Learning Structured World Model for Deformable Object Manipulation

Xingyu Lin

CMU-RI-TR-22-48

July 26, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
David Held, *chair*
Abhinav Gupta
Deepak Pathak
Ken Goldberg, *Univeristy of California, Berkeley*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

To Jiayun and my parents.

Abstract

Manipulation of deformable objects challenges common assumptions in robotic manipulation, such as low-dimension state representation, known dynamics, and minimal occlusion. Deformable objects have high intrinsic state representation, complex dynamics with high degrees of freedom, and severe self-occlusion. These properties make them difficult for state estimation and planning. In this thesis, we introduce benchmarks and methods for solving various deformable object manipulation tasks, hoping to relax the commonly made assumptions and build a more robust manipulation system.

We take the approach of learning the dynamics model from data for planning. Compared to analytical models, the learned models are more flexible. We can train them to model the dynamics at different levels of detail. More specifically, we learn structured world models with spatial and temporal abstraction. At a very granular level, we can represent the physical world as atoms and predict their movement at infinitesimal time steps. Such a dynamics model is general, accurate, and without any abstraction but also expensive to compute and difficult for state estimation. With spatial abstraction, we can reason at a higher level, such as representing the world as objects and their interactions with each other. Spatial abstraction enables efficient learning, planning, and compositional generalization. With temporal abstraction, we model the dynamics to predict the future state at longer time steps or even over the span of low-level skills. We can then plan with them to solve long-horizon tasks.

In this thesis, we first introduce the first benchmark on deformable object manipulation, including manipulation of fluid, cloth, and ropes. Second, we present methods that learn dynamics models for cloth manipulation, representing the cloth as a graph for spatial abstraction. Third, we propose a framework for learning the skill dynamics model and using it for planning long-horizon sequences. We then apply the framework to manipulate elastoplastic objects with multiple tools. Finally, we show how to combine spatial and temporal abstraction to achieve long-horizon planning with compositional generalization for deformable object manipulation.

Acknowledgments

My first and deepest thanks go to my Ph.D. advisor, David Held, for his mentorship, guidance and support over the last five years. Dave sets for me an example of a dedicated researcher and educator. Till this day, I am still amazed by his passion, curiosity and persistence for research. I have got so many detailed, on-point advice from Dave in all aspects of research. The philosophy and principles I learned from Dave will always guide the next steps of my research. Dave runs an exceptionally organized lab that I have not seen anywhere else. Everything in the lab are done in a scientific and efficient way. I feel extremely lucky to be in a lab surrounded by talented, like-minded peers, with a supportive culture. Dave has been not just a great advisor, but also a role model for me in both research and life.

I would like to thank Abhinav Gupta, Deepak Pathak and Ken Goldberg for taking the time out of their busy schedules to serve on my Ph.D. thesis committee. I am constantly inspired by their works throughout my Ph.D. I am grateful for their valuable feedback, suggestions and advice on my research that helped improve my thesis. I would also like to thank Oliver Kroemer and Adithya Murali for being in my Ph.D. qualifier committee and for their value suggestions on my research.

I would like to thank my colleagues in Dave's Robots Perceiving and Doing (R-PAD) lab, including Brian Okorn, Siddharth Ancha, Pengsheng Guo, Edward Ahn, Jianren Wang, Tiancheng Jin, Olivia Xu, Harjatin Baweja, Harshit Sikchi, Thomas Weng, Wenxuan Zhou, Ben Eisner, Yufei Wang, Jianing Qian, Gautham Narasimhan, Junyu Nan, Qiao Gu, Edward Li, Gaurav Pathak, Zixuan Huang, Chuer Pan, Carl Qi, Fan Yang, Harry Zhang, Sarthak Shetty and Daniel Seita. I especially enjoy working with the master students that I have mentored: Pengsheng Guo, Harjatin Baweja, Yufei Wang, Zixuan Huang and Carl Qi. They have been a sounding-board for most of my ideas and my most reliable comrades when pushing for paper deadlines.

During my Ph.D., I spent a wonderful summer as a intern in Nvidia's Seattle Robotics Lab, mentored by Clemens Eppner, Arsalan Mousavian and Dieter Fox. It was my first time diving into working with real robotic arms and I am grateful for their guidance throughout the journey. I consider the Seattle Robotics Lab as one of the best places for doing robotics research and I thank all the people that I have met there, including: Ankur Handa, Xinke Deng, Yunzhu Li, Adithya Murali, Jacky Liang, Carlos Florensa, Yu Xiang, Nathan Ratliff, Yashraj Narang, Ajay Mandlekar, Michele Lee, Anqi Li, Carolyn Matl.

I am grateful for the research environment in and outside CMU and all my amazing friends and colleagues including Ruochi Zhang, Martin Li, Xiaofang Wang, Anqi Yang, Allan Wang, Shuqi Dai, Ben Eysenbach, Wen Sun, Lerrel Pinto, Brandon Amos, Leonid Keselman, Pratyusha Sharma, Lijun Yu, Yifeng Zhu, Yimeng Zhang, Yifan Hou, Zhilin Yang, Tiancheng Zhi, Amy Zhang, Yuandong Tian, Zhiao Huang and Chuang Gan. I have learned so much from the conversations with them about both research and life. I would like to thank all the friends that I have skied and played tennis with. It has always been a great source of happiness for me throughout my Ph.D.

I would like to give special thanks to my undergraduate mentors Yizhou Wang and Tai Sing Lee. They introduced me to the research field of computer vision and machine learning and I am forever grateful for their invaluable guidance and help along the way.

I would like to thank my fiance, Jiayun Hong, who has been there for me since the beginning of my research. Despite years of long-distance relationship, she is still the one that I share my everyday life with and I cannot overstate how grateful I am for having Jiayun by my side during this journey.

Finally, I would like to thank my parents, Gang Lin and Xiaoping Huang, for their unconditioned love and support throughout my life. It is their encouragement and support that makes my pursuit of an academic career possible.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Robotic manipulation often assumes that the objects being manipulated are rigid. With the rigidity assumption, most approaches also assume that:

1. A low-dimension state representation is known, such as the poses of the objects.
2. The dynamics is known.
3. There is minimal occlusion.

These assumptions enable efficient planning and control. However, while these assumptions are sometimes true for rigid objects, they are almost always wrong for deformable objects. Deformable objects like fