plan a high-level trajectory for complex tasks and then combine with low-level controllers to execute the plan. 2. Our differential simulator can compute the gradient to physical parameters and optimize parameters to fit the data, which might help to close the sim2real gap. 3. PlasticineLab can also combine domain randomization and other sim2real methods [159]. One can customize physical parameters and the image renderer to implement domain randomization in our simulator. We hope our simulator can serve as a good tool to study real-world soft-body manipulation problems.

Finally, generalization is an important exploration direction. Our platform supports procedure generation and can generate and simulate various configurations with different objects [268], evaluating different algorithms' generalizability. PlasticineLab is a good platform to design rich goal-condition tasks, and we hope it can inspire future work.

2.8 Conclusion and Future Work

We presented PlasticineLab, a new differentiable physics benchmark for soft-body manipulation. To the best of our knowledge, PlasticineLab is the first skill-learning environment that simulates elastoplastic materials while being differentiable. The rich task coverage of PlasticineLab allows us to systematically study the behaviors of state-of-the-art RL and gradient-based algorithms, providing clues to future work that combines the two families of methods.

We also plan to extend the benchmark with more articulation systems, such as virtual shadow hands[2]. As a principled simulation method that originated from the computational physics community [233], MPM is convergent under refinement and has its own accuracy advantages. However, modeling errors are inevitable in virtual environments. Fortunately, apart from serving as a strong supervision signal for planning, the simulation gradient information can also guide systematic identification. This may allow robotics researchers to "optimize" tasks themselves, potentially simultaneously with controller optimization, so that sim-to-real gaps are automatically minimized.

We believe PlasticineLab can significantly lower the barrier of future research on soft-body manipulation skill learning, and will make its unique contributions to the machine learning community.

Acknowledgements

Chapter 2, in full, is a reprint of the material published in International Conference on Learning Representations 2021. PlasticineLab: A Soft-Body Manipulation Benchmark with Differential Physics; Huang, Zhiao; Hu, Yuanming; Du, Tao; Zhou, Siyuan; Su, Hao; Tenenbaum, Joshua B.; Gan, Chuang. The dissertation author was the primary investigator and author of this paper.

[2]https://en.wikipedia.org/wiki/Shadow_Hand

Chapter 3

Reparameterized Policy Learning for Multimodal Trajectory Optimization

We investigate the challenge of parametrizing policies for reinforcement learning (RL) in high-dimensional continuous action spaces. Our objective is to develop a multimodal policy that overcomes limitations inherent in the commonly-used Gaussian parameterization. To achieve this, we propose a principled framework that models the continuous RL policy as a generative model of optimal trajectories. By conditioning the policy on a latent variable, we derive a novel variational bound as the optimization objective, which promotes exploration of the environment. We then present a practical model-based RL method, called Reparameterized Policy Gradient (RPG), which leverages the multimodal policy parameterization and learned world model to achieve strong exploration capabilities and high data efficiency. Empirical results demonstrate that our method can help agents evade local optima in tasks with dense rewards and solve challenging sparse-reward environments by incorporating an object-centric intrinsic reward. Our method consistently outperforms previous approaches across a range of tasks. Code and supplementary materials are available on the project page https://haosulab.github.io/RPG/

3.1 Introduction

Reinforcement learning (RL) with *high-dimensional continuous action space* is notoriously hard despite its fundamental importance for many application problems such as robotic

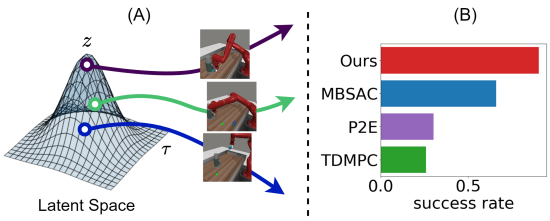


Figure 3.1. (A) Our method reparameterizes latent variables into multimodal policy to facilitate exploitation and exploration in continuous policy learning; (B) Average performance on 6 hard exploration tasks. Our method outperforms previous methods.

manipulation [182, 173]. In practice, popular frameworks [226, 74, 216] of deep RL formulate the continuous policy as a neural network that outputs a single-modal density function over the action space (e.g., a Gaussian distribution over actions). This formulation, however, breaks the promise of RL being a global optimizer of the return function because the single-modality policy parameterization introduces local minima that are hard to escape using gradients w.r.t. distribution parameters. Besides, a single-modality policy will significantly weaken the exploration ability of RL algorithms because the sampled actions are usually concentrated around the modality.

Although there are other candidates beyond the Gaussian distribution for policy parameterization, they often have limitations when used for continuous policy modeling. For example, Gaussian mixture models can only accommodate a limited number of modes; normalizing flow methods [204] can compute density values, but they may not be as numerically robust due to their dependency on the determinant of the network Jacobian; furthermore, normalizing flows must apply continuous transformations onto a continuously connected distribution, making it difficult to model disconnected modes [202]. Option-critic [12] represents policies with options and temporal structure, but it often requires specially designed option spaces for efficient learning, which motivates research on hierarchical imitation learning that uses demonstrations to avoid exploration problems [188, 51]. Skill discovery methods learn a population of skills without demonstrations or rewards by optimizing for diversity [49]. However, the separation of optimization and skill learning can be non-efficient as it expends effort on learning task-irrelevant

skills and may ignore more important ones that would benefit a specific task.

This work presents a principled framework for learning the continuous RL policy as a multimodal density function through multimodal action parameterization. We adopt a sequence modeling perspective [27] and view the policy as a density function over the entire trajectory space (instead of the action space)[278, 124]. This allows us to sample a population of trajectories that cover multiple modalities, enabling concurrent exploration of distant regions in the solution space. Additionally, we use a generative model to parameterize the multimodal policies, drawing inspiration from their success in modeling highly complex distributions such as natural images[68, 277, 205, 200]. We condition the policy on a latent variable z and use a powerful function approximator to "reparameterize" the random distribution z into the multimodal trajectory distribution [114], from which we can sample trajectories τ. This policy parameterization leads us to adopt the variational method [114, 74, 169] to derive a novel framework for modeling the posterior of the optimal trajectory using variational inference, which enables us to model multimodal trajectories and maximize the reward with a single objective.

This framework allows us to build Reparameterized Policy Gradient (RPG), a model-based RL method for multimodal trajectory optimization. The framework has two notable features: First, RPG combines the multimodal policy parameterization with a learned world model, enjoying the sample efficiency of the learned model and gradient-based optimization while providing the additional ability to jump out of the local optima; Second, we equip RPG with a novel density estimator to help the multimodal policy explore in the environments by maximizing the state entropy [80]. We verify the effectiveness of our methods on several robot manipulation tasks. These environments only provide sparse rewards when the agent successfully fully finishes the task, which is challenging for single-modal policies even when they are guided by intrinsic motivations. In comparison, our method is able to explore different modalities, improve the exploration efficiency, and outperform single-modal policies, as shown in Figure 3.1. Notably, our method is more robust than single-modal policies and consistently outperforms previous approaches across different tasks.

Our contributions are multifold: 1. We propose a variational policy learning framework that models the posterior of multimodal optimal trajectories for reward optimization. 2. We demonstrate that multimodal parameterization can help the policy escape local optima and accelerate exploration in continuous policy optimization. 3. When combined with a learned world model and a delicate density estimator, our method, RPG, is able to solve these challenging sparse-reward tasks more efficiently and reliably.

3.2 Related Work

Policy as Sequential Generative Model.

Maximum entropy reinforcement learning [242, 243, 245, 278, 108] can be viewed as variational inference in probabilistic graphical models [124] with optimality as an observed variable and sampled trajectories as latent variables. When the demonstration or a fixed dataset is provided in the *offline* RL setting [27, 203], policy learning is simplified as a sequence modeling task [27, 276, 203]. They use autoregressive models to learn the distribution of the whole trajectory, including actions, states, and rewards, and use the action prediction as policy. In our work, we learn a sequential generative model of policy for *online* RL via the variational method.

Variational Skill Discovery

Under additional assumptions of rewards, our method degenerates to skill discovery methods. However, previous skill discovery methods focus on unsupervised reinforcement learning [49, 2, 24] or diverse skill learning [120, 184]. These methods build latent variable policy and encourage the policy to reach states that are consistent with the sampled latent variables through a mutual information term as a reward. These methods do not consider reward maximization or exploration when learning the skills, making them differ from our method vastly. For example, [49, 2] does not optimize the learned skill for the environment rewards; [184] does not optimize the mutual information along trajectories; [120] needs to solve the optimization problem first before finding a diverse set of solutions. Moreover, these methods fix the latent

distributions, limiting their ability to achieve optimality when rewards are given. [160] also learns skills within a learned world model. However, it decouples the exploration and skill learning and needs offline data or data generated from other exploration policies to train the model. In contrast, we are motivated by the parameterization problems in *online* RL and jointly optimize the latent representation to model *optimal* trajectories. We show that learning a latent variable model benefits optimization and exploration and they can be considered together.

Hierarchical Methods

The hierarchical methods, e.g., option-critic [12], can be regarded as a special way of policy parameterization by conditioning the lower-level policy over a sequence of latent variables $z = (z_1, \cdots, z_T)$. Usually, most hierarchical RL methods need special designs for the latent space, e.g., state-based subgoals [118, 176, 175] or predefined skills [127] to avoid mode-collapse. [183] regularized options to maximize the mutual information between the action and the options, which are very relevant to ours. However, it does not model temporal structures as ours to ensure consistency along the trajectories. Goal-conditioned RL [8, 165, 176] can also be considered a special hierarchical method that uses states or goals to help parameterize the policy and has been proven efficient in exploration, but designing the goal space, sampling and generating goals in high-dimensional space is non-trivial. The specific reward design of goal-reaching tasks also makes extending goal-conditioned policies to general reward functions not easy.

Hierarchical imitation learning [71, 189, 218, 107, 152, 52] extracts temporal abstractions from demonstrations using generative models. For example, InfoGAN [137] and ASE [188] use adversarial training [69, 85] to imitate demonstrations. These works all rely on demonstrations rather than rewards to learn abstractions. [31] learns representation on the collected dataset with variational inference and then utilizes the trained model for planning or policy learning. The separation of the representation learning and reward maximization makes it differ from our methods: first, it requires a state reconstruction module to supervise the generative model, which is challenging for high-dimensional observations; second, it optimizes neither the latent

distribution nor the actions for the reward directly, thus requires additional planning procedure during the execution to find suitable actions.

3.3 Method

To overcome the limitations of single modality policies, we propose to use latent variables to parameterize multimodal policies in Section 3.3.1. We then propose a novel variational bound as the optimization objective to approximate the posterior of optimal trajectories in Section 3.3.2. The variational bound naturally combines maximum entropy RL and includes a term to encourage consistency [277] between the latent distribution and the sampled trajectories, preventing the policy from mode collapse. To optimize this objective in hard continuous control problems, we propose to learn a world model and build the Reparameterized Policy Gradient, a model-based latent variable policy learning framework in Section 3.3.3. We design intrinsic rewards in Section 3.3.3 to facilitate exploration. Figure 3.3 illustrates the whole pipeline.

3.3.1 Reparameterize Latent Variables for Multimodal Policy Learning

Policy parameterization matters.

In continuous RL, it is popular to model action distribution with a unimodal Gaussian distribution. However, theoretically, to make sure that the optimal policy will be captured by RL, the function class of continuous RL policies has to include density functions of arbitrary probabilistic distributions [235]. Consider maximizing a continuous reward function with two modalities as shown in Figure 3.2(A). When the action space is properly discretized, a SoftMax policy can model the multimodal distribution and find the global optimum after sampling over the entire action space as shown in Figure 3.2(B). However, discretization can lead to a loss of accuracy and efficiency. If we instead use a Gaussian policy $\mathcal{N}(\mu, \sigma^2)$ by the common practice in literature, we will have trouble – as shown in Figure 3.2(C), even if its standard deviation is so large to well cover both modalities, the policy gradient can push it towards the local optimum on the right side, causing it to fail to converge to the global optimum. To address the issue, a

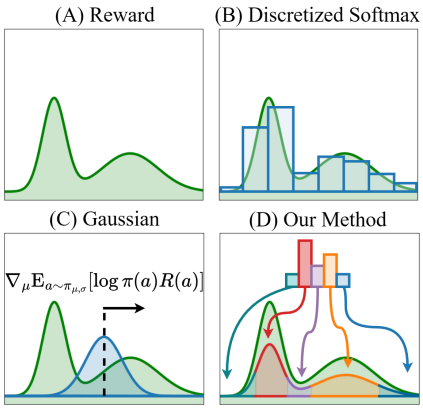


Figure 3.2. (A) rewards; (B); soft max policy over discrete action space; (C) single-modality Gaussian policy; (D) our methods reparameterize a random variable into multimodal distributions with neural networks.

more flexible policy parameterization is needed for continuous RL problems, one that is simple to sample and optimize.

Multimodal policy by reparameterizing latent variables

Motivated by recent developments in generative models that have shown superiority in modeling complex distributions [114, 86, 205, 200], we propose to parameterize policies using latent variables, as illustrated in Figure 3.2(D). Instead of adding random noise to perturb network outputs to generate an action distribution, we build a generative model of policy distribution by taking random noise as input and relying on powerful neural networks to transform it into actions of various modalities.

Formally, let $z \in \mathscr{Z}$ be a random variable, which can be either continuous or categorical. We design our "policy" as a joint distribution $\pi_\theta(z,\tau)$ of the latent z and the trajectory τ. This work considers a particular factorization of $\pi_\theta(z,\tau)$ that samples z in the beginning of each episode and then sample trajectory τ conditioning on z:

$$\pi_\theta(z,\tau) = p(s_1)\pi_\theta(z|s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|z,s_t) \tag{3.1}$$

where T is the length of the sampled trajectory.

One can use the policy gradient theorem [235], i.e., $\nabla J(\pi) = \mathbb{E}_\tau[R(\tau)\nabla \log p(\tau)]$ to

optimize the generative model policy. However, computing $p(\tau)$ needs to marginalize over z, i.e., computing $\int_z p(z,\tau)\,\mathrm{d}z$, which is often intractable when z is continuous. Besides, optimizing the marginal distribution $\log p(\tau)$ by gradient descent suffers from local optimality issues (e.g., using gradient descent to optimize Gaussian mixture models which have latent variables is not effective, so EM is often used instead [179]).

3.3.2 Variational Inference for Optimal Trajectory Modeling

To overcome these obstacles, following [242, 243, 245, 278, 108, 124, 74], we adopt variational method (maximum entropy RL) to directly optimize the joint distribution of the optimal policy without hassles of integrating over z.

The evidence lower bound

We learn $\pi_\theta(z,\tau)$ using variational inference [114, 74, 169]. Like an EM algorithm, we define an auxiliary distribution $p_\phi(z|\tau)$ to approximate the posterior distribution of z conditioning on τ using function approximators. This auxiliary distribution $p_\phi(z|\tau)$ helps to factorize the joint distribution of optimality O, latent z, and the trajectory τ as $p_\phi(O,z,\tau) = p(O|\tau)p_\phi(z|\tau)p(\tau)$. Treating $\pi_\theta(z,\tau)$ as the variational distribution, we can write the Evidence Lower Bound (ELBO) for the optimality O:

$$
\begin{aligned}
&\log p(O) \\
&= \underbrace{E_{z,\tau\sim\pi_\theta}\left[\log p_\phi(O,z,\tau) - \log \pi_\theta(z,\tau)\right]}_{\text{ELBO}} + D_{KL}(\pi_\theta(z,\tau)||p_\phi(z,\tau|O)) \\
&\geq E_{z,\tau\sim\pi_\theta}\left[\log p_\phi(O,\tau,z) - \log \pi_\theta(z,\tau)\right] \\
&= E_{z,\tau\sim\pi_\theta}\left[\log p(O,\tau) + \log p_\phi(z|\tau) - \log \pi_\theta(z,\tau)\right] \\
&= E_{z,\tau}\left[\underbrace{\log p(O|\tau)}_{\text{reward}} + \underbrace{\log p(\tau)}_{\text{prior}} + \underbrace{\log p_\phi(z|\tau)}_{\text{cross entropy}} - \underbrace{\log \pi_\theta(z,\tau)}_{\text{entropy}}\right]
\end{aligned}
\tag{3.2}
$$

If we optimize $\pi_\theta(z,\tau)$ and $p_\phi(z|\tau)$ using the gradient of the variational bound, the variational distribution $\pi_\theta(z,\tau)$ learns to model the optimal trajectory distribution $p(\tau|O)$.

How it works

ELBO contains four parts that can all be computed directly given the sampled z and τ (the environment probability $p(s_{t+1}|s_t,a_t)$ is canceled as in [124]). The first two parts are the predefined reward $\log p(O|\tau) = R(\tau)/\mathscr{T} + c$, where $\mathscr{T}$ is the temperature scalar, and c is the normalizing constant that can be ignored in optimization. The prior distribution $p(\tau)$ is assumed to be known. The third part is the log-likelihood of z, defined by our auxiliary distribution $p_\phi(z|\tau)$. It is easy to see that if we fix π_θ, maximize p_ϕ alone will minimize the cross-entropy $E_{z,\tau\sim\pi_\theta}[-\log p_\phi(z|\tau)]$, similar to the supervised learning of predicting z given τ. This achieves optimality when $p_\phi(z|\tau) = p_\theta(z|\tau) = \frac{\pi_\theta(z,\tau)}{\int_z \pi_\theta(z,\tau)dz}$, modeling the posterior of z for τ sampled from π_θ. On the other hand, by fixing ϕ, the policy π_θ is encouraged to generate trajectories that are easy to identify or classify; this helps to increase diversity and enforce consistency to avoid mode collapse, letting the network not ignore the latent variables. The fourth part is the policy entropy that enables maximum entropy exploration. Maximizing all terms together for the parameters θ and ϕ will minimize $D_{KL}(\pi_\theta(z,\tau)||p_\phi(z,\tau|O)) = D_{KL}(\pi_\theta(z,\tau)||p_\phi(z|\tau)p(\tau|O))$. The optimality can be achieved when $p_\phi(z|\tau)$ equals to $p(z|\tau)$, the true posterior of z. Then, $p_\theta(\tau) = p_\phi(z|\tau)p(\tau|O)/p(z|\tau) = p(\tau|O)$ where $p_\theta(\tau) = \int \pi_\theta(\tau,z)dz$ is the marginal distribution of τ sampled from π_θ.

Relationship with other methods

Our method is closely related to skill discovery methods [49, 160]. A skill discovery method usually uses mutual information $I(\tau,z) = H(\tau) - H(\tau|z)$ or $H(z) - H(z|\tau) \geq E_{z,\tau}[\log p_\phi(z|\tau) - \log p(z)]$ to encourage diversity. For example, DIYAN [49] directly optimizes mutual information to learn various skills without reward. Dropping out the reward term in Eq. 3.2 shows that the skill learning objective can be seamlessly embedded into the "RL as inference" framework with external reward, and there is no need to introduce the mutual infor-

mation term manually. Furthermore, the framework suggests we can model the posterior of the optimal trajectories, which enables us to unify generative modeling and trajectory optimization in a single framework. As for the relationship of our method with other generative models, we refer readers to a more thorough discussion in Section 3.6.

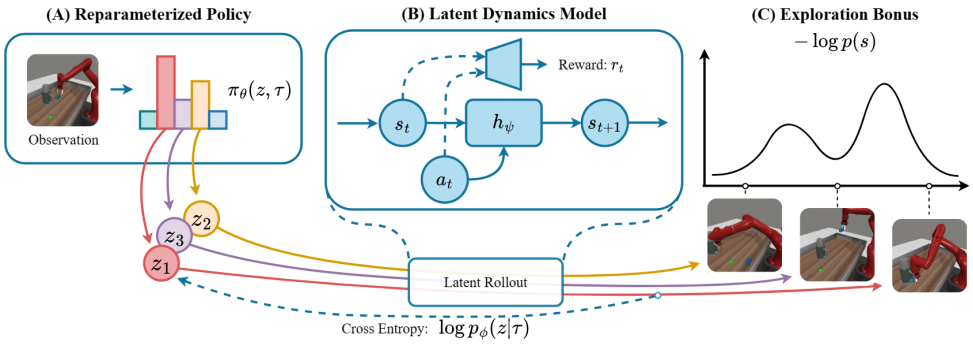


Figure 3.3. An overview of our model pipeline: A) a reparameterized policy from which we can sample latent variable z and action a given the latent state s; B) a latent dynamics model which can be used to forward simulate the dynamic process when a sequence of actions is known. C) an exploration bonus provided by a density estimator. Our Reparameterized Policy Gradient do multimodal exploration with the help of the latent world model and the exploration bonus.

3.3.3 Reparameterized Policy Gradient for Model-based Exploration

We now describe Reparameterized Policy Gradient (RPG), a model-based RL method with intrinsic motivation for sample efficient exploration in continuous control environments. We first simplify the right side of Eq. 3.2 using the factorization in Eq. 3.1 and assuming $\log p_\phi(z|\tau) = \sum_{t>0} \log p(z|s_t,a_t)$. Thus, the ELBO becomes $-\log \pi_\theta(z|s_1) + \sum_{t=1}^{\infty} R(s_t,a_t)/\mathscr{T} - \log \pi_\theta(a_t|s_t,z) + \log p_\phi(z|s_t,a_t)$, which can be optimized with an RL algorithm by maximizing the reward

$$\underbrace{R(s_t,a_t)/\mathscr{T}}_{r_t} \underbrace{-\alpha \log \pi_\theta(a_t|s_t,z) + \beta \log p_\phi(z|s_t,a_t)}_{r'_t},$$

where scalars α, β control the exploration and consistency. We use neural networks to model $\log p_\phi(z|s_t,a_t)$ and $\pi_\theta(a_t|s_t,z)$.

Model-based RL with Latent Variables

In our method Reparameterized Policy Gradient (RPG), we train a differentiable world model [75, 214, 267, 78] to improve data efficiency. The world model contains the following components: observation encoder $s_t = f_\psi(o_t)$, reward predictor $r_t = R_\psi(s_t, a_t)$, Q value $Q_t = Q_\psi(s_t, a_t, z)$ and dynamics $s_{t+1} = h_\psi(s_t, a_t)$.

Given any z and latent state $s_{t_0} = f_\psi(o_{t_0})$ at time step t_0, the learned dynamics network can generate an imaginary trajectory for any action sequence. If we sample actions from the policy $\pi_\theta(a_t|s_t, z)$ for $t \geq t_0$ and execute them in the latent model, it will produce a Monte-Carlo estimate for the value of s_{t_0} for optimizing the policy π_θ:

$$V_{\text{est}}(o_{t_0}, z) \approx \gamma^K (Q_{t_0+K} + r'_{t_0+K}) + \sum_{t=t_0}^{t_0+K-1} \gamma^{t-t_0} (r_t + r'_t) \tag{3.3}$$

We self-supervise the dynamics network to ensure state consistency without reconstructing observations as in [267, 78]. For any latent variable z and trajectory segments of length $K+1$ $\tau_{t_0:t_0+K} = \{o_{t_0}, a^{gt}_{t_0}, r^{gt}_{t_0}, o_{t_0+1}, \ldots, o_{t_0+K}\}$ sampled from the replay buffer, we execute actions $\{a^{gt}_t\}$ in the world model and use the following loss function to train the world model, as well as the Q function:

$$\begin{aligned} L_\psi(\tau) = \sum_{t=t_0}^{t_0+K-1} & L_1 \|s_{t+1} - \mathbf{ng}(f_\psi(o_{t+1}))\|^2 + L_2 (r_t - r^{gt}_t)^2 \\ & + L_3 (Q_t - \mathbf{ng}(r^{gt}_t + \gamma V_{\text{est}}(o_{t+1}, z)))^2 \end{aligned} \tag{3.4}$$

where $\mathbf{ng}(x)$ means stopping gradient and $L_1 = 1000, L_2 = L_3 = 0.5$ are constants to balance the loss.

Maximize State Entropy with Object-centric Randomized Network Distillation

For challenging continuous control tasks with sparse rewards, policies that maximize the action entropy of $\pi_\theta(a|s, z)$ usually have trouble obtaining a meaningful reward, making its exploration inefficient. We follow [80] to let the policy additionally maximize the entropy of the

discounted stationary state distribution $d_\pi(s) = (1-\gamma)\sum_{t=1}^{\infty}\gamma^t P(s_t = s|\pi)$.

We use the *object-centric* Randomized Network Distillation (RND) [23] as a simple and effective method to approximate the state density in continuous control tasks. RND uses a network $g_\theta(o_t)$ to distill the output of a random network $g'(o_t)$ by minimizing the difference $\|g_\theta(o_t) - g'(o_t)\|^2$ over states sampled by the current agent and treat the difference as the negative density of each observation o_t.

We make several modifications to the vallina RND to improve its performance for state vector observations in control problems. First, we inject object-prior to the RND estimator to make the policy sensitive to regions that include objects' position change. Specifically, before feeding objects' coordinates into the network, we apply positional encoding [249, 166] to turn all scalars x to a vector of $\{\sin(2^i x), \cos(2^i x)\}_{i=1,2,\ldots}$ for objects of interest (e.g., in robot manipulation, the end effector of the robot and the object). Second, we use a large replay buffer to store past states to avoid catastrophic forgetting [273]. We verified that it is necessary to normalize the RND's output to stabilize the training and make it an approximated density estimator. Lastly, to account for the latent world model, we relabel trajectories' rewards sampled from the replay buffer instead of estimating them directly in the latent model by reconstructing the observation.

An implicit benefit of a latent variable policy model is its ability to maximize the state entropy better, as will be shown in the experiments of Section 3.4.1. When combined with our RND method, RPG achieves much better state coverage while single modality policy cannot stabilize. The combination of multimodal policy learning and state entropy maximization accelerates the exploration of continuous control tasks with sparse rewards. We describe the whole algorithm in Alg. 1 and implementation details in Section 3.5.

3.4 Experiments

In this section, we first illustrate the potential of RPG in optimization and exploration through two example tasks. We then show that our method can help solve hard continuous control problems, even with only sparse rewards. We ablate essential design choices and provide additional experiments in section 3.4.3.

3.4.1 Illustrative Experiments

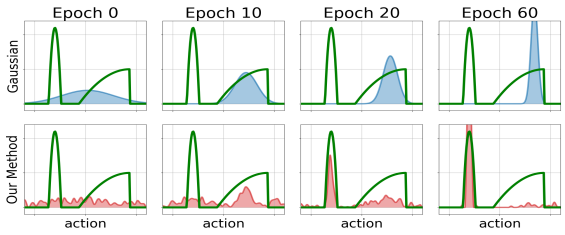


Figure 3.4. Illustrative experiment on continuous bandit

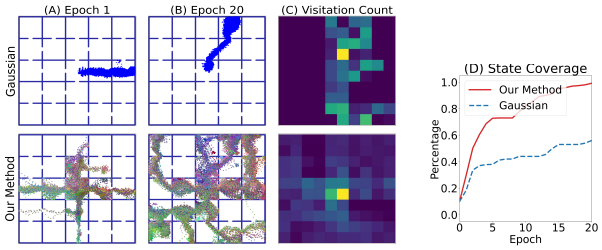


Figure 3.5. Illustrative experiment on 2D maze navigation problem

Can multimodal policies help escape local optima?

We study the effects of our method on a 1D bandit problem as shown in Figure 3.4. It has a 1d action space and a non-convex reward landscape with an additional discontinuous point.

Figure 3.4 compares the performance of our method with a single modality Gaussian policy optimized by REINFORCE. *Notice that we do not add the intrinsic reward for dense reward maximization tasks*. The Gaussian policy, initialized at 0 with a large standard deviation, can cover the whole solution space. However, the gradient w.r.t μ is positive, which means the action probability density will be pushed towards the right, as the expected return on the right side is larger than the left side, although the left side contains a higher extreme value. As a result, the policy will move right and get stuck at the local optimum with a low chance of jumping out. In contrast, under the entropy maximization formulation, our method maximizes the reward while seeking to increase diversity, providing more chances for the policy to explore

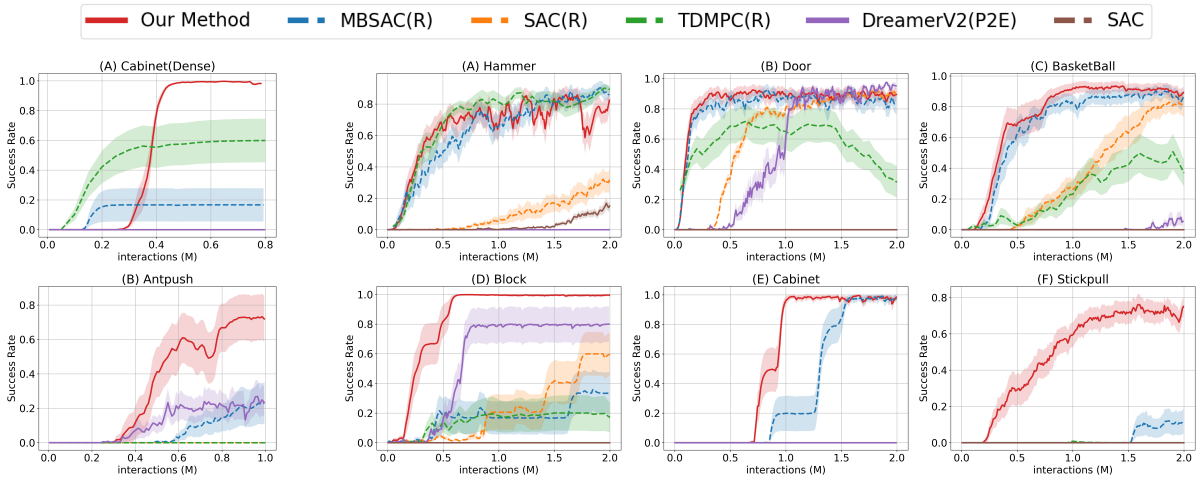


Figure 3.6. Results on dense reward tasks with local optima (exploration disabled)

Figure 3.7. Results on sparse reward tasks

the whole solution space. Furthermore, by turning the latent variables into action distribution, our method can build a multimodal policy distribution that fits the multimodal rewards, explore both modalities simultaneously, and eventually stabilize at the global optimum. This experiment suggests that a multimodal policy is necessary for reward maximization, and our method can help the policy better handle local optima.

Can multimodal policies accelerate exploration?

We argue that maintaining a multimodal policy is beneficial even in the existence of an intrinsic reward to guide the exploration. We illustrate it in a 2D maze navigation task shown in Figure 3.5. The maze consists of 5×5 grids. Each of them is connected with neighbors with a narrow passage. The agent starts in the center grid and can move in four directions. The action space is its position change in two directions $(\Delta x, \Delta y)$.

We apply RPG and single-modality model-based SAC [74] on this environment to maximize the intrinsic reward described in Section 3.3.3. We count the areas covered by the two policies during exploration with respect to the number of samples in Figure 3.5(D). The curve suggests that our method explores the domain much faster, quickly reaching most grids, while

the Gaussian agent only covers the right part of the maze within a limited sample budget.

To understand their differences, we visualize states sampled at different training steps of the two policies in Figure 3.5 (A-B). Our policy below quickly finds four directions to move and gradually expands the state distribution until it fully occupies all grids. Figure 3.5(C) shows the historic state visitation count. It is easy to see that our multimodal policy induces a more uniform distribution over the whole state space, generating a higher state distribution entropy. The optimization procedure of single-modality policy, as shown in the first row of Figure 3.5, suffers from its policy parameterization. It can only explore one modality every time and has to switch modalities one by one, where modalities refer to different regions of the state space. It is hard to predict when it switches modality, making algorithms behave vastly differently in different environments with different random seeds. Sometimes it moves slowly from one direction to another because it has to wait for samples for density estimators to generate enough momentum. As a result, it never explores the left side in Figure 3.5(C). While sometimes, it switches too fast due to the fast updates of the network and does not exploit some modalities enough, missing far-end grids of certain directions that it has explored once. This also causes issues when maximizing external rewards. Even if a single-modal policy finds the optimal solution, it may switch to another modality to continue exploration and it is hard to guarantee that it would come back in the end. In contrast, our method is more like Monte-Carlo sampling, which samples all candidates while converging to solutions of high rewards with high probability.

3.4.2 Continuous Control Problems

We now verify if our method can scale up and help solve challenging continuous control problems. We take 8 representative environments from standard RL benchmarks, including 2 table-top environments from MetaWorld [269], 2 dexterous hand manipulation tasks from [199], 1 navigation problems from [176], and 2 articulated object manipulation from ManiSkill [173]. We show environment examples and provide a detailed environment description in Section 3.5.2. Only **Cabinet (Dense)** and **AntPush** contain dense rewards that lead to local optima. *The*

remaining 6 *environments all only provide sparse rewards*, which means the agents receive a reward 1 when it succeeds to finish the task and 0 otherwise. This change dramatically increases the difficulty of these environments and disastrously hurts the performance of classical RL methods like SAC [74] and PPO [216].

We evaluate our methods against the following baselines: DreamerV2 + Plan2Explore [217], abbreviated as **DreamerV2 (P2E)**, a model-based exploration method based on the disagreement of learned models' prediction. We also consider 3 baselines, **TDMPC**, **MBSAC**, and **SAC** using the same intrinsic rewards as ours. The suffix **(R)** means that when we apply these methods to a sparse-reward environment, we will add RND intrinsic rewards that are the same as in our method. For all results evaluated on dense-reward environments in Figure 3.6, the exploration method of the corresponding algorithm is disabled. The standard SAC without intrinsic rewards validates the difficulty of our tasks. Details of the baseline implementations are in Section 3.5.3.

Figure 3.6 and 3.7 plots the learning progress of each algorithm in all environments (x-axis: number of environment interaction steps in million, y-axis: task success rate). For all environments, we run each algorithm for at least five trials. The curve and the shaded region shows the average and the standard deviation of performance over trials. ***MBSAC*** *shares almost the same implementation as our method, except that it does not condition its policy on latent variables.*.

We first observe that, for dense reward tasks, our method largely improves the success rate on tasks with local optima (Figure 3.6). We can see that in both **AntPush** and **Cabinet (Dense)** tasks, our method outperforms all baselines. Our method consistently finds solutions, regardless of the local optima in the environments. For example, in the task of opening the cabinets' two doors and going to the two sides of the block, our method usually explores the two directions simultaneously and converges at the global optima. In contrast, other methods' performance highly depends on their initialization. If the algorithm starts by opening the wrong doors or pushing the block in the wrong direction, it will not escape from the local minimums;

thus, its success rates are low.

Our methods successfully solve the 6 sparse reward tasks as shown in Figure 3.7. Especially, it consistently outperforms the **MBSAC(R)** baseline, which is a method that only differs from ours by the existence of latent variables to parameterize the policy. Our method reliably discovers solutions in environments that are extremely challenging for other methods (e.g., the **StickPull** environment), clearly demonstrating the advantages of our method in exploration. Notably, we find that **MBSAC(R)**, which is equipped with our object-centric RND, is a strong baseline that can solve **AdroitHammer** and **AdroitDoor** faster than **DreamerV2(P2E)**, proving the effectiveness of our intrinsic reward design. **TDMPC(R)** has a comparable performance with **MBSAC(R)** on several environments. We validate that it has a faster exploration speed in Adroit Environments thanks to latent planning. We find that the **Dreamer(P2E)** does not perform well except for the **BlockPush** environment without the object prior and is unable to explore the state space well.

Visualization of the Multimodal Exploration

Here we visualize modalities explored by our method. We plot the trajectory of the agent in *AntPush* environment, evaluated at different numbers of training stages in Figure 3.8. The agent learned to move forward and explored all directions that would decrease the l_2 distance. It found the left side was easier for moving up in the beginning, but at episode 360, it learned to explore all directions. Ultimately, it explored the left path to the upper room and converged on it.

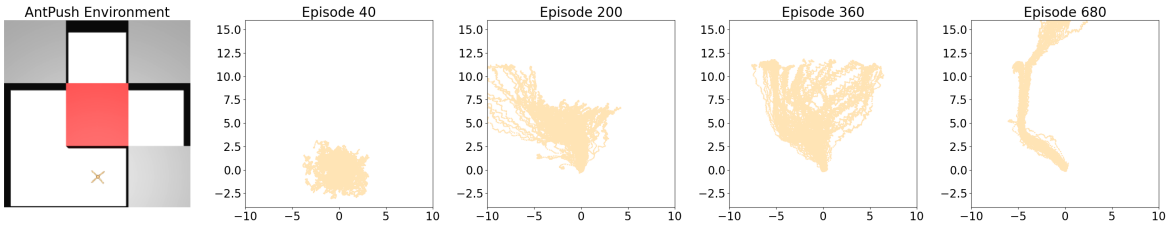


Figure 3.8. Exploration of AntPush, which has the dense reward to guide the agent to move forward.

We also plot the sampled states during exploration for Block, Cabinet, and Stickpull Envs

in Figure 3.9.

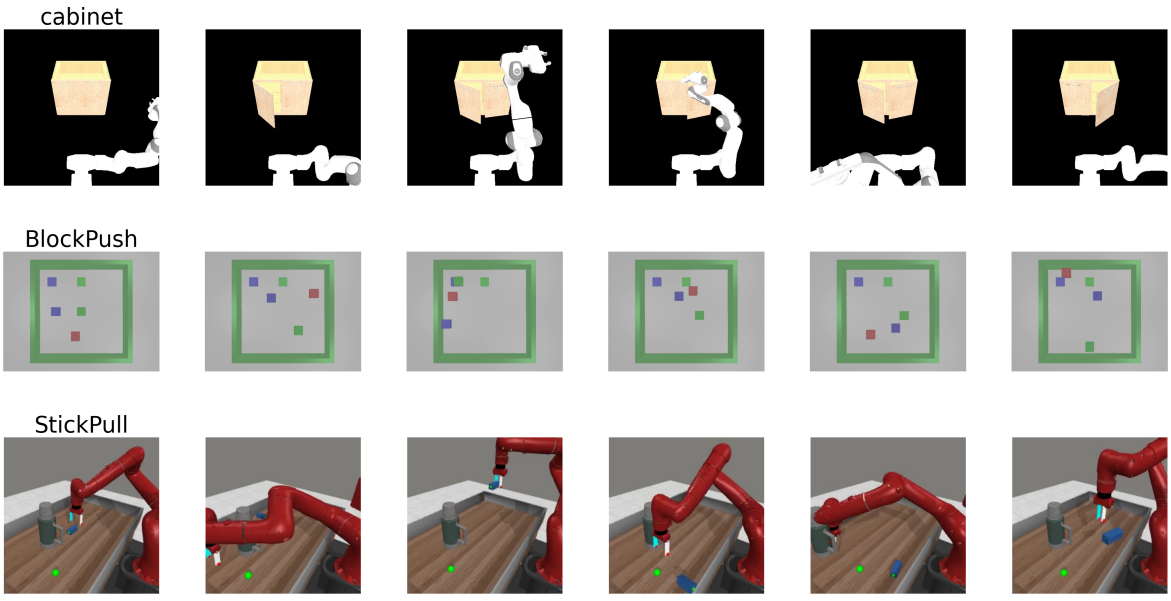


Figure 3.9. Exploration on several environments; The first column shows the initial state. The right 5 figures of the same row plot states sampled from a single agent.

3.4.3 Additional Experiments

Ablation study We analyze various factors influencing the performance of our method in the Maze navigation task in Section 3.4.1. More detailed discussion and experiment results are in Section 3.4.4. Experimental comparisons between different latent spaces show that a Gaussian distribution of dimension 12 outperforms the categorical latent space, both surpassing a baseline that does not use latent variables. A moderate latent space size ≥ 6 is found to be sufficient, with performance declining if the latent dimensions are too small. In terms of reward maximization, the weight of the cross-entropy term (β) is crucial, with results indicating an ideal range between 0.001 and 0.01 for the RND design. Furthermore, the performance from RND is tied to maintaining a large replay buffer and using positional embedding, with a lack of either resulting in degraded exploration. A comparative analysis of policy parameterization methods shows the superiority of the vanilla Gaussian policy over the Gaussian Mixture Models (GMM) and CEM-based policy. The latter two display several optimization issues; GMM struggles with

log-likelihood maximization, and CEM, despite its proficiency at finding local optima, tends to sacrifice its explorative capabilities. Finally, normalizing flow showed initial promise but soon encountered numerical instabilities, highlighting the need for further investigation.

Evaluation on locomotion environments We modified the HalfCheetah-v3 environment in OpenAI Gym [18] to study the performance of our methods in locomotion tasks, shown in Figure 3.10. The cheetah robot moves backward for a certain distance to receive a sparse reward of 1 to succeed. Our exploration method was able to effectively aid the exploration of the Cheetah robot and solve the task easily while removing the exploration term that led to the agent getting stuck. However, in this particular task, modeling multi-modal exploration did not increase the sampling efficiency, as there were only two modalities (moving forward and backward), and model-based SAC could exploit the two modes one by -one and solve the task. This made the advantage of our method negligible in this case. We also evaluated our method compared to SAC [74] on the standard Mujoco environments. Results are shown in Figure 3.11.

Vision-based RL As a proof of concept, we illustrate, in Figure 3.12, the potential of our method for image observations in a single-block pushing environment: the observation consists of two consecutive 64x64 RGB images; the agent needs to control the red block to push the purple box into the target region. We use 4-layer convolutional networks as the encoder for both the policy network and RND estimator. We compare our method with model-based SAC (RND), which has an intrinsic reward to guide exploration but only models single modality policies, and model-based SAC without RND. The result validates our method's effectiveness.

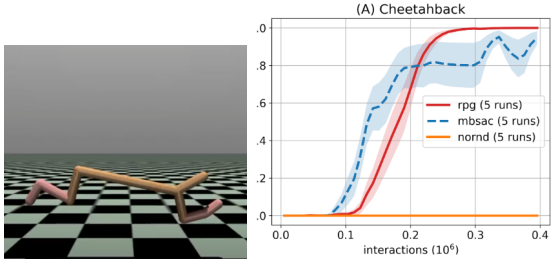


Figure 3.10. Cheetah Back Task (left), success rate (right)

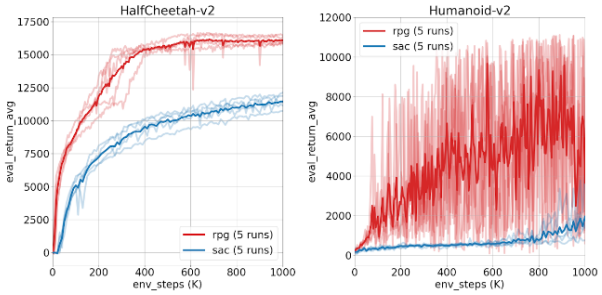


Figure 3.11. Results on Mujoco-v2 Environments

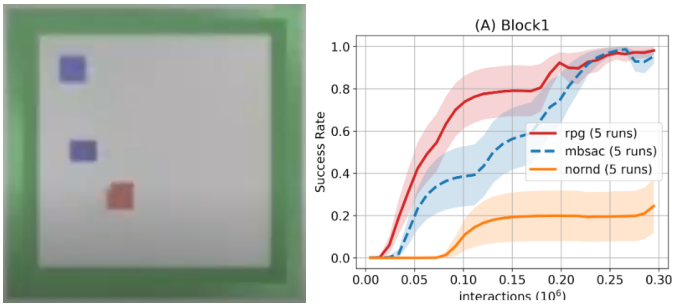


Figure 3.12. Visual Block Push Task (left), success rate (right)

3.4.4 Ablation Study

We study and compare various factors in our methods in Figure 3.13 on the Maze navigation task described in Section 3.4.1. Figure 3.13(A) compares different latent spaces to use. The continuous latent space modeled by a Gaussian distribution of dimension 12 outperforms the categorical latent space, while both are better than the one without latent variables, i.e., the **MBSAC** baselines. Figure 3.13(B) shows the effects of our method when using a Gaussian distribution as the latent space with different β values. The β controls the scale of the cross entropy term $\log p_\phi(z|s,a)$ in reward maximization, as mentioned in Section 3.3.3. The policy

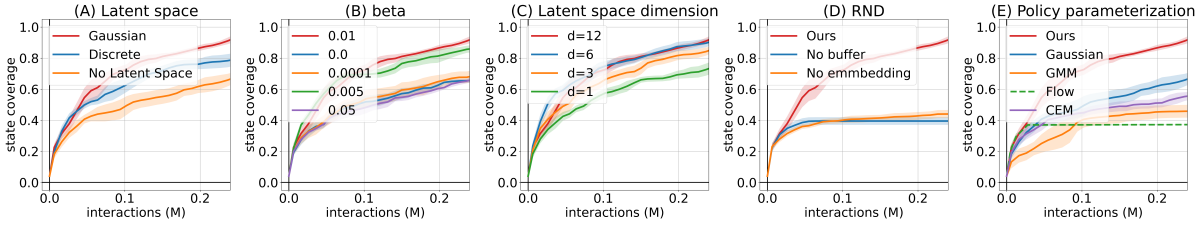


Figure 3.13. Comparing different factors in our methods.

will ignore the latent variable if the β is too small, e.g., $0., 1e-4$. But if the β is too large, though the policy generates diverse solutions, it may explore too much without exploiting past experience. This β plays a similar role as β in $\beta-$VAE [84]. In experiments, we find that β from 0.001 to 0.01 works well in the case of our RND design. Figure 3.13(C) shows the effects of the latent dimensions. For tasks like $2D$ maze, a moderate latent space size $d \geq 6$ is sufficient. But the performance will degrade when it is too small. Figure 3.13(D) ablates our design for the RND. When the RND estimator does not maintain a large replay buffer or does not use the positional embedding, the exploration will suffer a lot. We further compare various policy parameterization methods in Figure 3.13(E). We find that in our implementation, Gaussian mixture models (GMM) and CEM-based policy do not perform as well as the vanilla Gaussian policy. GMM may have trouble in log-likelihood maximization. We noticed several numerical issues in optimizing GMM and Flow when we applied them with RND in sparse reward tasks. Specifically, we have encountered some instability when optimizing the log prob for GMM due to its non-convex nature and the need for sampling to estimate entropy. Similarly, our experiments with Flow have revealed significant parameter divergence and instabilities, warranting further investigation to pinpoint the root cause. CEM has a stronger ability to find local optima and generates actions with less randomness, which may sacrifice its ability to do exploration. Besides, we find the policy parameterized by a normalizing flow distribution behaves well initially but soon meets numerical instabilities and fails to proceed with optimization, suggesting more investigations are needed in this direction.

3.5 Additional Details

3.5.1 Algorithm

Network architecture

We use the following two-layer MLP to model policy π_θ, value Q_ψ, state encoder f_ψ, and the encoder $p_\phi(z|s)$. The network structures are shown in the pytorch's convention [186].

```
Sequential(
  (0): Linear(in_features=inp_dim, out_features=256, bias=True)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=256, out_features=256, bias=True)
  (3): ELU(alpha=1.0)
  (4): Linear(in_features=256, out_features=out_dim, bias=True)
)
```

The dynamics network is a single-layer GRU with a hidden dimension 256. The RND network g_θ we use is a 3 layer MLP network with hidden dimension 512 and leaky ReLU as its activation function.

We maintain target networks like the standard double Q learning. The hyperparameters for training the network are listed in Table 3.1.

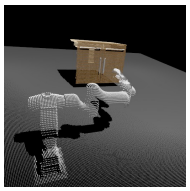
Algorithm 1. Model-based Reparameterized Policy Gradient

Input: $p_\phi, \pi_\theta, h_\psi, R_\psi, f_\psi, Q_\psi$ and an optional density estimator g_θ
Initialize p_ϕ, π_θ, construct the replay buffer $\mathscr{B}$.
while time remains **do**
Sample start state o_1 and encode it as $s_1 = f_\psi(o_1)$. Select z from $\pi_\theta(z|s_1)$.
Execute the policy $\pi_\theta(a|s,z)$ and store transitions into the replay buffer $\mathscr{B}$.
Sample a batch of trajectory segment of length K $\{\tau^i_{t:t+K}, z\}$ from the buffer $\mathscr{B}$.
Optional: update and estimate the density estimator g_θ and relabel transitions with the negative density as the intrinsic reward.
Optimize ψ using Equation 3.4.
Optimize $\pi_\theta(a|s,z)$ with gradient descent to maximize the value estimate in Equation 3.3 for s, z sampled from the buffer.
Optimize $\pi_\theta(z|s_1)$ with policy gradient to maximize $V_{\text{estimate}}(s_1, z) - \alpha \log \pi_\theta(z|s_1)$ for s_1 sampled from the buffer.
Optimize α, β if necessary .
end while

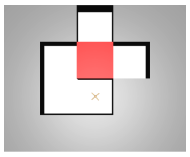
Table 3.1. RPG hyperparameters. We here list the hyper-parameters used in the experiments. The hyper-parameters keep the same for our **MBSAC** baseline except that **MBSAC** has no latent space. Notice that for dense reward tasks, the entropy of $\pi_\theta(z|s_1)$ is linearly decayed starting from 3×10^5 environment steps to $1M$ steps to ensure optimality.

Hyperparameter	Value
Discount factor (γ)	0.99
Seed step	1000
Replay buffer size	800000
Model rollout horizon (H)	3
Action distribution	Tanh Normal
Entropy target	$-\lvert\mathscr{A}\rvert$
Initial entropy coefficient α	0.01
Cross-entropy coefficient β	0.005
RND coefficient β	0.1
Environment steps per gradient update	5
Temperature	$\mathscr{T}$
Learning rate	3×10^{-4}
Batch size	512
Target network update ratio	0.005
Actor update freq	2
State embedding dimension	100
grad norm clip	1.0
Positional encoding dimension	6
Latent distribution $\mathscr{Z}$	Normal
$\mathscr{Z}$ dimension	12
$p_\phi(z\|s,a)$ distribution	Normal distribution with std 0.38
$\pi_\theta(z\|s_1)$	$\mathscr{N}(0,1)$ for sparse reward tasks

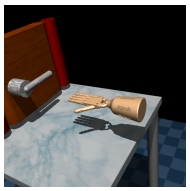
3.5.2 Environment



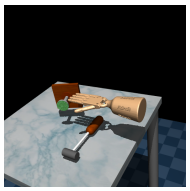
Cabinet (Dense) [70]. The agent controls the movement of a 12 dof mobile robot arm and gripper robot to open both cabinet doors. The agent receives a dense reward for reaching its nearest door's handle. Besides, it receives a higher reward when it opens the right door than the left door. The agent succeeds when it fully opens the right door while the dense reward will typically drive the agent close to the handle of the left door. The episode length is 60.



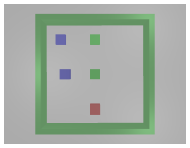
AntPush [176]. The agent controls an ant robot with action dimension 8 to go to the upper room. The reward is the l_2 distance between the agent and a point in the upper room. The optimal path is to go to the left of the red block and push it to the right and go to the upper room. However, agents often get stuck at the local optima, which pushes the block forward or moves to go to the right side. The episode length is 400.



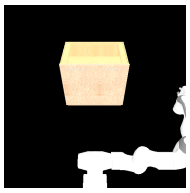
Door [199]. The agent controls a dexterous hand with action dimension 26 to open a door. The agent only receives a reward of 1 when it successfully undoes the latch and opens the door. The episode length is 100 with an action repeat 2. Objects of interest include the hand's palm, the latch, and the door.



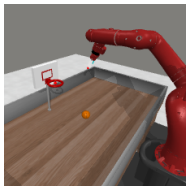
Hammer [199]. The agent controls a dexterous hand with action dimension 26 to force drive a nail into the board. The agent only receives a reward of 1 when it has driven the nail all the way in. Action repeat is 2. The episode length is 125. We encode the position of the hand's palm, the hammer, and the nail.



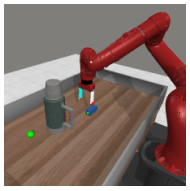
BlockPush [262]. The agent controls the movement of the red block with action dimension 2 to push the green block (middle) to the green destination (above) and the blue block (middle) to the blue destination (above). The agent only receives a reward of 1 when it has successfully pushed both blocks to the exact destination with a small tolerance. The objects of interest contain the location of the three blocks. The environment horizon is 60.



Cabinet (Sparse) [70]. The agent controls the movement of a 9 dof robot arm and gripper robot to open both doors of the cabinet. The agent only receives a reward of 1 when both cabinet doors are fully opened. We encode the position of the robot's end effector and the location of the cabinet's door. Its episode length is 60.



Meta-World BaseketBall [269]. The agent controls the movement of a gripper with a 4 dof controller to move the ball into the basket. The agent only receives a reward of 1 when the ball is sufficiently close to the basket. The locations of the ball and the location of robots' fingertips are what we are concerned about. The episode length is 100, including 2 action repeats.



Meta-World StickPull [269]. The agent controls the movement of a gripper with a 4 dof controller to pull the container with a blue stick. The agent receives a reward of 1 only when the stick is inserted inside the handle, and the container is already pulled sufficiently close to the green dot. We encode the positions of the fingertips, the stick, and the handle of the cup for computing intrinsic rewards. The remaining setup is the same as **BasketBall**.

3.5.3 Baseline

TDMPC [78], we used the publically available official implementation and default hyperparameters provided by the authors at https://github.com/nicklashansen/tdmpc.

SAC [74], we implemented according to the original paper and used the default hyperparameter provided by the authors.

We use the abbreviation **TDMPC(R)**, **SAC(R)** to represent that we add an intrinsic reward with scale 0.1 for exploration in environments with only sparse rewards.

DreamerV2 [77], we used the publically available official implementation and default hyperparameters provided by the authors at https://github.com/danijar/dreamerv2.

Plan2Explore [217], we run DreamerV2 according to the instructions provided by

https://github.com/ramanans1/plan2explore with hyperparameters provided by the authors of the paper.

For all baseline algorithms, we only change model update frequency to once every 5 environment steps.

3.6 Connection with Other Generative Models

Our method is based on the same variational bound shared with many other generative models

$$\log p(x) = E_{z\sim q(z)}\left[\log p(x,z) - \log q(z)\right] + KL(q(z)\|p(z|x)).$$

By different choices of latent space, posterior $q(z|x)$, joint distribution $p(x,z)$, we can obtain different generative models. For example, VAE models $p_\theta(x,z) = p_\theta(x|z)p(z)$ and $q(z) = q_\phi(z|x)$ using neural networks and then optimize θ,ϕ jointly to maximize the ELBO bound. By doing so, $q_\phi(z|x)$ will align with the true posterior of $p_\theta(z|x)$. Thus

$$\log p(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z) + \log p(z) - \log q_\phi(z|x)]$$

The Expectation–maximization algorithm (EM) [40] for learning Gaussian mixture models assumes that we have $p_\theta(x,z) = p_\theta(x|z)p_\theta(z)$ where z is a categorical representation.

- E-step: finding $q_\phi(z|x)$ by solving $\max_\phi \log p_\theta(x) - D_{KL}(q_\phi(z|x)||p_\theta(z|x))$ where

$$p_\theta(z|x) = p_\theta(x,z)/\int p_\theta(x,z)dz.$$

- M-step: fixing ϕ, find $\max_\theta E_{q_\phi}[\log p_\theta(x,z)] - E_{q_\phi}[\log q_\phi(z|x)]$ which is exactly maximizing the ELBO.

In Maximum Entropy RL [124], we have optimality $p(O,\tau) = p(O|\tau)p(\tau)$ defined by the reward, and we optimize $\pi_\theta(\tau|O)$ only. The ELBO bound becomes a maximum entropy term

$E_{\tau\sim\pi}\left[\log p(O|\tau)+\log p(\tau)-\log \pi(\tau)\right]$. Our method differs from it by introducing an additional variable z. Table 3.2 compares various generative models.

Table 3.2. Comparison of different algorithms that optimize ELBO bounds for inference

	Latent	Encoder $q(z\|x)$	Joint $p(x,z)$	MLE objective
VAE	z	$p_\phi(z\|x)$	$p_\theta(x\|z)p(z)$	$p(x)$
EM	z	$\max_\phi \log p_\theta(x) - D_{KL}(q_\phi(z\|x)\|\|p_\theta(z\|x))$	$p_\theta(x\|z)p_\theta(z)$	$p(x)$
Diffusion	$\{x_t\}_{t\geq 1}$	$\prod_{i=1}^{T}\mathcal{N}(x_t;\sqrt{1-\beta_t}x_{t-1},\beta_t\mathbf{I})$	$p(x_T)\prod_{t\geq 1}p_\theta(x_{t-1}\|x_t)$	$p(x_0)$
MaxEntRL	τ	$\pi_\theta(\tau)$	$p(O\|\tau)p(\tau)$	$p(O)$
RPG	τ,z	$\pi_\theta(z,\tau)$	$p(O\|\tau)p_\phi(z\|\tau)p(\tau)$	$p(O)$

3.7 Limitation and Future Work

Our approach capitalizes on the advantages offered by multiple components, effectively addressing complex exploration issues in continuous spaces. However, it also introduces certain hurdles and constraints. For instance, our intrinsic reward is predicated on assumptions regarding the recognition of objects and their spatial positioning. This approach may be unsuitable in environments with unidentified objects or where observations don't plainly reveal object-related information, akin to scenarios in vision-based RL; Learning the world model typically results in a slower pace of gradient updates; Incorporating a cross-entropy network adds an extra layer of complexity to the network design and training. Therefore, it is worth discussing potential future directions that might address these limitations.

Object-centric learning for vision-based RL While the Random Network Distillation (RND) is initially tailored for image observations, integrating object-centric design to accelerate exploration in vision-based RL will be an interesting direction. This suggests two typical strategies to apply our method to tasks with vision observations: (1) The first involves directly encoding observations without considering object information. It proves effective in scenarios with no occlusion and a static background, wherein objects emerge as the sole salient feature of the input. We provide a proof-of-concept experiment in Section 3.4.3. (2) The second approach harnesses computer vision techniques to identify objects for object-centric exploration. This

includes applying recent large-scale vision foundation models, which possess zero-shot object detection capabilities as outlined in [272] or leveraging slot-attention for object discovery as described in [150].

Combining with previous model-based control and planning methods Instead of learning the world model from on-policy data, we can pre-train a physical world model [138] or use analytical models [192, 96] to gain generalizability and efficiency. Moreover, we drew inspiration from RRT-like motion planners [109] to derive our policy to sample over the configuration space and bias the exploration towards significant kinematics changes. Thus, an exciting direction is incorporating structures in model-based control into RL algorithms, including temporal structures like dynamics motion primitives [231] and semantic information from TAMP [62].

Extending to other probabilistic models Our method can be viewed as variational inference [201] over a particular stochastic computation graph [254]. The computation graph contains hidden variables, and we use the Bellman equation and a learned model to estimate its gradient. This provides a new perspective that bridges online Reinforcement Learning (RL) with generative models and sequence modeling. In the future, we are interested in exploring how sequence-modeling techniques, such as transformers and hierarchical methods, can be used to model the policy in our framework.

3.8 Conclusion

We derive a framework that models the policy of continuous RL by a multimodal distribution in the variational inference framework. The method reparameterizes latent variables into trajectories like generative models. Under this framework, we learn a world model to help learn multimodal policy data efficiently. Incorporating an object-centric intrinsic reward, our method can solve challenging continuous control problems with little to no reward signal.

Acknowledgements

Chapter 3, in full, is a reprint of the material published in International Conference on Machine Learning 2023. Reparameterized Policy Learning for Multimodal Trajectory Optimization; Huang, Zhiao; Liang, Litian; Ling, Zhan; Li, Xuanlin; Gan, Chuang; Su, Hao. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Mapping State Space using Landmarks for Universal Goal Reaching

An agent that has well understood the environment should be able to apply its skills for any given goals, leading to the fundamental problem of learning the Universal Value Function Approximator (UVFA). A UVFA learns to predict the cumulative rewards between all state-goal pairs. However, empirically, the value function for long-range goals is always hard to estimate and may consequently result in failed policy. This has presented challenges to the learning process and the capability of neural networks. We propose a method to address this issue in large MDPs with sparse rewards, in which exploration and routing across remote states are both extremely challenging. Our method explicitly models the environment in a hierarchical manner, with a high-level dynamic landmark-based map abstracting the visited state space, and a low-level value network to derive precise local decisions. We use the farthest point sampling to select landmark states from past experience, which has improved exploration compared with simple uniform sampling. Experimentally we showed that our method enables the agent to reach long-range goals at the early training stage, and achieve better performance than previous RL algorithms in a number of challenging tasks.

4.1 Introduction

Reinforcement learning (RL) allows training agents for planning and control tasks by feedbacks from the environment. While significant progress has been made in the standard setting of achieving a goal known at training time, e.g., to reach a given flag as in MountainCar [170], very limited efforts have been exerted on the setting when goals at evaluation are unknown at training time. For example, when a robot walks in an environment, the destination may vary from time to time. Tasks of this kind are unanimous and of crucial importance in practice. We call them Universal Markov Decision Process (UMDP) problems following the convention of [126].

Pioneer work handles UMDP problems by learning a *Universal Value Function Approximator* (UVFA). In particular, Schaul et al. [211] proposed to approximate a goal-conditioned value function $V(s,g)$[1] by a multi-layer perceptron (MLP), and Andrychowicz et al. [8] proposed a framework called *hindsight experience replay* (HER) to smartly reuse past experience to fit the universal value function by TD-loss. However, for complicated policies of long-term horizon, the UVFA learned by networks is often not good enough. This is because UVFA has to memorize the cumulative reward between all the state-goal pairs, which is a daunting job. In fact, the cardinality of state-goal pairs grows by a high-order polynomial over the horizon of goals.

While the general UMDP problem is extremely difficult, we consider a family of UMDP problems whose state space is a low-dimension manifold in the ambient space. Most control problems are of this type and geometric control theory has been developed in the literature [22]. Our approach is inspired by manifold learning, e.g., Landmark MDS [37]. We abstract the state space as a small-scale map, whose nodes are landmark states selected from the experience replay buffer, and edges connect nearby nodes with weights extracted from the learned local UVFA. A network is still used to fit the local UVFA accurately. The map allows us to run high-level planning using pairwise shortest path algorithm, and the local UVFA network allows us to derive

[1] s is the current state and g is the goal.

an accurate local decision. For a long-term goal, we first use the local UVFA network to direct to a nearby landmark, then route among landmarks using the map towards the goal, and finally reach the goal from the last landmark using the local UVFA network.

Our method has improved sample efficiency over purely network learned UVFA. There are three main reasons. First, the UVFA estimator in our framework only needs to work well for local value estimation. The network does not need to remember for faraway goals, thus the load is alleviated. Second, for long-range state-goal pairs, the map allows propagating accurate local value estimations in a way that neural networks cannot achieve. Consider the extreme case of having a long-range state-goal pair never experienced before. A network can only guess the value by extrapolation, which is known to be unreliable. Our map, however, can reasonably approximate the value as long as there is a path through landmarks to connect them. Lastly, the map provides a strong exploration ability and can help to obtain rewards significantly earlier, especially in the sparse reward setting. This is because we choose the landmarks from the replay buffer using a farthest-point sampling strategy, which tends to select states that are closer to the boundary of the visited space. In experiments, we compared our methods on several challenging environments and have outperformed baselines.

Our contributions are: First, We propose a sample-based method to map the visited state space using landmarks. Such a graph-like map is a powerful representation of the environment, maintains both local connectivity and global topology. Second, our framework will simultaneously map the visited state space and execute the planning strategy, with the help of a locally accurate value function approximator and the landmark-based map. It is a simple but effective way to improve the estimation accuracy of long-range value functions and induces a successful policy at the early stage of training.

4.2 Related work

Variants of goal-conditioned decision-making problems have been studied in literature [236, 156, 211, 191]. We focus on the goal-reaching task, where the goal is a subset of the state space. The agent receives meaningful rewards if and only if it has reached the goal, which brings significant challenges to existing RL algorithms. A significant recent approach along the line is Hindsight Experience Replay (HER) by Andrychowicz et al [8]. They proposed to relabel the reached states as goals to improve data efficiency. However, they used only a single neural network to represent the Q value, learned by DDPG [142]. This makes it hard to model the long-range distance. Our method overcomes the issue by using a sample-based map to represent the global structure of the environment. The map allows to propagate rewards to distant states more efficiently. It also allows to factorize the decision-making for long action sequences into a high-level planning problem and a low-level control problem.

Model-based reinforcement learning algorithms usually need to learn a local forward model of the environment, and then solve the multi-step planning problem with the learned model [76, 180, 225, 83, 229, 270]. These methods rely on learning an accurate local model and require extra efforts to generalize to the long term horizon [111]. In comparison, we learn a model of environment in a hierarchical manner, by a network-based local model and a graph-based global model (map). Different from previous works to fit forward dynamics in local models, our local model distills local cumulative rewards from environment dynamics. In addition, our global model, as a small graph-based map that abstracts the large state space, supports reward propagation at long range. One can compare our framework with Value Iteration Networks (VIN) [238]. VIN focused on the 2D navigation problem. Given a predefined map of known nodes, edges, and weights, it runs the value iteration algorithm by ingeniously simulating the process through a convolutional neural network [123]. In contrast, we construct the map based upon the learned local model.

Sample-Based Motion Planning (SBMP) has been widely studied in the robotics con-

text [79, 121, 110]. The traditional motion planning algorithm requires the knowledge of the model. Recent work has combined deep learning and deep reinforcement learning for [101, 197, 115, 53]. In particularly, PRM-RL addressed the 2D navigation problem by combining a high-level shortest path-based planner and a low-level RL algorithm. To connect nearby landmarks, it leveraged a physical engine, which depends on sophisticated domain knowledge and limits its usage to other general RL tasks. In the general RL context, our work shows that one can combine a high-level planner and a learned local model to solve RL problems more efficiently. Some recent work also utilize the graph structure to perform planning [209, 271], however, unlike our approach that discovers the graph structure in the process of achieving goals, both [209, 271] require supervised learning to build the graph. Specifically, [209] need to learn a Siamese network to judge if two states are connected, and [271] need to learn the state-attribute mapping from human annotation.

Our method is also related to hierarchical RL research [126, 118, 176]. The sampled landmark points can be considered as sub-goals. [126, 176] also used HER-like relabeling technique to make the training more efficient. These work attack more general RL problems without assuming much problem structure. Our work differs from previous work in how high-level policy is achieved. In their methods, the agent has to learn the high-level policy as another RL problem. In contrast, we exploit the structure of our universal goal reaching problem and find the high-level policy by solving a pairwise shortest path problem in a small-scale graph, thus more data-efficient.

4.3 Background

Universal Markov Decision Process (UMDP) extends an MDP with a set of goals $\mathscr{G}$. UMDP has reward function $\mathscr{R} : \mathscr{S} \times \mathscr{A} \times \mathscr{G} \to \mathscr{R}$, where $\mathscr{S}$ is the state space and $\mathscr{A}$ is the action space. Every episode starts with a goal selected from $\mathscr{G}$ by the environment and is fixed for the whole episode. We aim to find a goal conditioned policy $\pi : \mathscr{S} \times \mathscr{G} \to \mathscr{A}$

to maximize the expected cumulative future return $V_{g,\pi}(s_0) = E_\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, g)]$, which called goal-conditioned value, or universal value. Universal Value Function Approximators (UVFA) [211] use neural network to model $V(s,g) \approx V_{g,\pi^*}(s)$ where π^* is the optimal policy, and apply Bellman equation to train it in a bootstrapping way. Usually, the reward in UMDP is sparse to train the network. For a given goal, the agent can receive non-trivial rewards only when it can reach the goal. This brings a challenge to the learning process.

Hindsight Experience Replay (HER) [8] proposes goal-relabeling to train UVFA in sparse reward setting. The key insight of HER is to "turn failure to success", i.e., to make a failed trajectory become success, by replacing the original failed goals with the goals it has achieved. This strategy gives more feedback to the agent and improves the data efficiency for sparse reward environments. Our framework relies on HER to train an accurate low-level policy.

4.4 Universal Goal Reaching

Problem Definition:

Our universal goal reaching problem refers to a family of UMDP tasks. The state space of our UDMP is a low-dimension manifold in the ambient space. Many useful planning problems in practice are of this kind. Example universal goal reaching environments include labyrinth walking (e.g., AntMaze [46]) and robot arm control (e.g., FetchReach [190]). Their states can only transit in a neighborhood of low-dimensionality constrained by the degree of freedom of actions.

Following the notions in Sec 4.3, we assume that a goal g in goal space $\mathscr{G}$ which is a subset of the state space $\mathscr{S}$. For example, in a labyrinth walking game with continuous locomotion, the goal can be to reach a specific location in the maze at any velocity. Then, if the state s is a vector consisting of the location and velocity, a convenient way to represent the goal g would be a vector that only contains the dimensions of location, i.e., the goal space is a projection of the state space.

The universal goal reaching problem has a specific transition probability and reward

structure. At every time step, the agent moves into a local neighborhood based on the metric in the state space, which might be perturbed by random noise. It also receives some negative penalty (usually a constant, e.g., -1 in the experiments) unless it has arrived at the vicinity of the goal. A 0 reward is received if the goal is reached. To maximize the accumulated reward, the agent has to reach the goal in fewest steps. Usually the only non-trivial reward 0 appears rarely, and the universal goal reaching problem falls in the category of *sparse reward* environments, which are hard-exploration problems for RL.

A Graph View:

Assume that a policy π takes at most steps T to move from s to g and the reward at each step r_k's absolute value is bounded by R_{max}. Let $w_\pi(s,t)$ be the expected total reward along the trajectory, and $d_\pi(s,t) = -w_\pi(s,t)$ for all s,t. If $\gamma \approx 1$, we can show that UVFA $V_\pi(s,g)$ can be approximated as (see supplementary for details):

$$V_\pi(s,g) \approx E[w_\pi(s,g)] = E[-d_\pi(s,g)] \tag{4.1}$$

This suggests us to view the MDP as a directed graph, whose nodes are the state set $\mathscr{S}$, and edges are sampled according to the transition probability in the MDP. The general value iteration for RL problems is exactly the shortest path algorithm in terms of $d_\pi(s,g)$ on this directed graph. Besides, because the nodes form a low-dimensional manifold, nodes that are far away in the state space can only be reached by a long path.

The MDP of our universal goal reaching problem is a large-scale directed graph whose nodes are in a low-dimensional manifold. This structure allows us to estimate the all-pair shortest paths accurately by a landmark based coarsening of the graph.

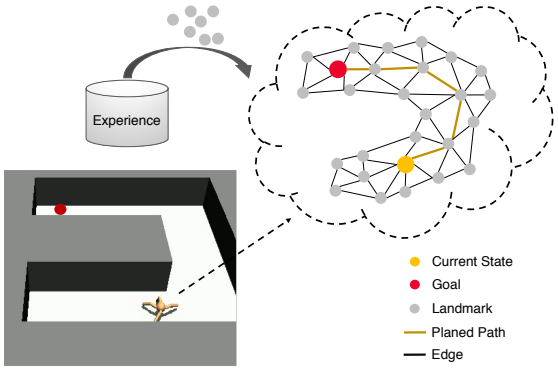


Figure 4.1. An illustration of our framework. The agent is trying to reach the other side of the maze by planning on a landmark-based map. The landmarks are selected from its past experience, and the edges between the landmarks are formed by a UVFA.

4.5 Approach

In this work, we choose deep RL algorithms such as DQN and DDPG for discrete and continuous action space, respectively. UVFA [211] is a goal-conditioned extension of the original DQN, while HER (Sec 4.3), can produce more informative feedback for UVFA learning. Our algorithm is thus based upon HER, and the extension of this approach for other algorithms is also straightforward.

4.5.1 Basic Idea

Our approach aims at addressing the fundamental challenges in UVFA learning. As characterized in the previous section, the UVFA estimation solves a pair-wise shortest path problem, and the underlying graph has a node space of high cardinality. Note that UVFA has to memorize the distance between every state-goal pairs, through trajectory samples from the starting state to the goal, which is much larger than the original state space.

Such large set of state-goal pairs poses the challenge. First, it takes longer time to sample enough state-goal pairs. Particularly, at the early stage, only few state-goal samples have been collected, so learning from them requires heavy extrapolation by networks, which is well known to be unreliable. Second, memorizing all the experiences is too difficult even for large networks.

We propose a map to abstract the visited state space by landmarks and edges to connect them. This abstraction is reasonable due to the underlying structure of our graph — a low-dimensional manifold [67]. We also learn local UVFA networks that only needs to be accurate in the neighborhood of landmarks. As illustrated in Figure 4.1, an ant robot is put in an "U" Maze to reach a given position. It should learn to model the maze as a small-scale map based on its past experiences.

This solution addresses the challenges. For the UVFA network, it only needs to remember experiences in a local neighborhood. Thus, the training procedure requires much lower sample complexity. The map decomposes a long path into piece-wise short ones, and each of which is from an accurate local network.

Our framework contains three components: a value function approximator trained with hindsight experience replay, a map that is supported by sampled landmarks, and a planner that can find the optimal path with the map. We will introduce them in Sec 4.5.2, Sec 4.5.3, and Sec 4.5.4, respectively.

4.5.2 Learning a Local UVFA with HER

Specifically, we define the following reward function for goal reaching problem:

$$r_t = \mathscr{R}(s_t, a_t, g) = \begin{cases} 0 & |s'_t - g| \leq \delta \\ -1 & otherwise \end{cases}$$

Here s'_t is the next observation after taking action a_t. We first learn a UVFA based on HER, which has proven its efficiency for UVFA. In experiments (see Sec 4.6.3), we find out that the agent trained with HER does master the skill to reach goals of increasing difficulty in a curriculum way. However, the agent can seldom reach the most difficult goals constantly, while the success rate of reaching easier goals remains stable. All these observations prove that HER's value and policy is locally reliable.

One can pre-train the HER agent and then build map for planner. However, as an off-policy algorithm, HER can work with arbitrary exploration policy. Thus we use the planner based on current local HER agent as the exploration policy and train the local HER agent jointly. We sample long horizon trajectories with the planner and store them into the replay buffer. We change the replacement strategy in HER, ensuring that the replaced goals are sampled from the near future within a fixed number of steps to increase the agent's ability to reach nearby goals at the early stage.

The UVFA trained in this step will be used in the planner for two purposes: (1) to estimate the distance between two local states belonging to the same landmark, or between two nearby landmarks; and (2) to decide whether two states are close enough so that we can trust the distance estimation from the network. Although the learned UVFA is imperfect globally, it is enough for the two local usages.

4.5.3 Building a Map by Sampling Landmarks

After training the UVFA, we will obtain a distance estimation $d(s,g)$[2], a policy for any state-goal pair (s,g), and a replay buffer that contains all the past experiences. We will build a landmark-based map to abstract the state space based on the experiences. The pseudo-code for the algorithm is shown in Algorithm 2.

Landmark Sampling

The replay buffer stores visited states during training. Instead of localizing few important states that play a key role in connecting the environment, we seek to sample many states to cover the visited state space.

Limited by computation budget, we first uniformly sample a big set of states from the replay buffer, and then use the farthest point sampling (FPS) algorithm [10] to select landmarks to support the explored state space. The metric for FPS can either be the Euclidean distance

[2]If the algorithm returns a Q function, we will calculate the value by selecting the optimal action and calculate the Q function and convert to d by Eq. 4.1

Algorithm 2. Planning with State-space Mapping (Planner)

Input: state obs, goal g, UVFA $Q(s,g,a)$, clip_value τ
Output: Next subgoal g_{next}

Sample transitions $T = (s,a,s')$ from replay buffer B
$V \leftarrow$ **FPS**$(S = \{s \| (s,a,s') \in T\}) \cup \{g\}$ ▷ Farthest point sampling to find landmarks
$W_{ij} \leftarrow \infty$ ▷ Initialize Map as graph $G = \langle V, W \rangle$
for For all pairs of (v_i, v_j) **do**
    $w_{ij} \leftarrow \min_a -Q(v_i, v_j, a)$
    if $w_{ij} \leq \tau$ **then**
        $W_{ij} \leftarrow w_{ij}$
    end if
end for
$D \leftarrow$**Bellman_Ford**(W) ▷ Calculate pairwise distance
$g_{next} \leftarrow \arg\min_{v_i,a} -Q(obs, v_i, a) + D_{v_i,g}$
return g_{next}

between the original state representation or the pairwise value estimated by the agent.

We compare different sampling strategies in Section 4.6.3, and demonstrate the advantage of FPS in abstracting the visited state space and exploration.

Connecting Nearby Landmarks

We first connect landmarks that have a reliable distance estimation from the UVFA and assign the UVFA-estimated distance between them as the weight of the connecting edge.

Since UVFA is accurate locally but unreliable for long-term future, we choose to only connect nearby landmarks. The UVFA is able to return a distance between any pair (s,g), so we connect the pairs with distance below a preset threshold τ, which should ensure that all the edges are reliable, as well as the whole graph is connected.

With these two steps, we have built a directed weighted graph which can approximate the visited state space. This graph is our map to be used for high-level planning. Such map induces a new environment, where the action is to choose to move to another landmark. The details can be found in Algorithm 3.

4.5.4 Planning with the Map

We can now leverage the map and the local UVFA network to estimate the distance between any state-goal pairs, which induces a reliable policy for the agent to reach the goal.

For a given pair of (s,g), we can plan the optimal path between (s,g) by selecting a serial of landmarks $l_1,\cdots,l_k$, so that the approximated distance will be $\bar{d}(s,g) = \min_{l_1,\cdots,l_k} d(s,l_1) + \sum_{i=1}^{k-1} d(l_i,l_{i+1}) + d(l_k,g)$. The policy from s to g can then be approximated as: $\bar{\pi}(s,g) = \pi(s,l_1) + \sum_{i=1}^{k-1} \pi(l_i,l_{i+1}) + \pi(l_k,g)$. Here the summation of π is the concatenation of the corresponding action sequence.

In our implementation, we run the shortest path algorithm to solve the above minimization problem. To speed up the pipeline, we first calculate the pairwise distances $d(l_i,g)$ between each landmark l_i and the goal g when episode starts. When the agent is at state s, we can choose the next subgoal by finding $g_{next} = \arg\min_{l_i} d(s,l_i) + d(l_i,g)$.

4.5.5 Proof of the Approximation

We have proposed to view the MDP as a directed graph. We can prove that this is true when $\gamma \approx 1$.

Assume that a policy π takes at most steps T to move from s to g and the reward at each step r_k's absolute value is bounded by R_{max}. Let $w_\pi(s,t)$ be the expected total reward along the trajectory, and $d_\pi(s,t) = -w_\pi(s,t)$ for all s,t. If $\gamma = 1-\varepsilon$, we can prove that [3]: $|V_\pi(s,g) - w_\pi(s,g)| \leq T^2 R_{max}\varepsilon$. Thus, when $\gamma \approx 1$ and $T^2 R_{max}(1-\gamma) \xrightarrow{+} 0$, UVFA can be approximated as:

$$V_\pi(s,g) \approx E[w_\pi(s,g)] = E[-d_\pi(s,g)]. \tag{4.2}$$

In this case, it is easy to show that the value iteration based on Bellman Equation $V_{\pi^*}(s,g) =$

[3] When $\varepsilon \xrightarrow{+} 0$, we can approximate $(1-\varepsilon)^k$ by its first-order Taylor expansion $1-\varepsilon k$.

$R(s,a,g) + \gamma \mathbb{E}[V_{\pi^*}(s',g)]|_{s' \sim \mathscr{P}_{\pi^*}(\cdot|s,a)}$ implies

$$w_{\pi^*}(s,g) \approx R(s,a,g) + w_{\pi^*}(s',g)|_{s' \sim \mathscr{P}_{\pi^*}(\cdot|s,a)},$$

where $\mathscr{P}_{\pi^*}$ is the transition probability of optimal policy π^*.

To prove this, note that γ is smaller than 1, we can further replace γ with $\varepsilon = 1 - \gamma$. When $\varepsilon \xrightarrow{+} 0$, we can approximate $(1-\varepsilon)^k$ by its first-order Taylor expansion $1 - \varepsilon k$. Thus we have:

$$\begin{aligned}
|V_\pi(s,g) - w_\pi(s,g)| &= |E[\sum_{k=1}^{T} r_k \gamma^{k-1}] - E[\sum_{k=1}^{T} r_k]| \\
&= |E[\sum_{k=1}^{T} r_k (1-\varepsilon)^{k-1}] - E[\sum_{k=1}^{T} r_k]| \\
&\approx |E[\sum_{k=1}^{T} r_k - (k-1)\varepsilon r_k] - E[\sum_{k=1}^{T} r_k]| \\
&= |\sum_{k=1}^{T} (k-1)\varepsilon E[r_k]| \\
&\leq T^2 R_{max} \varepsilon \xrightarrow{+} 0,
\end{aligned}$$

So we finish the proof of the relationship between V_π and w_π when $\gamma \to 1$, as mentioned in the main paper.

4.5.6 Training Algorithm Outline

The Pseudo-code for the training algorithm is listed in Algorithm 3. The Algorithm 2 is how we build the map and select subgoals by planning in a dynamic programming way.

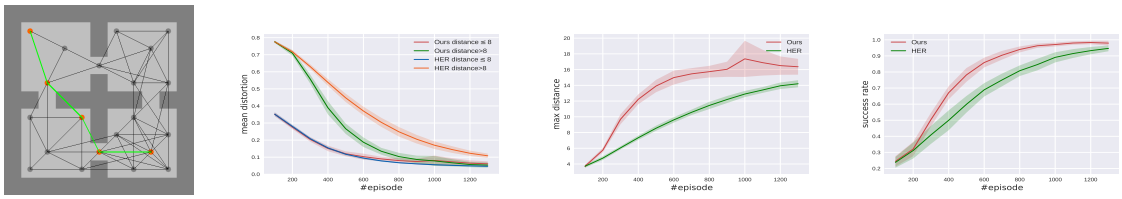
4.5.7 Implementation Details in Experiments

We use the DDPG architecture and hyperparameters for all the experiments in Table 4.1.

Algorithm 3. Train and Test with Planning

Input:current observation obs, desired goal g

for every training step **do**
    with probability α, action $\leftarrow$ Actor(obs, g) + noise
    with probability $1-\alpha$, action $\leftarrow$ Planner(obs, g)
    next_obs $\leftarrow$ env.step(action)
    Train actor and critic network with **hindsight experience replay**
    store trajectories in replay buffer when episode ends
end for
for every test step **do**
    action = Planner(obs, g)
    next_obs = env.step(action)
end for



(a) FourRoom **(b)** Mean Distortion **(c)** Max Distance Est. **(d)** Success Rate

Figure 4.2. The results on FourRoom Environment. Figure 4.2a shows the sampled landmarks and the planned path based on our algorithm. Figure 4.2c, 4.2b, 4.2d are different evaluation metrics of value estimation and success rate to reach the goal.

4.6 Experiments

4.6.1 FourRoom: An Illustrative Example

We first demonstrate the merits of our method in the FourRoom environment, where the action space is discrete. The environment is visualized in Figure 4.2a. There are walls separating the space into four rooms, with narrow openings to connect them. For this discrete environment, we use DQN [168] with HER [8] to learn the Q value. Here, we use the one-hot representation of the x-y position as the input of the network. The initial states and the goals are randomly sampled during training.

We first get $V(s,g)$ from the learned Q-value by equation $V(s,g) = \arg\max_a Q(s,a,g)$, and convert $V(s,g)$ to pairwise distance $D(s,g)$ based on Eq. 4.1. To evaluate the accuracy of

Table 4.1. Implementation Details in Experiments

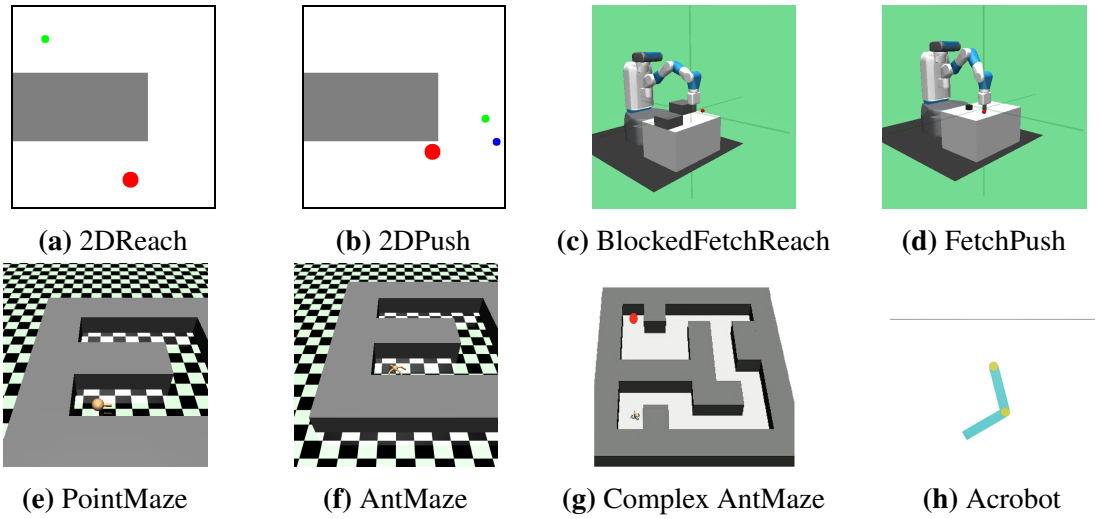
Parameter	**Value**
Q/Critic Network Layers	5
Q/Critic Network Hidden Dimension	400
Policy Network Layers	3
Policy Network Hidden Dimension	400
Network Activation	ReLU
Noise	OU-noise with $\sigma = 0.02$ (except AntMaze) 0.2 epsilon-greedy (AntMaze)
Discount Factor	0.99
Batch Size	128
Actor Learning Rate	0.0003
Critic Learning Rate	0.0003
Target Network Update Ratio	0.005
HER Replace Ratio	0.8
Episode Length	500 (PointMaze, AntMaze) 50 (Fetch/Push) 1500 (Complex AntMaze) 200 (Acrobot)
Distance Threshold δ	0.03 (2DMaze, 2DPush, Acrobot) 0.025 (FetchPush, FetchReach) 0.1 (PointMaze, AntMaze, Complex AntMaze)

distance estimation, we further calculate the ground truth distance $D_{gt}(s,g)$ by running a shortest path algorithm on the underlying ground-truth graph of maze. Then we adapt the mean distortion error (MDE) as the evaluation metric: $\frac{|D(s,g)-D_{gt}(s,g)|}{D_{gt}(s,g)}$.

Results are shown in Figure 4.2b. Our method has a much lower MDE at the very beginning stage, which means that the estimated value is more accurate.

To better evaluate our superiority for distant goals, we first convert predicted values to corresponding distances, and then plot the maximal distance during training. From Figure 4.2c, we can observe that the planning module have a larger output range than DQN. We guess that this comes from the max-operation in the Bellman-Ford equation, which pushes DQN to overestimate the Q value, or in other words, underestimate the distance for distant goals. However, the planner can still use piece-wise correct estimations to approximate the real distance to the goal.

We also compare our method with DQN on success reaching rate, and their performances



(a) 2DReach **(b)** 2DPush **(c)** BlockedFetchReach **(d)** FetchPush

(e) PointMaze **(f)** AntMaze **(g)** Complex AntMaze **(h)** Acrobot

Figure 4.3. The environments we use for continuous control experiments.

are shown in Figure 4.2d. Our method can achieve better accuracy at the early stage.

4.6.2 Continuous Control

In this section, we will compare our method with HER on challenging classic control tasks and MuJoCo [244] goal-reaching environments.

Environment Description

We test our algorithms on the following environments:

2DReach A green point in a 2D U-maze aims to reach the goal represented by a red point, as shown in Figure 4.3a. The size of the maze is 15×15. The state space and the goal space are both in this 2D maze. At each step, the agent can move within $[-1,1] \times [-1,1]$ as δ_x, δ_y in x and y directions.

2DPush The green point A now need to push a blue point B to a given goal (red point) lying in the same U-maze as 2DReach, as shown in Figure 4.3b. Once A has reached B, B will follow the movement of A. In this environment, the state is a 4-dim vector that contains the location of both A and B.

BlockedFetchReach & FetchPush We need to control a gripper to either reach a location in 3d space or push an object in the table to a specific location, as shown in Figure 4.3c and Figure 4.3d. Since the original FetchReach implemented in OpenAI gym [19] is very easy to solve, we further add some blocks to increase the difficulty. We call this new environment BlockedFetchReach.

PointMaze & AntMaze As shown in Figure 4.3e and Figure 4.3f, a point mass or an ant is put in a 12×12 U-maze. Both agents are trained to reach a random goal from a random location and tested under the most difficult setting to reach the other side of maze within 500 steps. The states of point and ant are 7-dim and 30-dim, including positions and velocities.

Complex AntMaze As shown in Figure 4.3g, an ant is put in a 56×56 complex maze. It is trained to reach a random goal from a random location and tested under the most difficult setting to reach the farthest goal (indicated as the red point) within 1500 steps.

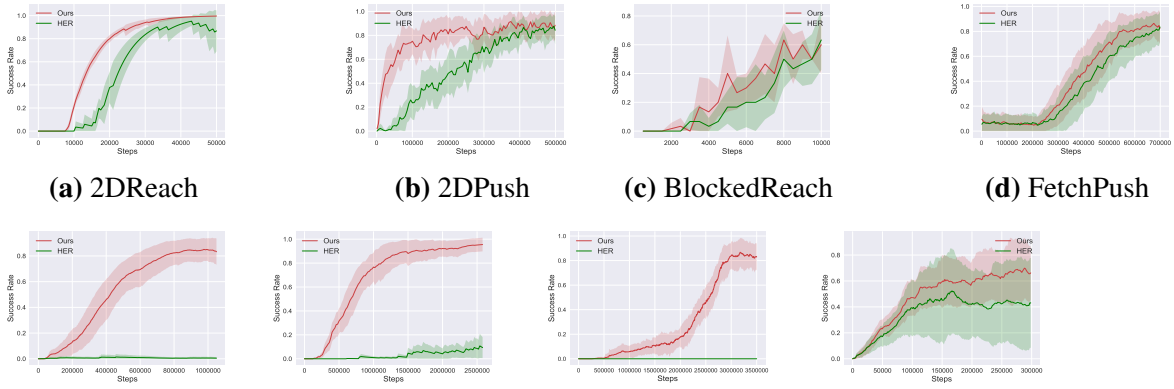
Acrobot As shown in Figure 4.3h, an acrobot includes two joints and two links. Goals are states that the end-effector is above the black line at specific joint angles and velocities. The states and goals are both 6-dim vectors including joint angles and velocities.

Experiment Result

The results compared with HER are shown in Figure 4.4. Our method trains UVFA with planner and HER. It is evaluated under the test setting, using the model and replay buffer at corresponding training steps.

In the **2DReach** and **2DPush** task (shown in Figure 4.4b), we can see our method achieves better performance. When incorporating with control tasks, for **BlockedFetchReach** and **FetchPush** environments, the results still show that our performance is better than HER, but the improvement is not so remarkable. We guess this comes from the strict time limit of the two environments, which is only 50. We observe that pure HER can finally learn well, when the task horizon is not very long.

We expect that building maps would be more helpful for long-range goals, which is



(a) 2DReach **(b)** 2DPush **(c)** BlockedReach **(d)** FetchPush

(e) PointMaze **(f)** AntMaze **(g)** Complex AntMaze **(h)** Acrobot

Figure 4.4. Experiments on the continuous control environments. The red curve indicates the performance of our method at different training steps.

evidenced in the environments with longer episode length. Here we choose **PointMaze** and **AntMaze** with scale 12×12. For training, the agent is born at a random position to reach a random goal in the maze. For testing, the agent should reach the other side of the "U-Maze" within 500 steps. For these two environments, the performance of planning is significantly better and remains stable, while HER can hardly learn a reliable policy. Results are shown in Figure 4.4e and Figure 4.4f.

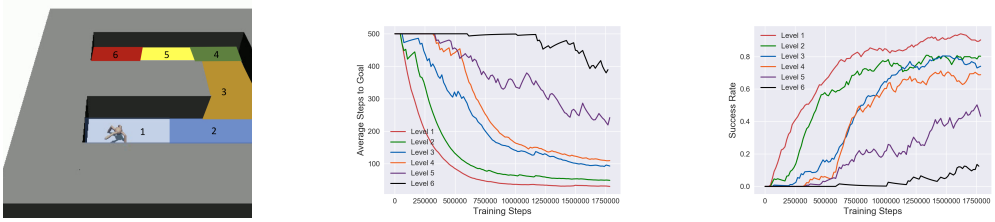
We also evaluate our method on classic control, and more complex navigation + locomotion task. Here we choose **Complex Antmaze** and **Acrobot**, and results are shown in Figure 4.4h and Figure 4.4g. The advantage over baseline demonstrates our method is applicable to complicated navigation tasks as well as general MDPs.

We also compare our method with Hierarchy RL on AntMaze and our method outperform recent Hierarchy RL methods. See supplementary material for details.

4.6.3 Ablation Study

We study some key factors that affect our algorithm on AntMaze.

Choice of clip range and landmarks There are two main hyper-parameters for the planner – the number of landmarks and the edge clipping threshold τ. Figure 4.6a shows the



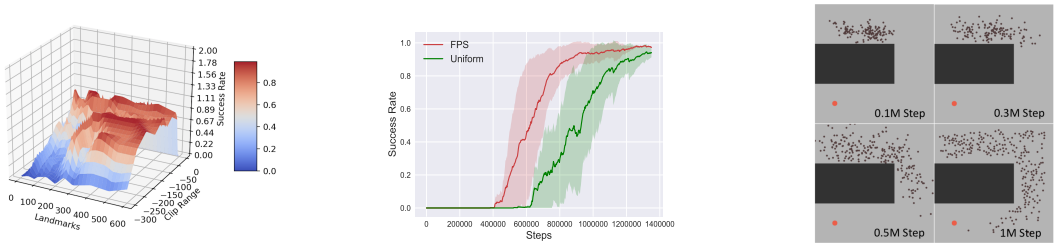
(a) Multi-level AntMaze **(b)** Average Steps **(c)** Success Rate

Figure 4.5. AntMaze of multi-level difficulty. Figure 4.5b and Figure 4.5c is the average steps and success rate to reach different level of goals, respectively.

evaluation result of the model trained after 0.8M steps in AntMaze. We see that our method is generally robust under different choices of hyper-parameters. Here τ is the negative distance between landmarks. If it's too small, the landmarks will be isolated and can't form a connected graph. The same problem comes when the landmarks are not enough.

The local accuracy of HER We evaluate our model trained between 0$\sim$2.5M steps, for goals of different difficulties. We manually define the difficulty level of goals, as shown in Figure 4.5a. Goal's difficulty increases from Level 1 to Level 6. We plot the success rate as well as the average steps to reach these goals. We find out that, for the easier goals, the agent takes less time and less steps to master the skill. The success rate and average steps also remain more stable during the training process, indicating that our base model is more reliable and stable in the local area.

Landmark sampling strategy comparison Our landmarks are dynamically sampled from the replay buffer by iterative FPS algorithm using distances estimated by UVFA, and get updated at the beginning of every episode. The FPS sampling tends to find states at the boundary of the visited space, which implicitly helps exploration. We test FPS and uniform sampling in fix-start AntMaze (The ant is born at a fixed position to reach the other side of maze for both training and testing). Figure 4.6b shows that FPS has much higher success rate than uniform sampling. Figure 4.6c shows landmark-based graph at four training stages. Through FPS, landmarks expand gradually towards the goal (red dot), even if it only covers a small proportion of states at the beginning.



(a) Hyperparameters of the planner **(b)** FPS vs. Uniform Sampling **(c)** Landmark-based Map

Figure 4.6. Figure 4.6a shows the relationship with the landmarks and clip range in the planner. Figure 4.6b shows FPS outperforms uniform sampling. And Figure 4.6c is the landmark-based map at different training steps constructed by FPS.

4.6.4 Comparison with HRL

We compare our method with HRL algorithms on large AntMaze (size 24×24), as shown in Table 4.2. We choose to compare with HIRO [176], which is the SOTA HRL algorithm on AntMaze, and HAC [126], which also uses the hindsight experience replay. We test these algorithms with the published codes[45], under both sparse reward setting and dense reward setting.

On sparse reward setting, our algorithm can work well and reach the goal at the very early stage (*Ours sparse* in Table 4.2). In contrast, neither HAC nor HIRO are able to reach the goal in 2M steps. HIRO doesn't use HER to replace the unachievable goals, which makes such setting very challenging for the algorithm.

For dense reward setting, the map planner can obtain a high success rate at very early stage shown as *Ours dense* in Table 4.2. Compared with *HIRO dense*, we can see that a planner can reach distant goals sooner, since we don't need to train a high-level policy to propose subgoals for the low-level agent.

HAC introduced several complex hyper-parameters, and we couldn't make it work well for both settings.

Table 4.2. Success Rate on Large AntMaze at different training steps.

	0.5M	0.75M	1M	1.25M	1.5M	1.75M	2M
Ours Sparse	0.0	0.03	0.3	0.4	0.45	0.5	0.5
HIRO Sparse	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Ours Dense	0.0	**0.09**	**0.45**	**0.5**	**0.7**	**0.8**	**0.9**
HIRO Dense	0.0	0.0	0.0	0.1	0.4	0.6	0.8

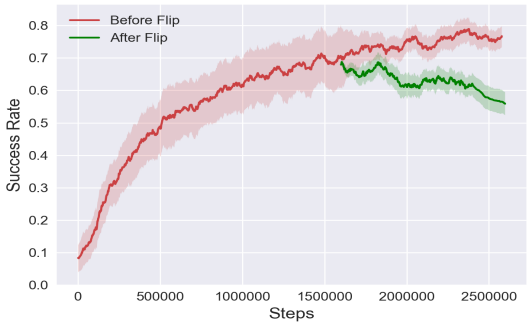


Figure 4.7. The success rate of HER to reach a random goal after we flip the training and testing setting.

4.6.5 The Forgetting Issue of HER

We observe that HER may forget how to reach the ultimate goal even if it learns to reach it some steps ago. For AntMaze, as shown in the main paper, the success rate for pure HER is always below 0.2. Since the goal for testing is the most difficult one, to better evaluate this issue for a larger goal space $\mathscr{G}$ of different difficulties, we then flip the setting for training and testing, i.e., for training, the agent aims to reach a fixed goal at the other side of the maze, but for testing, the agent is born at a random location and tries to reach a random goal. Here we use a well-pretrained model, which has almost 0.7 success rate to reach a random sampled goal within 200 steps. We then retrain it to reach a fixed goal under the new setting. We observe that, although its performance to reach a fixed goal is slowly increasing, its ability to reach a randomly picked goal in the maze drops to $0.5 \sim 0.6$.

[4]HIRO: https://github.com/tensorflow/models/tree/master/research/efficient-hrl
[5]HAC:https://github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-

4.7 Conclusion

Learning a structured model and combining it with RL algorithms are important for reasoning and planning over long horizons. We propose a sample-based method to dynamically map the visited state space and demonstrate its empirical advantage in routing and exploration in several challenging RL tasks. Experimentally we showed that this approach can solve long-range goal reaching problems better than model-free methods and hierarchical RL methods, for a number of challenging games, even if the goal-conditioned model is only locally accurate. However, our method also has limitations. First, we empirically observe that some parameters, particularly the threshold to check whether we have reached the vicinity of a goal, needs hand-tuning. Secondly, a good state embedding is still important for the learning efficiency of our approach, since we do not include heavy component of learning state embedding. Thirdly, we find that in some environments whose intrinsic dimension is very high, especially when the topological structure is hard to abstract, sample-based method is not enough to represent the visited state space. And for those environments which is hard to obtain a reliable and generalizable local policy, this approach will also suffer from the accumulated error.

Acknowledgements

Chapter 4, in full, is a reprint of the material published in Advances in Neural Information Processing Systems 32 (2019). Mapping State Space using Landmarks for Universal Goal Reaching; Huang, Zhiao; Liu, Fangchen; Su, Hao. The dissertation author was the primary investigator and author of this paper.

Chapter 5

DiffVL: Scaling Up Soft Body Manipulation using Vision-Language Driven Differentiable Physics

Combining gradient-based trajectory optimization with differentiable physics simulation is an accurate and efficient technique for solving soft-body manipulation problems. Using a well-crafted optimization objective, the solver can quickly converge onto a valid trajectory. However, writing the appropriate objective functions requires expert knowledge, making it difficult to collect a large set of naturalistic problems from non-expert users. We introduce DiffVL, a framework that integrates the process from task collection to trajectory generation leveraging a combination of visual and linguistic task descriptions. A DiffVL task represents a long horizon soft-body manipulation problem as a sequence of 3D scenes (key frames) and natural language instructions connecting adjacent key frames. We built GUI tools and tasked non-expert users to transcribe 100 soft-body manipulation tasks inspired by real-life scenarios from online videos. We also developed a novel method that leverages large language models to translate task language descriptions into machine-interpretable optimization objectives, which can then help differentiable physics solvers to solve these long-horizon multistage tasks that are challenging for previous baselines. Experiments show that existing baselines cannot complete complex tasks, while our method can solve them well. Videos can be found on the website https://sites.google.com/view/diffvl/home.

5.1 Introduction

This work focuses on soft body manipulation, a research topic with a wide set of applications such as folding cloth [146, 253, 100], untangling cables [251, 250], and cooking foods [221, 207]. Due to their complicated physics and high degree of freedom, soft body manipulations raise unique opportunities and challenges. Recent works such as [96] and [264] have heavily leveraged various differentiable physics simulators to make these tasks tractable. Towards generalizable manipulation skill learning, such approaches have the potential to generate data for learning from demonstration algorithms [144, 130]. However, to enable differentiable physics solvers to generate a large scale of meaningful trajectories, we must provide them with suitable tasks containing the scene and the optimization objectives to guide the solver. Previous tasks are mostly hand-designed [96] or procedure-generated [144], resulting in a lack of diverse and realistic soft-body manipulation tasks for researchers.

This work takes another perspective by viewing tasks as data, or more precisely, task specifications as data. Each data point contains a pair of an initial scene and an optimization objective depicting the goal of the task. Taking this perspective, we can annotate meaningful tasks like annotating data of other modalities, like texts, images, videos, or each action trajectory, by leveraging non-expert human labor, providing possibilities for scaling up the task space of differentiable physics solvers.

A core problem in building a task collection framework with trajectory generation is finding a suitable representation of the tasks. The representation should be intuitive for non-expert annotators while being accurate enough to describe the complex shapes and motions of the soft bodies that appear in various soft body manipulation processes. Last but not least, it should be reliably interpreted by differentiable physics solvers to yield a valid trajectory.

Studies such as [5, 3, 162, 252] have shown that humans are proficient at defining goal-driven, spatial object manipulation tasks using either sketches, natural languages, or both. Figure 5.1(A), for instance, depicts the complete process of dumpling creation, supplemented

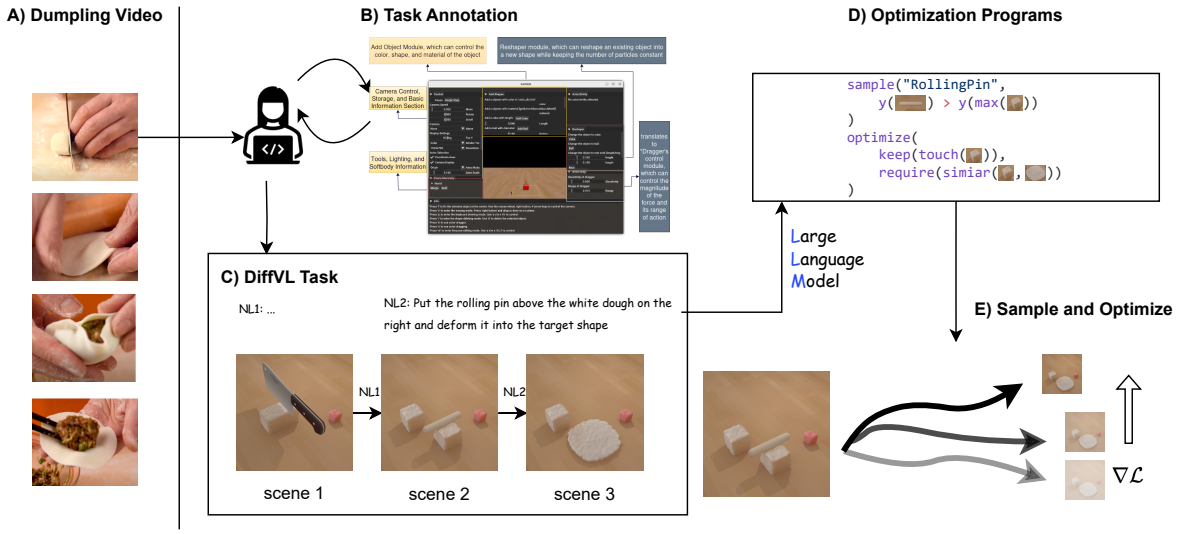


Figure 5.1. (A) A dumpling making video; (B) The annotator interacts with our GUI tool to create DiffVL tasks; (C) A DiffVL task contains a sequence of 3D scenes along with natural language instructions to guide the solver; (D) DiffVL leverages a large language model to compile instructions into optimization programs consisting of vision elements; (E) The optimization program guides the solver to solve the task in the end.

by textual instructions. Taking this as an inspiration, we present DiffVL, a framework that enables non-expert users to specify soft-body manipulation tasks to a differentiable solver using a combination of vision and language. Specifically, each DiffVL task consists of a sequence of 3D scenes (keyframes), with natural language instruction connecting adjacent keyframes. The sequence of key frames specifies the sequence of *subgoals* of the manipulation task, and the natural language *instructions* provides suggestions on how to use the actuators to manipulate the objects through this sequence of subgoals. See Figure 5.1.

We develop tools to ease the annotation for non-expert users. With our interactive simulator equipped with a GUI, the user can edit, manipulate, draw, or carve shapes easily like other 3D editing tools and observe the consequence through simulation in an intuitive manner. Meanwhile, when it is tedious for users to edit all the intermediate steps of a complex motion, they can use natural language to describe the goal instead of drawing them step by step. This enables us to build **SoftVL100**, a vision-language dataset with 100 diverse tasks. DiffVL uses LLM to compile the natural language instructions to an optimization program – which enforces

a set of constraints during the soft-body manipulations from one keyframe to the next. This optimization program is then used by a differentiable physics solver to generate a working trajectory.

To summarize, our work makes the following contributions:

- We propose a new multi-stage vision-language representation for defining soft-body manipulation tasks that is suitable for non-expert user annotations.
- We developed a corresponding GUI, and curated SoftVL100, consisting of 100 realistic soft-body manipulation tasks from online videos[1].
- We develop a method, DiffVL, which marries the power of a large-language model and differentiable physics to solve a large variety of challenging long-horizon tasks in SoftVL100.

5.2 Related Work

Differentiable simulation for soft bodies Differentiable physics [91, 257, 195, 55, 90] has brought unique opportunities in manipulating deformable objects of various types [155, 89, 147, 99, 256, 140, 259, 234, 167, 265, 259, 153, 88, 136, 212, 64], including plasticine [96], cloth [133], ropes [149] and fluids [261] simulated with mass-spring models [91], Position-based Dynamics [174], Projective Dynamics [195, 44], MPM [104] or Finite Element Method [81], which could be differentiable through various techniques [91, 44, 154, 17]. It has been shown that soft bodies have smoother gradients compared to rigid bodies [96, 91, 257, 9, 232], enabling better optimization. However, the solver may still suffer from non-convexity [132] and discontinuities [232], motivating approaches to combine either stochastic sampling [132], demonstrations [130, 9] or reinforcement learning [263, 171] to overcome the aforementioned issues. DiffVL capitalizes on an off-the-shelf differentiable physics solver to annotate tasks. It incorporates human priors through a novel vision-language task representation. This allows users

[1] both the GUI and dataset will be made public

to employ natural language and GUI to guide the differentiable solver to solve long-horizon tasks, distinguishing DiffVL from previous methods.

Task representation for soft body manipulation There are various ways of defining tasks for soft body manipulation. To capture the complex shape variations of soft bodies, it is natural to use either RGB images [259, 144, 208] or 3D point cloud [222, 136] to represent goals. However, given only the final goal images, it may be hard for the solver to find a good solution without further guidance. Besides, generating diverse images or point clouds that adhere to physical constraints like gravity and collision avoidance in varied scenes can be challenging without appropriate annotation tools. Our GUI tools are designed to address these issues, simplifying the process. Other works use hand-defined features [26, 251] or formalized languages [128]. These methods necessitate specialist knowledge and require tasks to be defined on a case-by-case basis. Learning from demonstrations [130, 208] avoids defining the task explicitly but requires agents to follow demonstration trajectories and learn to condition on various goal representations through supervised learning or offline reinforcement learning [57, 51, 223, 71, 152, 102, 28, 4, 7]. However, collecting demonstrations step-by-step [196, 275, 130] requires non-trivial human efforts, is limited to robots with human-like morphologies, and may be challenging for collecting tasks that are non-trivial for humans (e.g., involving complex dynamics or tool manipulation). Our annotation tool emphasizes the description of tasks over the identification of solutions. Annotators are merely required to define the task, not execute the trajectories themselves, thereby simplifying the annotation process. Our method is also related to [135], but we consider a broader type of tasks and manipulation skills.

Language-driven robot learning Many works treat language as subgoals and learn language-conditioned policies from data [224, 61, 177, 129, 20, 164, 163] or reinforcement learning [105, 50, 237, 32]. It has been shown that conditioning policies on languages lead to improved generalization and transferability [224, 105]. [219] turns languages into constraints to correct trajectories. Our method distinguishes itself by integrating language with differentiable physics through machine-interpretable optimization programs. Recent works aim at leveraging

large language models [6, 43, 258, 143, 35] to facilitate robot learning. Most use language or code [139] to represent policies rather than goals and focus on high-level planning while relying on pre-defined primitive actions or pre-trained policies for low-level control. In contrast, our language focuses on low-level physics by compiling language instructions to optimization objectives rather than immediate commands. It's noteworthy that VIMA [106] also introduces multimodal prompts, embodying a similar ethos to our vision-language task representation. Their approach views multimodal representation as the task prompts to guide the policy. However, it still requires pre-programmed oracles to generate offline datasets for learning, which is hard to generalize to soft-body tasks with complex dynamics. Contrastingly, we adopt a data annotation perspective, providing a comprehensive framework for task collection, natural language compiling, and trajectory optimization, enabling us to harness the potential of differentiable physics effectively.

Task and motion planning Our optimization program compiled from natural language is closely connected to the field of Task and Motion Planning (TAMP) [62], particularly Logic Geometric Programming (LGP) [246, 247]. We draw inspiration from LGP's multiple logical states in designing our DiffVL task representation. However, unlike LGP, our approach asks human annotators to assign "logic states" and facilitates the manipulation of soft bodies through our novel vision-language representation. Some recent works have explored the intersection of TAMP and language models [41, 227, 157] while primarily focusing on generating high-level plans.

5.3 SoftVL100: Vision-Language Driven Soft-body Manipulation Dataset

In this representation, the original tasks are divided into a sequence of 3D scenes, or key frames, along with text descriptions that detail the steps for progressing to the next scene. Figure 5.2(A) illustrates our representation. Each key frame consists of multiple soft bodies with

Figure 5.2. (A) Example of multi-stage tasks and their text annotations; (B) Snapshots of scenes in SoftVL100 dataset.

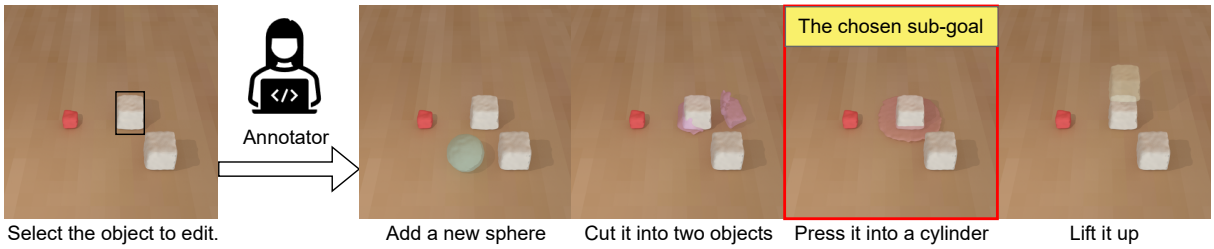


Figure 5.3. Example operations in GUI tools

different positions and shapes. By grouping two consecutive key frames together, we form a manipulation stage. Natural language instructions may guide the selection of an actuator and provide guidance on how to control it to reach the subsequent key frames. In Section 5.3.1, we develop annotation tools to facilitate the task representation annotation process. These tools make it easy to construct a diverse set of tasks, referred to as SoftVL100, as discussed in Section 5.3.2.

5.3.1 Vision-Language Task Annotator

Our annotation tool is based on PlasticineLab [96], a simulation platform that utilizes the Material Point Method (MPM) [92] for elastoplastic material simulation. To enhance the user experience and support scene creation, we integrate it into SAPIEN [262], a framework that enables the development of customized graphical user interfaces.

Our simulator runs on GPU servers and the interface is accessible as a web service through VNC, allowing users to interact with the simulator directly from their web browser. When starting a new task annotation, users can open the simulator, perform actions, and save the current scene. Our GUI tools provide comprehensive support for creating, editing, managing, and simulating objects, as illustrated in Figure 5.3. **Scene creation:** The GUI offers a variety of primitive shapes, such as ropes, spheres, cubes, and cylinders, that users can add to the scene. Before adding a soft-body shape, users can adjust its size, color, and materials (e.g., rubber, fiber, dough, iron) if needed. **Shape editing:** Users have the ability to select, edit, or delete shapes within the scene. The GUI tool includes a range of shape operators, including moving, rotating, carving, and drawing. **Simulation:** Since the interface is built on a soft body simulator, users can simulate and interact with the soft body using rigid actuators or magic forces. They can also run simulations to check for stability and collision-free states. **Object management:** Users can merge connected shapes to create a single object or divide an object into two separate ones with distinct names. The annotation tool keeps tracking the identities of objects across scenes.

Once users have finished editing the scene and creating a new key frame, they can save the scene into the key frame sequence. The key frames are displayed below the simulator, visualizing the current task annotation progress. Users have the option to open any of these saved scenes in the simulator, delete a specific scene if needed, and add text annotations for each scene.

5.3.2 The SoftVL100 Dataset

Our annotation tool simplifies the task creation process, enabling us to hire non-expert users to collect new tasks to form a new task set, SoftVL100. The task collection involves several stages, starting with crawling relevant videos from websites like YouTube that showcase soft body manipulation, particularly clay-making and dough manipulation. We developed tools to segment these videos and extract key frames to aid in task creation.

We hired students to annotate tasks, providing them with a comprehensive tutorial and step-by-step guidelines for using our annotation tools. It took approximately two hours for each

annotator to become proficient in using the annotation tool. Subsequently, they were assigned a set of real-world videos to create similar tasks within the simulator. After collecting the keyframes, we proceed to relabel the scenes with text descriptions. Annotators are provided with a set of example text descriptions. We encourage users to include detailed descriptions of the actuators, their locations, moving directions, and any specific requirements, such as avoiding shape breakage for fragile materials. The annotation process for each task typically takes around 30 minutes. Using our task annotation tool, we have created a dataset called SoftVL100, which consists of 100 tasks, and there are more than 4 stages on average. The tasks cover a wide range of skills as illustrated in Figure 5.2 (A). Sample tasks from the dataset can be seen in Figure 5.2 (B).

5.3.3 Details of choosing keyframes

The selection of keyframes is determined by the annotators, as we believe humans can naturally decompose complex tasks into simpler ones for communication. When annotators manipulate a scene within the GUI, they can save the current scene as a key frame within the task representation. They can also add, modify, or delete key frames using the editors within the web server interface.

The annotators were instructed on what kind of decomposition would likely result in a working trajectory, i.e., segmenting the manipulation processes at contact point changes., which is a simple and effective way for the annotator to provide high-quality labels, without having any expert understanding of the physics and the solver. If the agent is required to manipulate a new object or the actuator needs to establish contact on a different face of the object, annotators would introduce a new keyframe. Additionally, we employ heuristic methods to segment YouTube videos. This segmentation occurs when there is a significant disparity between two frames, aiming to simplify the keyframe selection for annotators.

5.4 Optimization with Vision-Language Task Description

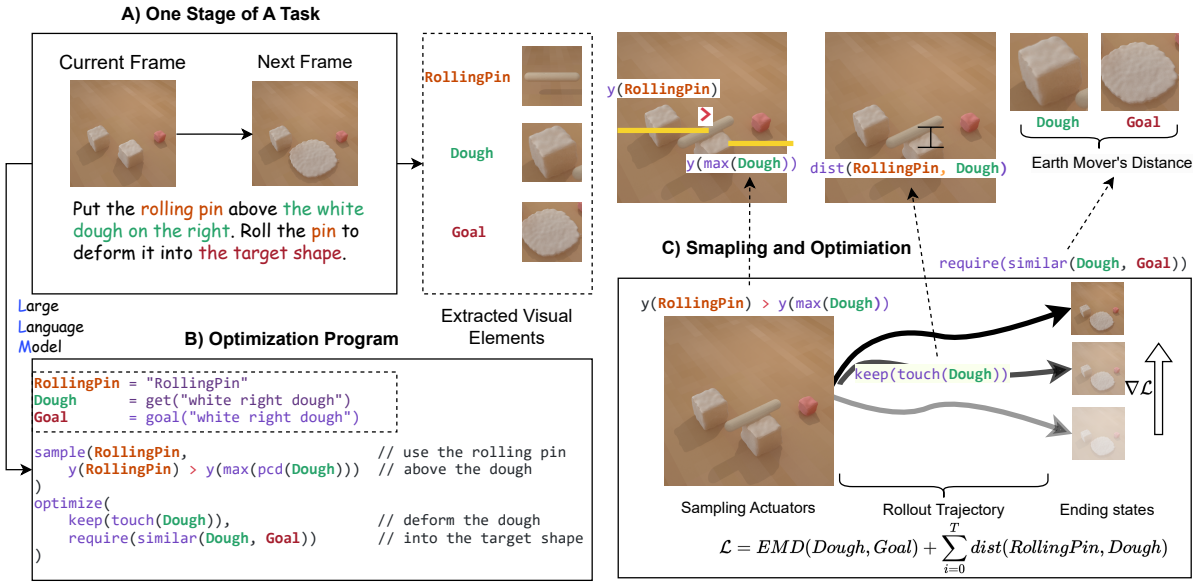


Figure 5.4. (A) One stage of the DiffVL task of two keyframes and a natural language instruction. (B) The compiled optimization program. (C) The sampling and optimization process.

We propose DiffVL to tackle the challenging tasks in SoftVL100. Given a stage of a task depicted in Figure 5.4(A), DiffVL utilizes large language models to compile the natural language instructions into a machine-interpretable optimization program in Figure 5.4(B). The program comprises of the names of visible elements and Python functions, effectively capturing the essence of the language instructions. We introduce the design of our DSL in Section 5.4.1 and outline the DiffVL compiler based on large language models (LLM) in Section 5.4.2. The resulting optimization program includes crucial information for selecting and locating actuators and can facilitate a differentiable physics solver to generate valid trajectories, as discussed in Section 5.4.3.

5.4.1 Optimization Program

The optimization program is formulated using a specialized domain-specific language (DSL) that incorporates several notable features. Firstly, it includes functions that extract the

names of visible elements from the 3D scenes and support operations on these elements, represented as 3D point clouds. This enables manipulation and analysis of the visual information within the optimization program. Secondly, the DSL provides various functions to express geometric and temporal relations, encompassing common geometric and motion constraints. These functions facilitate the representation and handling of natural language instructions. Furthermore, the DSL clauses are interpreted into PyTorch [187], making the constraints automatically differentiable and directly applicable for differentiable physics solvers.

The example in Figure 5.4 already exposes multiple components of our DSL. In the program, we can refer to the names of visible elements through `get` and `goal` functions, which can look for and extract the 3D objects that satisfy the description, identified by their color, shapes, and locations (we ensured that these attributes are sufficient to identify objects in our tasks). The program will then start with a `sample` statement, which is used to specify the actuators and their associated constraints over initial positions. Actuators refer to the entities utilized for manipulation, such as a robot gripper, a knife, or the rolling pin depicted in Figure 5.4. The constraint `y(RollingPin) > y(max(pcd(Dough)))` within the `sample` statement represents a specific constraint. It ensures that the height of the actuator (RollingPin) is greater than the highest y coordinate of the white dough in the scene, reflecting the meaning of "put bove" of the language instructions. Following the `sample` statement, an `optimize` statement is included. This statement plays a vital role in defining the constraints and objectives for the manipulation trajectories. These constraints and objectives serve as the differentiable optimization objectives that the differentiable physics solver aims to maximize. In an optimization program, two special functions, namely `require` and `keep`, can be used to specify the temporal relationship of the optimization objectives. By default, the `require` function evaluates the specified condition at the end of the trajectories. On the other hand, the `keep` function is applied to each frame of the trajectories, ensuring that the specified conditions are maintained throughout the trajectory. In the example program shown in Figure 5.4(B), the optimization program instructs the actuator to continuously make contact with the white dough. The function `similar` computes the Earth

Mover distance [206] between two objects' point clouds, which aims to shape the dough into a thin pie, the goal shape extracted from the next key frame. We illustrate more example optimization programs in Figure 5.5.

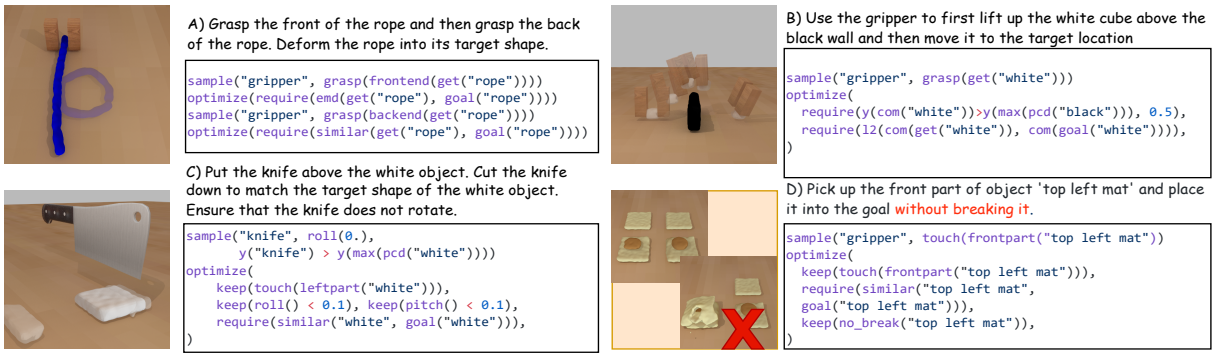


Figure 5.5. A) Decomposing a stage into sub-stages: manipulating a rope into a circle using a single gripper requires the agent to first grasp the upper part of the rope and then manipulate the other side to form the circle. The program counterpart contains two pairs of `sample` and `optimize`, reflecting the two sub-stages in the language instruction. `backend` extracts the back end of 3D objects; B) Guiding the motion of the objects to avoid local optima: we use `com` to compute the center of the mass of objects and `pcd` to get the clouds; relation operators compare two objects' coordinate. The first `require` takes an additional argument 0.5, meaning the inner clause, which asks the white object above the black one, should be evaluated halfway through the trajectory to represent meanings of 'first do A and then do B.' C) Selecting a suitable tool to split the objects; D) The program can include additional constraints like not breaking the shape, which is critical for manipulating fragile materials.

5.4.2 Compiling Natural Languages with LLM

We utilize the power of the Large Language Model (LLM) to compile text instructions into optimization programs. This approach capitalizes on the few-shot capabilities of the LLM and employs various few-shot prompts [255, 21]. The prompts begin by defining the "types" and "functions" within our DSL. Each function is accompanied by a language explanation to clarify its purpose. Additionally, examples are given to demonstrate how primitive functions can be combined to form complex clauses. To facilitate the translation, we provide the language model with both the scene description of the objects that it contains and the natural language instructions that we wish to compile. We then prompt the model with the instruction, "Please translate the

instructions into a program." This step is repeated for all frames, resulting in their respective optimization programs. Through our experimentation, we have observed that GPT-4 [181] surpasses other models in terms of performance.

5.4.3 Solving the Optimization through Differentiable Physics

We leverage the differentiable physics solver in [96], which has been proven accurate and efficient in solving the generated optimization programs. The process is illustrated in Figure 5.4(C). The `sample` function is employed within a sampling-based motion planner. Initially, it samples the pose of the specified actuator type to fulfill the provided constraints. Subsequently, an RRT planner [122] determines a path toward the target location, and a PD controller ensures that the actuators adhere to the planned trajectory. With the actuator suitably initialized, the `optimize` clause is then passed to a gradient-based optimizer to optimize the action sequence for solving the task. The solver performs rollouts, as depicted in Figure 5.4(C), where the conditions enclosed by `keep` operators are evaluated at each time step. Conversely, conditions enclosed by `require` are evaluated only at specified time frames. The resulting differentiable values are accumulated to calculate a loss, which is utilized to compute gradients for optimizing the original trajectories. For handling multistage tasks involving vision-language representation, we can solve them incrementally, stage by stage.

5.5 Experiments

In this section, we aim to justify the effectiveness of our vision-language task representation in guiding the differentiable physics solver. To better understand the performance of different solvers, we evaluate the tasks of two tracks. The first track focuses on tasks that only require agents to transition from one key frame to another. Tasks in this track are often short-horizon. This setup allows us to evaluate the efficiency and effectiveness of the differentiable physics solver in solving short-horizon soft body manipulation tasks, as described in Section 5.5.1. In the second track, we validate the agent's capability to compose multiple key frames in order to

solve long-horizon manipulation tasks. These tasks involve complex scenarios where agents need to switch between different actuators and manipulate different objects. The evaluation of this composition is discussed in Section 5.5.2.

We clarify that our approach does not use a gradient-based optimizer to optimize trajectories generated by the RRT planner. Motion planning and optimization occur in separate temporal phases. Initially, we plan and execute a trajectory to position the actuator at the specified initial pose. Subsequently, we initialize a new trajectory starting from this pose, and the actuator remains fixed in this trajectory as we initialize the policy with zero actions to maintain its position. The sample function generates poses in line with the annotators' guidance.

5.5.1 Mastering Short Horizon Skills using Differentiable Physics

Before evaluating the full task set, we first study how our vision-language representation can create short-horizon tasks that are solvable by differentiable physics solvers. We picked 20 representative task stages as our test bed from the SoftVL100. We classify these tasks into 5 categories, with 4 tasks in each category. The categories are as follows: **Deformation** involves the deformation of a large single 3D shape, achieved through pinching and carving; **Move** asks the movement of an object or stacking one object onto another; **Winding** involves the manipulation of a rope-like object to achieve different configurations. **Fold** focuses on folding a thin mat to wrap other objects. **Cut** involves splitting an object into multiple pieces. The selected tasks encompass a wide spectrum of soft body manipulation skills, ranging from merging and splitting to shaping. We evaluate different methods' ability to manipulate the soft bodies toward their goal configuration. The agents are given the goal configuration in the next frame, and we measure the 3D IoU of the ending state as in [96]. We set a threshold for each scene's target IoU score to measure successful task completion, accommodating variations in IoU scores across different scenes.

We compare our method against previous baselines, including two representative reinforcement learning algorithms, Soft Actor-Critic (SAC) [74] and Proximal Policy Gradient

(PPO) [216]. The RL agents perceive a colored 3D point cloud as the scene representation and undergo 10^6 steps to minimize the shape distance towards the goal configuration while contacting objects. Additionally, we compare our approach with CPDeform [132], which employs a differentiable physics solver. However, CPDeform uses the gradient of the Earth mover's distance as a heuristic for selecting the initial actuator position. It differs from DiffVL which utilizes the text description to determine ways to initialize the actuators and the optimization objectives. For differentiable physics solvers, we run Adam [113] optimization for 500 gradient steps using a learning rate of 0.02.

Table 5.1 presents the success rates and mean IoU of various approaches. It is evident that, except for specific deformation and cutting tasks where SAC shows success, the RL baselines struggle to solve most tasks. Although RL agents demonstrate an understanding of how to manipulate the soft bodies correctly, they lack the precision required to match the target shape accurately. CPDeform achieves partial success in each task category. It surpasses the RL baselines by utilizing a differentiable physics solver and employing a heuristic that helps identify contact points to minimize shape differences. However, CPDeform's heuristic lacks awareness of physics principles and knowledge about yielding objects to navigate around obstacles. Consequently, CPDeform fails to perform effectively in other tasks due to these limitations. We want to emphasize that integrating language into RL baselines entails non-trivial challenges. For example, many reward functions in our optimization program have temporal aspects, making the original state non-Markovian. As we focus on enhancing the optimization baseline through language-based experiments, we've chosen to defer tackling the complexities of language-conditioned RL to future research.

Our approach surpasses all other methods, achieving the highest overall success rate. To examine the impact of different components and illustrate their effects, we conduct ablation studies. First, we remove the constraints in `sample` named "- Sample" in the table and ask the agent to sample random tools without using hints from the language instructions. As a result, the performance drops significantly, either getting stuck at the local optima or failing to select

Table 5.1. Single stage experiment results. The metrics we use are Success Rate (SR) and 3D Intersection Over Union (IOU).

	previous baselines			DiffVL		
	SAC	**PPO**	**CPDeform**	**- Sample**	**- Optimize**	**Ours**
	SR/IOU	**SR/IOU**	**SR/IOU**	**SR/IOU**	**SR/IOU**	**SR/IOU**
Deform	0.25 (0.504)	0.00/0.392	0.25/0.532	0.11/0.489	0.44/0.524	**1.00/0.564**
Move	0.00/0.541	0.00 /0.527	0.08 /0.570	0.00/0.533	0.33/0.618	**1.00/0.641**
Wind	0.00/0.308	0.00/0.289	0.21/0.362	0.25 /0.351	0.08 /0.408	**0.59/0.446**
Fold	0.00/0.572	0.00 /0.457	0.33/0.612	0.00/0.550	**1.00 /0.650**	0.94/0.643
Cut	0.33/0.449	0.00/0.408	**0.89/0.482**	0.33/0.357	0.56/0.447	0.87/0.490
Total	0.12/0.475	0.00/0.415	0.35/0.512	0.14/0.456	0.48/0.529	**0.88/0.557**

suitable tools to finish the task. For example, in the cutting task, it failed to select knives and split the white material into two pieces. On the other hand, the variant "- Optimize" replaces the statement in `optimize` by an objective focusing solely on optimizing shape distance, removing additional guidance for the physical process. The effects are lesser compared to the previous variant, demonstrating that with a suitable tool initialization method, a differentiable physics solver is already able to solve many tasks. However, as illustrated in Figure 5.6(A), it does not lift the white cube to avoid the black wall in a moving task and breaks the mat shown in the second row. In the winding task, it fails to find the relatively complex lifting-then-winding plan directly from shape minimization. In contrast, our method can successfully solve these tasks after leveraging the trajectory guidance compiled from the natural language instructions. We find that our solver is quite efficient. For a single-stage task, it takes 10 minutes for 300 gradient descent steps on a machine with NVIDIA GeForce RTX 2080, for optimizing a trajectory with 80 steps. For most tasks, 300 gradient steps are sufficient for finding good solutions.

5.5.2 Driving Multi-Stage Solver using Vision-Language

Having evaluated short-horizon tasks, we apply our method to long-horizon tasks in SoftVL100. We leverage the natural language instructions to generate the evaluation metrics for each task, which might measure the relationships between objects, check if soft materials

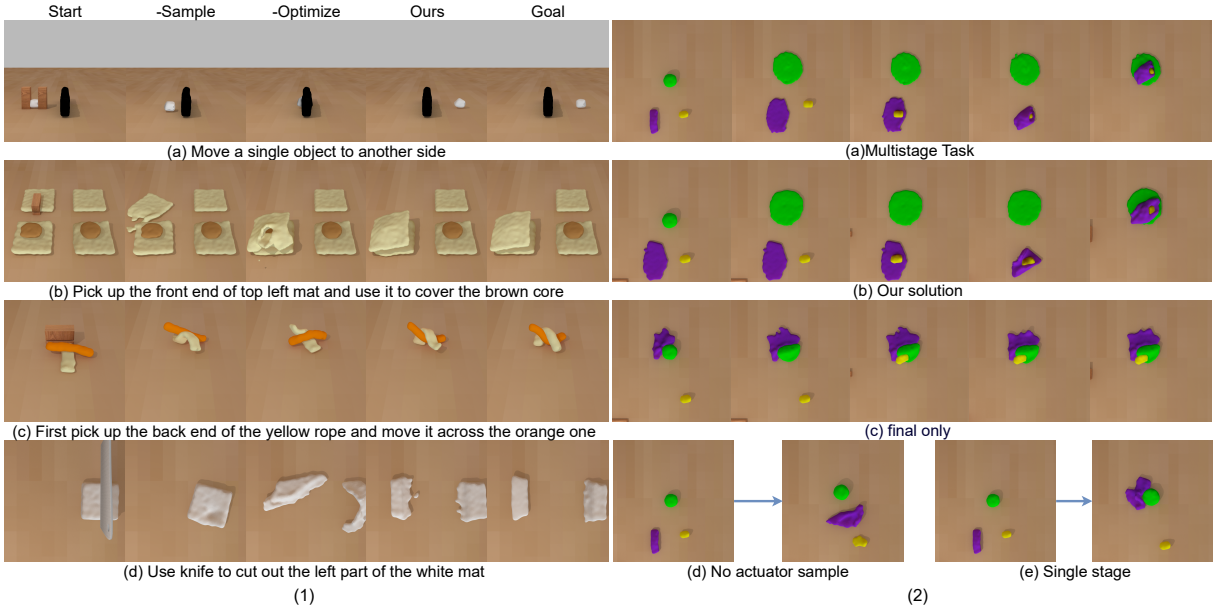


Figure 5.6. (1) Performance on single stage tasks. (2) Performance on a multistage task that includes moving, packing, and pressing.

rupture, or if two shapes are similar.

Our multistage vision-language task representation decomposes the long-horizon manipulation process into multiple short-horizon stages, allowing our method to solve them in a stagewise manner. As baseline algorithms fail to make notable progress, we apply ablation to our method: We first compare our method with a single-stage approach (single), which refers to solving long-horizon tasks like short-horizon tasks, by dropping the key frames and directly optimizing for the final goal. We then dropped the actuator sampling process in the middle (no actuator sample). The solver is still told to optimize for key frames (sub-goals) at each stage but was not informed to switch actuators and objects to manipulate. As expected, dropping the stage information and the language instructions in our vision task representation significantly degrades the performance, making solvers unable to complete most tasks. After leveraging the language instructions, the FinalOnly solver removes the vision subgoals in each sub-stage but provides the agent's actuators to choose the objects to manipulate. In this case, similar to the "-Sample" variant, the differentiable physics solver can solve certain tasks by only minimizing

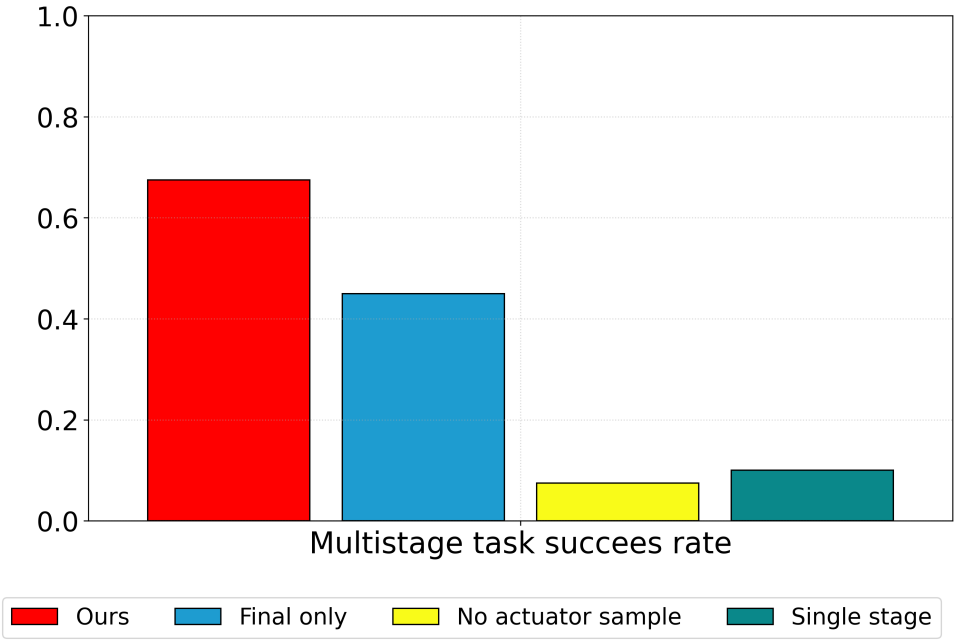


Figure 5.7. Success rate of multi-stage tasks of different ablations

shape distance. However, it may have troubles for tasks that require objects to deform into multiple shapes. For example, as illustrated in Figure 5.6(B), the solver needs to first compress the purple shape into a thin mat to warp the yellow dough.

The solver fails to discover the compression process after removing the subgoals but directly moves the purple toward its final configurations, resulting in the failure in the end. This showcases the importance of the introduction of the key frames in our vision-language task annotation.

5.6 Additional Details of DiffVL

In this section, we provide additional details on the dataset, the GUI tools, and the optimization program used in the DiffVL framework. We also provide a comparison between GPT3.5 and GPT4 in the translation task.

5.6.1 DSL for Optimization Programs

Table 5.2 lists the functions and primitives in our domain-specific language.

Table 5.2. Elements in the optimization program

Category	FuncName	Explanation
Objects	get(desc)	Get the point cloud of the objects with the description, for example, get("left white mat"). If the desc is 'all', then get all the objects.
	goal(descr)	Get the point cloud of the goal for the objects with the given name. The name can also be 'all'.
Temporal conditions	keep(cond, start=0, end=1)	Minimize the cond from time s to time t. It only takes one cond as an argument.
	require(cond, end=1)	Reach the condition at time t. By default end is 1.
	and(cond1, cond2, ...)	Needs to satisfy all constraints
Shape operators	com(shape)	Compute the center of the mass of the point cloud
	similar(A, B)	Compare the shape distance using EMD distance
	pcd(shape)	Get the PyTorch tensor that represents the point cloud of the shape
	leftpart, rightend, etc.	Get a part of the point cloud based on the description.
Actuator	touch(ShapeA)	Minimize the signed distance function of the actuators and the shape
	away	Check if the actuator is far away from the shape
	roll, vertical, etc.	Get the rotation angle of the actuator
Tensor operators	Relation $>, <, \geq, \leq$	Compare relationships of two vectors or scalar
	Algebra $+, -$	Add values to something.
	l_2(A, B)	Compare the l_2 distance
	min, max	Get the min max of a tensor
Soft Body Constraints	fix_shape	Do not change the shape of the objects for too much
	fix_place	Keep the object not moving
	no_break	Do not break the objects
Special	stage(sample_fn, optimize_fn)	An optimization stage that may involve a motion planning procedure and an optimization procedure. It is only used for compiling.
	sample(tool_name, *conds)	Select the tool of tool_name. We provide "knife", "board", "rolling pin" and "gripper". The conds are list of conditions that we hope the tool satisfy
	optimize(*conds)	Optimize the trajectory to satisfy the conditions.

We rely on the large language model's few-shot ability to compile the natural language description like "lift up the white cube above the black wall" into an executable optimization program. The prompt start with the introduction to the DSL, listing the functions as well as the explanations, followed by several examples illustrating how to compose those functions to implement complex functions, e.g., by checking the x coordinates of two objects to decide if one is on the left of the other. Then we provide several pairs of natural language inputs and corresponding programs. In the end, we provide the natural language input to compile. Below is an example prompt.

```
Here are the functions for obtaining objects and target objects. Their
    return types are all one single point-cloud object. A desc can only
    include strings that contain shape ["rope", "sphere", "box", "mat"],
     colors like ["white", "gray", "green", "red", "blue", "black"], and
     position like ["left", "right", "top", "bottom"]:
- get(desc) # Get the point cloud of the objects with the description,
    for example, get("left white mat"). If the desc is 'all', then get
    all the objects.
- goal(desc) # Get the point cloud of the goal for the objects with the
     given name. The name can also be 'all'.
- others(desc) # Get all other point clouds except the object with the
    given name.
- others_goals(desc) # Get all other point clouds of the goals except
    the object with the given name.

Here are the functions that use constraints. Note that the obj, and
    goal can only be one single point-cloud:
- keep(cond, start=s, end=t) # Minimize the reward from time s to time
    t. It only takes one cond as an argument.
- require(cond, end=t) # Reach the condition at time t. By default end
    =1.
```

```
- similar(obj, goal) # Compare the shape earth mover distance of the
    objects to the given goal shapes.
- touch(obj) # Minimize the distance between the tool and the object.
- fix_place(obj) # Ensure that the shape of the given object(s) and its
     positions do not change. It only takes one argument.
- no_break(obj) # Do not break the objects.
- away() # check if the actuator is away from all objects
- roll() # actuators' rotation about x
- pitch() # actuator's rotation about y
- yaw() # actuator's rotation about y
- vertical() # actuator is vertical
- horizontal() # actuator is horizontal


Here are the example functions to obtain additional information:
- com(obj) # Center of the objects.
- pcd(obj) # point clouds of the objects.
- max(pcd(obj)) # max of the point cloud of the objects.
- min(pcd(obj)) # min of the point cloud of the objects.
- x(com(obj0)) # X coordinate of the points.
- x(min(pcd(obj0))) # smallest X coordinate of the boundary of the
    points.


We can compose those functions to implement various functions. Below
    are examples:
# The x coordinate of the objects
 x(com(obj0))

# Obj0 is on the left of obj1.
 lt(x(com(obj0)), x(com(obj1)))
```

```
# the front end of the Obj0 is above obj1.
 gt(y(com(frontend(obj0))), y(max(pcd(obj1))))

# Obj0 is in front of obj1.
 lt(z(com(obj0)), z(min(pcd(obj1))))

# deform the rope named by 'blue' into its goal shape
 require(similar('blue', goal('blue')))

# deform object 'blue' into its goal shape while fixing others
 require(similar('blue', goal('blue'))), keep(fix_place(others('blue'))
     )

# first do A then do B
 require(A, end=0.5), require(B, end=1.0)

# not rotate the actuator aboux x axis
 keep(roll() < 0.1)
```

Here are special function for

- stage(sample_clause, optimize_clause) # Each program may have multiple stage and each stage must have a 'sample' function and an 'optimize' function. There are at most three stages. Start a new stage if and only if we need to change the actuators or manipulate different objects or parts.
- sample(actuator_name, *args) # sample the actuator with the given name in the beginning, args denotes for the conditions or requirements of the actuator's pose.
- optimize(*args) # conditions needs to satiesfied during the

```
    manipulation.

Below are examples of the scenes and the program to solve the scenes.


Input: grasp the front end of the blue rope vertically and then deform
    into its goal pose and please do not break it. Then grasp the back
    of the mat and then pick up the mat and move it into its goal
    position.
Program:
blue_rope = get("blue rope")
stage(
    sample("gripper", grasp(frontend(blue rope)), vertical()),
    optimize(
        require(similar(blue_rope, goal('blue rope'))),
        keep(touch(blue_rope)),
        keep(no_break(blue_rope))
    )
)
mat = get("mat")
stage(
    sample("gripper", grasp(backpart(mat))),
    optimize(
        require(l2(com(get(mat)), com(goal("mat")))),
        keep(touch(backpart(mat)))
    )
)


Input: cut the mat into its goal shapes and then move the knife away.
Program:
mat = get("mat")
```

```
stage(
    sample("knife", touch(mat)),
    optimize(
        require(similar(mat, goal('mat'))),
        keep(touch(mat), end=0.5),
        requre(away())
    )
)
```

```
Input: move the object 'A' to the left of 'B' then move it to the right
   .
Program:
A = get('A')
B = get('B')
stage(
    sample("gripper", grasp(A)),
    optimize(
        require(lt(x(com(A)), x(com(B))), end=0.5),
        require(gt(x(com(A)), x(com(B))), end=1),
        keep(touch(A))
    )
)
```

```
Input: Put the board above all objects and deform them into their goal
   shapes and keep them not broken
Program:
stage(
    sample("board", gt(y("board"), y(max(pcd('all'))))),
    optimize(
        require(similar('all', goal('all'))),
        keep(touch('all')),
```

```
        keep(no_break('all')),
    )
)
```

```
Input: Use the gripper to grasp the right part of the object on the
   right and deform it into its target shape while fixing others.
Program:
left = get('left')
stage(
    sample("gripper", rightpart(left)),
    optimize(
        require(similar(left, goal('left'))),
        keep(fix_place(others('left'))),
        keep(touch(rightpart(left))),
    )
)
```

```
Input: There are two objects of different colors "red", "green". Use
   the board to touch them one by one. Deform them into their target
   shape.
Program:
stage(
    sample("board", touch("red")),
    optimize(
        keep(touch('red')),
        require(similar("red", goal("red")))
    )
)
stage(
    sample("board", touch("green")),
    optimize(
```

```
        keep(touch('green')),
        require(similar("green", goal("green")))
    )
)
```

```
Please compile the input into a program. The generated programs satisfy
    several requirements: Do not use functions not mentioned before;
   the program should be concise; only use 'touch' at most once within
   a single 'optimize' and if we need to manipulate multiple objects,
   please use touch('all').

Please include an 'optimize' for each stage. Do not use more than one
   stage if it does not need to manipulate different objects or parts.
   Here is the input to compile:
```

5.6.2 Comparing GPT3.5 and GPT4

We notice that GPT4 has a better performance compared with GPT3.5. Below we provide the results of GPT3.5 and GPT4 of example tasks shown in the main text. Generally, we have found that it is common for GPT3.5 to misinterpret the instructions, add/ignore constraints, and write incorrect commands with incorrect grammar. In contrast, GPT4 can generate pretty good results.

Although GPT4 already outperforms GPT3.5 in the translation task, we observed several failure cases in the translation process. For example, the output of GPT4 will contain objects that do not exist. In a task that lifts a sphere above a box, the output may contain an invalid sentence like require(similar('sphere', goal('sphere above box'))) where 'sphere above box' which doesn't correspond to a proper object. It may also generate a statement with a type error. For example, sometimes it may output require(similar('sphere', y(max(pcd('box'))))), which would trigger a compilation error as we can not compute the shape distance between an object

and a coordinate. Fortunately, those compilation errors could be detected during the compilation and can be resolved by resampling. However, there are also rare cases where LLM removes or adds more constraints to the instructions. For example, it may ask the solver to fix the position of an object that needs to move. In these cases, our solver may fail to find a suitable solution. Restricting the output to fit a particular syntax, for instance, using context-free grammars would resolve such problems.

Test Case 1

Grasp the front of the rope and then grasp the back of the rope. Deform the rope into its target shape.

GPT3.5: added constraints not mentioned in the language

```
rope = get("rope")
stage(
    sample("gripper", grasp(frontend(rope))),
    optimize(
        keep(touch(rope)),
        require(similar(rope, goal('rope'))),
        keep(no_break(rope))
    )
)
stage(
    sample("gripper", grasp(backpart(rope))),
    optimize(
        keep(touch(rope)),
        require(similar(rope, goal('rope'))),
        keep(no_break(rope))
    )
)
```

```
rope = get("rope")
stage(
    sample("gripper", grasp(frontend(rope))),
    optimize(
        keep(touch(frontend(rope))),
        require(similar(rope, goal("rope")), end=0.5)
    )
)
stage(
    sample("gripper", grasp(backend(rope))),
    optimize(
        keep(touch(backend(rope))),
        require(similar(rope, goal("rope")), end=1.0)
    )
)
```

Test case 2

Use the gripper to first lift up the white cube above the black wall and then move it to the target location.

GPT3.5: it creates two stages for single a stage task (not need to regrasp the object).

```
white_cube = get("white cube")
black_wall = get("black wall")
stage(
    sample("gripper", lift(white_cube)),
    optimize(
        require(gt(y(com(white_cube)), y(max(pcd(black_wall))))),
        keep(touch(white_cube)),
        keep(no_break(white_cube))
    )
```

```
)
stage(
    sample("gripper", gt(x(com(white_cube)), x(com(black_wall)))),
    optimize(
        require(l2(com(white_cube), com(goal("white cube")))),
        keep(touch(white_cube)),
        keep(no_break(white_cube))
    )
)
```

GPT4

```
white_cube = get("white cube")
black_wall = get("black wall")

stage(
    sample("gripper", grasp(white_cube)),
    optimize(
        require(gt(y(com(white_cube)), y(max(pcd(black_wall)))), end
            =0.5),
        require(l2(com(white_cube), com(goal("white cube"))), end=1),
        keep(touch(white_cube))
    )
)
```

Test case 3

Put the knife above the white object. Cut the knife down to match the target shape of the white object. Ensure that the knife does not rotate.

GPT3.5: It uses z instead y to extract the objectsheight while the latter is true. It also misinterprets the meaning to deform the knife into the goal.

```
white_obj = get("white")
stage(
    sample("knife", gt(z(com("knife")), z(max(pcd(white_obj))))),
    optimize(
        require(l2(com("knife"), com(white_obj))),
        require(similar("knife", goal("white"))),
        keep(pitch() < 0.1, yaw() < 0.1)
    )
)
```

GPT4

```
stage(
    sample("knife", above(get("white"))),
    optimize(
        require(similar(get("white"), goal("white"))),
        keep(touch("white")),
        keep(roll() < 0.1),
        keep(pitch() < 0.1),
        keep(yaw() < 0.1)
    )
)
```

Test case 4

Pick up the front part of object 'top left mat' and place it into the goal without breaking it.

GPT3.5: It adds an incorrect command to let the gripper to move away from the object in the beginning.

```
mat = get("top left mat")
stage(
    sample("gripper", grasp(frontpart(mat)), away()),
    optimize(
        require(similar(mat, goal('top left mat'))),
        keep(touch(mat)),
        keep(no_break(mat))
    )
)
```

GPT4

```
stage(
    sample("gripper", grasp(frontpart(get("top left mat")))),
    optimize(
        require(l2(com(get("top left mat")), com(goal("top left mat")))
            ),
        keep(touch(frontpart(get("top left mat")))),
        keep(no_break(get("top left mat")))
    )
)
```

5.6.3 Reinforcement Learning Baseline

We use the SAC and PPO implementation from stable-baselines3 with hyperparameters in Table 5.3. Figure 5.8 compares the optimization performance of RL algorithms and the Adam optimizer with respect to the IOU score of the ending state with different numbers of sampled episodes for the 20 single-stage tasks of five categories.

Table 5.3. Parameters for Reinforcement Learning

(a) SAC

gamma	0.95
learning rate	3×10^{-4}
buffer size	10^6
target update coef	0.005
batch size	256

(b) PPO Parameters

update steps	2048
learning rate	3×10^{-4}
entropy coef	0
value loss coef	0.5
batch size	64

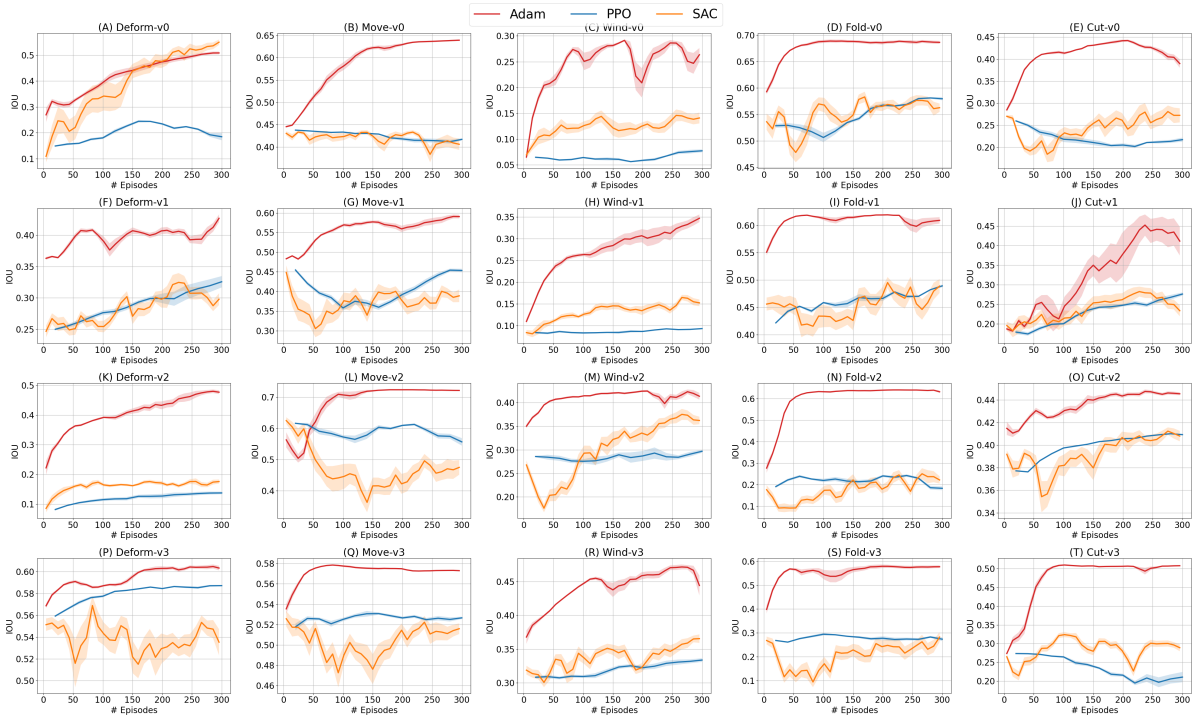


Figure 5.8. Comparison between RL and differentiable physics solver

5.6.4 Evaluation Metrics

We notice that in some tasks, a simple Intersection Over Union (IOU) score comparing shapes does not effectively explain the human intuition of task completion. For example, a solver might ”cut” two objects similar to the target shape without truly separating them, as the space between the two parts might only contain a minimal amount of space. In other circumstances, the solver may violate the requirements of no_break and break a rope into two or destroy a mat. To this end, we add checkers to check if two shapes are split and if a shape is not broken for success evaluation. For no-breaking constraint, we track particles and ensure their distance to their initial nearest neighbor does not change too much. On the other hand, we can check if two shapes are separated from each other by finding the connected components of particles.

5.7 Limitations and Conclusion

We introduce DiffVL, a method that enables non-expert users to design specifications of soft-body manipulation tasks through a combination of vision and natural language. By leveraging large language models, task descriptions are translated into machine-interpretable optimization objectives, allowing differentiable physics solvers to efficiently solve complex and long-horizon multi-stage tasks. The availability of user-friendly GUI tools enables a collection of 100 tasks inspired by real-life manipulations.

Several limitations of our approach must be acknowledged. Firstly, it relies heavily on human labor for dataset creation, and the use of large language models incurs significant computational costs. Additionally, the GUI tool for keyframe annotation requires further refinement to enhance user-friendliness and currently lacks support for rigid body simulations and reinforcement learning.

Despite these challenges, we see promising avenues for future exploration. We anticipate expanding the range of tasks by leveraging crowdsourcing, which we believe is vital to scaling up robot learning. Our initial efforts demonstrate how non-experts can contribute meaningfully

to this process, potentially inspiring further research on involving individuals from diverse backgrounds in advancing robot manipulation capabilities.

Moreover, transferring the policies developed in a differentiable simulator to real-world applications is crucial. Several components of our framework can contribute to this transition. Our vision-language representation naturally aligns with real-world tasks, while large language models can generate meaningful rewards for evaluating real-world outcomes. Furthermore, our GUI tool can facilitate effective manipulation of real-world materials, particularly in constrained tasks, serving as a direct interface between human operators and robot controllers.

By leveraging these elements, we aim to bridge the gap between simulation and reality, enhancing the applicability and effectiveness of robotic systems in diverse scenarios.

Acknowledgements

Chapter 5, in full, is a reprint of the material published in Advances in Neural Information Processing Systems 36 (2023). DiffVL: Scaling Up Soft Body Manipulation using Vision-Language Driven Differentiable Physics; Huang, Zhiao; Chen, Feng; Pu, Yewen; Lin, Chunru; Gan, Chuang; Su, Hao. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Finale

This dissertation summarizes my study in modeling-based optimization. Although it is not yet complete, I believe these efforts collectively outline a roadmap for developing a practical robotic system in the real world.

- **Accelerating Policy Learning with Simulators**: As suggested in PlasticineLab, a proper digital twin can significantly accelerate policy learning and reduce data requirements for effective robot training. By bridging real-world data with simulations, we can construct more accurate and sophisticated physical simulators for modeling environments.

- **Generative Skill Learning**: Within the simulator, generative skill learning [97] can be utilized to learn low-level manipulation policies efficiently. This approach can even help the robot discover policies that exceed human capabilities.

- **Vision-Language-Based Skill Representation**: The key to selecting the skills a robot needs to master lies in leveraging human data through a vision-language-based skill representation [95]. This representation bridges the gap between the robot's physical world and human intentions, enabling us to build a robot manipulation dataset with semantic meaning.

- **Developing High-Level Planners**: With a robust skill representation, we can develop high-level planners that mimic human reasoning and interaction with the real world [98].

Deploying these algorithms in real-world scenarios and continually refining the models with new data, we can close the loop and create a truly scalable system. However, this path is not without its challenges. Even with a well-designed GUI tool, collecting large-scale policy learning data for diverse scenes and tasks remains a significant hurdle. Additionally, developing more accurate and efficient models for real-world soft materials requires further advancements. Addressing the sim-to-real gap and achieving seamless hardware integration will also pose substantial challenges in the future. Despite these obstacles, I am confident that by addressing these challenges, we can build a more robust and adaptive robotic system capable of thriving in real-world environments.

Besides, there are some of the related and interesting topics that I believe are worth exploring in the future:

1. Schulman et al. [215] present the stochastic computation graph as a versatile framework for reinforcement learning. An intriguing direction for future research is to develop a framework that seamlessly integrates this with general graphical models in probabilistic modeling while supporting modern deep reinforcement learning techniques. A key challenge in this endeavor is how to effectively describe the optimization and probabilistic modeling problems, and subsequently, how to automatically identify the Markov structure [254] or other suboptimal structures within the problem. Additionally, extending this framework to support bilevel optimization problems [274] would be another compelling avenue for exploration.

2. The environment model acts as a proximal element in the optimization process, helping to provide more accurate gradients. However, according to the ”law of requisite variety,” there may be additional relationships between the model and the learned policy that have yet to be fully explored. Developing a unified mathematical framework where planning with the model naturally emerges would be an intriguing and valuable area of research.

3. Finding effective policies within a given environment is foundational to robotics. However,

real-world scenarios often involve dynamic and evolving environments where tasks, scenes, or even the robot itself may undergo significant changes. Adapting to these variations requires not only the ability to continually learn from new experiences but also the development of a robust organizational structure that can accommodate and integrate these changes over time. This continual learning and structural organization are key challenges that must be addressed to enable more adaptable and resilient systems. Moreover, the organization and evolution of such complex systems—where multiple interacting components must adapt and co-evolve—remains a largely unexplored area of research. Understanding and mastering this evolution could lead to significant advancements in the development of real-world robotic systems capable of thriving in unpredictable and varied environments.

Bibliography

[1] Nvidia physx sdk. https://developer.nvidia.com/physx-sdk. Accessed: 2021-08-27.

[2] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

[3] Samuel Acquaviva, Yewen Pu, Marta Kryven, Catherine Wong, Gabrielle E Ecanow, Maxwell Nye, Theodoros Sechopoulos, Michael Henry Tessler, and Joshua B Tenenbaum. Communicating natural programs to humans and machines. *arXiv preprint arXiv:2106.07824*, 2021.

[4] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[5] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics (TOG)*, 22(3):828–837, 2003.

[6] 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 J 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, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[7] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

[8] 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. *Advances in neural information processing systems*, 30, 2017.

[9] Rika Antonova, Jingyun Yang, Krishna Murthy Jatavallabhula, and Jeannette Bohg. Rethinking optimization with differentiable simulation from a global perspective. *arXiv preprint arXiv:2207.00167*, 2022.

[10] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[11] Umar Asif, Jianbin Tang, and Stefan Harrer. Graspnet: An efficient convolutional neural network for real-time grasp detection for low-powered devices. In *IJCAI*, volume 7, pages 4875–4882, 2018.

[12] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[13] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[14] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[15] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[16] James M Bern, Pol Banzet, Roi Poranne, and Stelian Coros. Trajectory optimization for cable-driven soft robot locomotion. In *Robotics: Science and Systems*, 2019.

[17] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[18] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[19] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[20] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J 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 Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023.

[21] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[22] Francesco Bullo and Andrew D. Lewis. *Geometric Control of Mechanical Systems*, volume 49 of *Texts in Applied Mathematics*. Springer Verlag, New York-Heidelberg-Berlin, 2004.

[23] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[24] Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giró-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning*, pages 1317–1327. PMLR, 2020.

[25] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *ICLR*, 2016.

[26] Lawrence Yunliang Chen, Baiyu Shi, Daniel Seita, Richard Cheng, Thomas Kollar, David Held, and Ken Goldberg. Autobag: Learning to open plastic bags and insert objects. *arXiv preprint arXiv:2210.17217*, 2022.

[27] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[28] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15084–15097. Curran Associates, Inc., 2021.

[29] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion, 2024.

[30] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.

[31] John Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International conference on machine learning*, pages 1009–1018. PMLR, 2018.

[32] Cédric Colas, Ahmed Akakzia, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Language-conditioned goal generation: a new approach to language grounding for rl. *arXiv preprint arXiv:2006.07043*, 2020.

[33] Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, 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, Homanga Bharadhwaj, 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, Jay Vakil, 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, Shubham Tulsiani, 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, Vikash Kumar, 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. *arXiv preprint arXiv:2310.08864*, 2023.

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

[35] Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. Can foundation models perform zero-shot task specification for robot manipulation? In *Learning for Dynamics and Control Conference*, pages 893–905. PMLR, 2022.

[36] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pages 7178–7189, 2018.

[37] Vin De Silva and Joshua B Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, 2004.

[38] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. *arXiv preprint arXiv:1611.01652*, 2016.

[39] Cosimo Della Santina, Robert K Katzschmann, Antonio Biechi, and Daniela Rus. Dynamic control of soft robots interacting with the environment. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 46–53. IEEE, 2018.

[40] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[41] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247*, 2023.

[42] Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint. Learning multi-object dynamics with compositional neural radiance fields. In *International Symposium of Robotics Research*, 2022.

[43] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023.

[44] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics (TOG)*, 41(2):1–21, 2021.

[45] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[46] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016.

[47] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[48] Tom Erez and Emanuel Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4914–4919. IEEE, 2012.

[49] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[50] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv:2206.08853*, 2022.

[51] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. *arXiv preprint arXiv:1910.13395*, 2019.

[52] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. In *Conference on Robot Learning*, pages 42–52. PMLR, 2020.

[53] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[54] Philipp Foehn, Davide Falanga, Naveen Kuppuswamy, Russ Tedrake, and Davide Scaramuzza. Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload. *Robotics: Science and System*, 2017.

[55] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[56] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[57] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.

[58] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Damian Mrowca, Michael Lingelbach, Aidan Curtis, Kevin T. Feigelis, Daniel M. Bear, Dan Gutfreund, David D. Cox, James J. DiCarlo, Josh H. McDermott, Joshua B. Tenenbaum, and Daniel L. K. Yamins. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.

[59] Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel L. K. Yamins, James J. DiCarlo, Josh H. McDermott, Antonio Torralba, and Joshua B. Tenenbaum. The threedworld transport challenge: A visually guided task-and-motion planning benchmark for physically realistic embodied ai. *arXiv preprint arXiv:2103.14025*, 2021.

[60] Ming Gao, Andre Pradhana Tampubolon, Chenfanfu Jiang, and Eftychios Sifakis. An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Transactions on Graphics (TOG)*, 36(6):1–12, 2017.

[61] Divyansh Garg, Skanda Vaidyanath, Kuno Kim, Jiaming Song, and Stefano Ermon. Lisa: Learning interpretable skill abstractions from language. *arXiv preprint arXiv:2203.00054*, 2022.

[62] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[63] Johan Gaume, T Gast, J Teran, A van Herwijnen, and C Jiang. Dynamic anticrack propagation in snow. *Nature communications*, 9(1):1–10, 2018.

[64] Narasimhan Gautham, Zhang Kai, Eisner Ben, Lin Xingyu, and Held David. Transparent liquid segmentation for robotic pouring. In *International Conference on Robotics and Automation (ICRA)*, 2022.

[65] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *arXiv preprint arXiv:2007.00987*, 2020.

[66] Thomas George Thuruthel, Yasmin Ansari, Egidio Falotico, and Cecilia Laschi. Control strategies for soft robotic manipulators: A survey. *Soft robotics*, 5(2):149–163, 2018.

[67] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[68] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[69] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[70] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiaing Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023.

[71] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[72] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H. Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y. Siegel, Roland Hafner, Michael Bloesch, Kristian Hartikainen, Arunkumar Byravan, Leonard Hasenclever, Yuval Tassa, Fereshteh Sadeghi, Nathan Batchelor, Federico Casarini, Stefano Saliceti, Charles Game, Neil Sreendra, Kushal Patel, Marlon Gwira, Andrea Huber, Nicole Hurley, Francesco

Nori, Raia Hadsell, and Nicolas Heess. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9(89):eadi8022, 2024.

[73] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2017.

[74] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[75] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[76] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[77] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[78] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

[79] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[80] Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR, 2019.

[81] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting. In *Proceedings of Robotics: Science and Systems*, July 2021.

[82] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019.

[83] Mikael Henaff, William F Whitney, and Yann LeCun. Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*, 2017.

[84] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017.

[85] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[86] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[87] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. Pmlr, 2018.

[88] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *International Conference on Learning Representations*, 2020.

[89] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. VisuoSpatial Foresight for Physical Sequential Fabric Manipulation. *Autonomous Robots (AURO) journal*, 2021.

[90] Taylor A Howell, Simon Le Cleac'h, J Zico Kolter, Mac Schwager, and Zachary Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 2022.

[91] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[92] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[93] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019.

[94] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[95] Zhiao Huang, Feng Chen, Yewen Pu, Chunru Lin, Hao Su, and Chuang Gan. Diffvl: scaling up soft body manipulation using vision-language driven differentiable physics. *Advances in Neural Information Processing Systems*, 36:29875–29900, 2023.

[96] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In *International Conference on Learning Representations*, 2020.

[97] Zhiao Huang, Litian Liang, Zhan Ling, Xuanlin Li, Chuang Gan, and Hao Su. Reparameterized policy learning for multimodal trajectory optimization. In *International Conference on Machine Learning*, pages 13957–13975. PMLR, 2023.

[98] Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. *Advances in Neural Information Processing Systems*, 32:1942–1952, 2019.

[99] Zixuan Huang, Xingyu Lin, and David Held. Mesh-based dynamics model with occlusion reasoning for cloth manipulation. *Robotics: Science and Systems (RSS)*, 2022.

[100] Zixuan Huang, Xingyu Lin, and David Held. Mesh-based dynamics with occlusion reasoning for cloth manipulation, 2022.

[101] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *CoRR*, abs/1807.10366, 2018.

[102] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[103] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):1–10, 2015.

[104] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, pages 1–52. 2016.

[105] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[106] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv: Arxiv-2210.03094*, 2022.

[107] Zhengyao Jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, and Yuandong Tian. Efficient planning in a compact latent action space. *arXiv preprint arXiv:2208.10291*, 2022.

[108] Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182, 2012.

[109] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[110] Lydia Kavraki, Petr Svestka, and Mark H Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, volume 1994. Unknown Publisher, 1994.

[111] Nan Rosemary Ke, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra. Modeling the long term future in model-based reinforcement learning. 2018.

[112] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[113] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[114] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[115] Tobias Klamt and Sven Behnke. Towards learning abstract representations for locomotion planning in high-dimensional state spaces. *arXiv preprint arXiv:1903.02308*, 2019.

[116] Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. Drucker-prager elastoplasticity for sand animation. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[117] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[118] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[119] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[120] Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. *Advances in Neural Information Processing Systems*, 33:8198–8210, 2020.

[121] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[122] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan. *Algorithmic and computational robotics*, pages 303–307, 2001.

[123] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[124] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[125] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[126] Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical reinforcement learning with hindsight. *arXiv preprint arXiv:1805.08180*, 2018.

[127] Chengshu Li, Fei Xia, Roberto Martin-Martin, and Silvio Savarese. Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators. In *Conference on Robot Learning*, pages 603–616. PMLR, 2020.

[128] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, Karen Liu, Jiajun Wu, and Li Fei-Fei. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.

[129] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems*, 35:31199–31212, 2022.

[130] Sizhe Li, Zhiao Huang, Tao Chen, Tao Du, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Dexdeform: Dexterous deformable object manipulation with human demonstrations and differentiable physics. *arXiv preprint arXiv:2304.03223*, 2023.

[131] Sizhe Li, Zhiao Huang, Tao Du, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Contact points discovery for soft-body manipulations with differentiable physics. In *International Conference on Learning Representations*, 2022.

[132] Sizhe Li, Zhiao Huang, Tao Du, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Contact points discovery for soft-body manipulations with differentiable physics. *arXiv preprint arXiv:2205.02835*, 2022.

[133] Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Transactions on Graphics (TOG)*, 42(1):1–20, 2022.

[134] Yinxiao Li, Yonghao Yue, Danfei Xu, Eitan Grinspun, and Peter K Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6000–6006. IEEE, 2015.

[135] Yulong Li, Shubham Agrawal, Jen-Shuo Liu, Steven K Feiner, and Shuran Song. Scene editing as teleoperation: A case study in 6dof kit assembly. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4773–4780. IEEE, 2022.

[136] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022.

[137] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. *Advances in Neural Information Processing Systems*, 30, 2017.

[138] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

[139] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[140] Junbang Liang, Ming C Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 2019.

[141] Litian Liang, Liuyu Bian, Caiwei Xiao, Jialin Zhang, Linghao Chen, Isabella Liu, Fanbo Xiang, Zhiao Huang, and Hao Su. Robo360: A 3d omnispective multi-material robotic manipulation dataset. *arXiv preprint arXiv:2312.06686*, 2023.

[142] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[143] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

[144] Xingyu Lin, Zhiao Huang, Yunzhu Li, Joshua B Tenenbaum, David Held, and Chuang Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *arXiv preprint arXiv:2203.17275*, 2022.

[145] Xingyu Lin, Carl Qi, Yunchu Zhang, Zhiao Huang, Katerina Fragkiadaki, Yunzhu Li, Chuang Gan, and David Held. Planning with spatial-temporal abstraction from point clouds for deformable object manipulation. *arXiv preprint arXiv:2210.15751*, 2022.

[146] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, 2021.

[147] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, 2021.

[148] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 2020.

[149] Fei Liu, Entong Su, Jingpei Lu, Mingen Li, and Michael C Yip. Robotic manipulation of deformable rope-like objects using differentiable compliant position-based dynamics. *IEEE Robotics and Automation Letters*, 2023.

[150] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020.

[151] Tiange Luo, Kaichun Mo, Zhiao Huang, Jiarui Xu, Siyu Hu, Liwei Wang, and Hao Su. Learning to group: A bottom-up framework for 3d part discovery in unseen categories. *arXiv preprint arXiv:2002.06478*, 2020.

[152] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

[153] Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Trans. Graph.*, 37(4), July 2018.

[154] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

[155] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315. IEEE, 2010.

[156] Jiayuan Mao, Honghua Dong, and Joseph J Lim. Universal agent for disentangling environments and tasks. 2018.

[157] Jiayuan Mao, Tomás Lozano-Pérez, Josh Tenenbaum, and Leslie Kaelbling. Pdsketch: Integrated domain programming, learning, and planning. *Advances in Neural Information Processing Systems*, 35:36972–36984, 2022.

[158] Andrew D Marchese, Russ Tedrake, and Daniela Rus. Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator. *The International Journal of Robotics Research*, 35(8):1000–1019, 2016.

[159] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018.

[160] Pietro Mazzaglia, Tim Verbelen, Bart Dhoedt, Alexandre Lacoste, and Sai Rajeswar. Choreographer: Learning and adapting skills in imagination. *arXiv preprint arXiv:2211.13350*, 2022.

[161] Aleka McAdams, Andrew Selle, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Computing the singular value decomposition of 3x3 matrices with minimal branching and elementary floating point operations. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2011.

[162] William P McCarthy, Robert D Hawkins, Haoliang Wang, Cameron Holdaway, and Judith E Fan. Learning to communicate about shared procedural abstractions. *arXiv preprint arXiv:2107.00077*, 2021.

[163] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters*, 7(4):11205–11212, 2022.

[164] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 2022.

[165] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34:24379–24391, 2021.

[166] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[167] Peter Mitrano, Dale McConachie, and Dmitry Berenson. Learning where to trust unreliable models in an unstructured world for deformable object manipulation. *Science Robotics*, 6(54):eabd8170, 2021.

[168] 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.

[169] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[170] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, 1990.

[171] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR, 2021.

[172] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. *arxiv preprint arXiv:1806.08047*, 2018.

[173] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. *under submission to NeurIPS 2021 Dataset Track*, 2021.

[174] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[175] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018.

[176] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[177] Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichiter, Silvio Savarese, and Chelsea Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022.

[178] Rhys Newbury, Jack Collins, Kerry He, Jiahe Pan, Ingmar Posner, David Howard, and Akansel Cosgun. A review of differentiable simulators. *IEEE Access*, 2024.

[179] Andrew Ng. Cs229 lecture notes. *CS229 Lecture notes*, 1(1):1–3, 2000.

[180] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *NIPS*, 2017.

[181] OpenAI. Gpt-4 technical report, 2023.

[182] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[183] Takayuki Osa, Voot Tangkaratt, and Masashi Sugiyama. Hierarchical reinforcement learning via advantage-weighted information maximization. *arXiv preprint arXiv:1901.01365*, 2019.

[184] Takayuki Osa, Voot Tangkaratt, and Masashi Sugiyama. Discovering diverse solutions in deep reinforcement learning by maximizing state–action-based mutual information. *Neural Networks*, 152:90–104, 2022.

[185] Tao Pang, HJ Terry Suh, Lujie Yang, and Russ Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. *IEEE Transactions on robotics*, 2023.

[186] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[187] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[188] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *arXiv preprint arXiv:2205.01906*, 2022.

[189] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR, 2021.

[190] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018.

[191] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep RL for model-based control. *CoRR*, abs/1802.09081, 2018.

[192] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[193] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018.

[194] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Scalable differentiable physics for learning and control. *arXiv preprint arXiv:2007.02168*, 2020.

[195] Yiling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Differentiable simulation of soft multi-body systems. *Advances in Neural Information Processing Systems*, 34:17123–17135, 2021.

[196] Yuzhe Qin, Hao Su, and Xiaolong Wang. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. *IEEE Robotics and Automation Letters*, 7(4):10873–10881, 2022.

[197] Ahmed H. Qureshi and Michael C. Yip. Deeply informed neural sampling for robot motion planning. *CoRR*, abs/1809.10252, 2018.

[198] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*, pages 234–244. Springer, 2016.

[199] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[200] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[201] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial intelligence and statistics*, pages 814–822. PMLR, 2014.

[202] Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, pages 8857–8868. PMLR, 2021.

[203] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[204] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[205] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[206] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99, 2000.

[207] Md. Sadman Sakib, David Paulius, and Yu Sun. Approximate task tree retrieval in a knowledge network for robotic cooking. *IEEE Robotics and Automation Letters*, 7(4):11492–11499, 2022.

[208] Gautam Salhotra, I-Chun Arthur Liu, Marcus Dominguez-Kuhne, and Gaurav S Sukhatme. Learning deformable object manipulation from expert demonstrations. *IEEE Robotics and Automation Letters*, 7(4):8775–8782, 2022.

[209] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018.

[210] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. *ICCV*, 2019.

[211] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320, 2015.

[212] Connor Schenck and Dieter Fox. Visual closed-loop control for pouring liquids. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2629–2636. IEEE, 2017.

[213] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. *arXiv preprint arXiv:1806.06094*, 2018.

[214] 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, 2020.

[215] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

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

[217] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8583–8592. PMLR, 13–18 Jul 2020.

[218] Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pages 8624–8633. PMLR, 2020.

[219] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. *arXiv preprint arXiv:2204.05186*, 2022.

[220] Kenneth Shaw, Shikhar Bahl, and Deepak Pathak. Videodex: Learning dexterity from internet videos. In *Conference on Robot Learning*, pages 654–665. PMLR, 2023.

[221] Haochen Shi, Huazhe Xu, Samuel Clarke, Yunzhu Li, and Jiajun Wu. Robocook: Long-horizon elasto-plastic object manipulation with diverse tools, 2023.

[222] Haochen Shi, Huazhe Xu, Zhiao Huang, Yunzhu Li, and Jiajun Wu. Robocraft: Learning to see, simulate, and shape elasto-plastic objects with graph networks. *Robotics: Science and Systems (RSS)*, 2022.

[223] Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. *arXiv preprint arXiv:2207.07560*, 2022.

[224] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022.

[225] 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.

[226] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[227] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[228] Koushil Sreenath, Nathan Michael, and Vijay Kumar. Trajectory generation and control of a quadrotor with a cable-suspended load-a differentially-flat hybrid system. In *2013 IEEE International Conference on Robotics and Automation*, pages 4888–4895. IEEE, 2013.

[229] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *CoRR*, abs/1804.00645, 2018.

[230] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

[231] Freek Stulp and Olivier Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61, 2013.

[232] Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

[233] Deborah Sulsky, Shi-Jian Zhou, and Howard L Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer physics communications*, 87(1-2):236–252, 1995.

[234] Priya Sundaresan, Jennifer Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph E Gonzalez, and Ken Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9411–9418. IEEE, 2020.

[235] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[236] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[237] Allison Tam, Neil Rabinowitz, Andrew Lampinen, Nicholas A Roy, Stephanie Chan, DJ Strouse, Jane Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations. *Advances in Neural Information Processing Systems*, 35:25377–25389, 2022.

[238] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[239] Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. *Advances in Neural Information Processing Systems*, 33:15811–15822, 2020.

[240] Sarah Tang and Vijay Kumar. Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2216–2222. IEEE, 2015.

[241] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832). http://underactuated.mit.edu/, 2020. Downloaded on 2020-09-30.

[242] Emanuel Todorov. Linearly-solvable markov decision problems. *Advances in neural information processing systems*, 19, 2006.

[243] Emanuel Todorov. General duality between optimal control and estimation. In *2008 47th IEEE Conference on Decision and Control*, pages 4286–4292. IEEE, 2008.

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

[245] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1049–1056, New York, NY, USA, 2009. Association for Computing Machinery.

[246] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[247] Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. *Robotics: Science and Systems Foundation*, 2018.

[248] James Townsend. Differentiating the singular value decomposition. Technical report, Technical Report 2016, https://j-towns. github. io/papers/svd-derivative . . . , 2016.

[249] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[250] Vainavi Viswanath, Kaushik Shivakumar, Jainil Ajmera, Mallika Parulekar, Justin Kerr, Jeffrey Ichnowski, Richard Cheng, Thomas Kollar, and Ken Goldberg. Learning to trace and untangle semi-planar knots (tusk), 2023.

[251] Vainavi Viswanath, Kaushik Shivakumar, Justin Kerr, Brijen Thananjeyan, Ellen Novoseller, Jeffrey Ichnowski, Alejandro Escontrela, Michael Laskey, Joseph E Gonzalez, and Ken Goldberg. Autonomously untangling long cables. *arXiv preprint arXiv:2207.07813*, 2022.

[252] Ruocheng Wang, Yunzhi Zhang, Jiayuan Mao, Chin-Yi Cheng, and Jiajun Wu. Translating a visual lego manual to a machine-executable plan. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVII*, pages 677–694. Springer, 2022.

[253] Yufei Wang, Zhanyi Sun, Zackory Erickson, and David Held. One policy to dress them all: Learning to dress people with diverse poses and garments, 2023.

[254] Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit assignment techniques in stochastic computation graphs. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2650–2660. PMLR, 2019.

[255] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[256] Thomas Weng, Sujay Bajracharya, Yufei Wang, Khush Agrawal, and David Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *Conference on Robot Learning*, 2021.

[257] Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021.

[258] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023.

[259] Yilin Wu, Wilson Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations, 2020.

[260] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, pages 9068–9079, 2018.

[261] Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. *arXiv preprint arXiv:2303.02346*, 2023.

[262] 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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[263] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*, 2022.

[264] Zhenjia Xu, Zhou Xian, Xingyu Lin, Cheng Chi, Zhiao Huang, Chuang Gan, and Shuran Song. Roboninja: Learning an adaptive cutting policy for multi-material objects. *arXiv preprint arXiv:2302.11553*, 2023.

[265] Mengyuan Yan, Yilin Zhu, Ning Jin, and Jeannette Bohg. Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE Robotics and Automation Letters*, 5(2):2372–2379, 2020.

[266] Kaizhi Yang, Xiaoshuai Zhang, Zhiao Huang, Xuejin Chen, Zexiang Xu, and Hao Su. Movingparts: Motion-based 3d part discovery in dynamic radiance field. *arXiv preprint arXiv:2303.05703*, 2023.

[267] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

[268] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B Tenenbaum. Clevrer: Collision events for video representation and reasoning. *arXiv preprint arXiv:1910.01442*, 2019.

[269] 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 *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[270] Tianhe Yu, Gleb Shevchuk, Dorsa Sadigh, and Chelsea Finn. Unsupervised visuomotor control through distributional planning networks. *CoRR*, abs/1902.05542, 2019.

[271] Amy Zhang, Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, and Arthur Szlam. Composable planning with attributes. *arXiv preprint arXiv:1803.00512*, 2018.

[272] Haotian Zhang, Pengchuan Zhang, Xiaowei Hu, Yen-Chun Chen, Liunian Li, Xiyang Dai, Lijuan Wang, Lu Yuan, Jenq-Neng Hwang, and Jianfeng Gao. Glipv2: Unifying localization and vision-language understanding. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 36067–36080. Curran Associates, Inc., 2022.

[273] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Noveld: A simple yet effective exploration criterion. *Advances in Neural Information Processing Systems*, 34:25217–25230, 2021.

[274] Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu. An introduction to bilevel optimization: Foundations and applications in signal processing and machine learning. *IEEE Signal Processing Magazine*, 41(1):38–59, 2024.

[275] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[276] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. *arXiv preprint arXiv:2202.05607*.

[277] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[278] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.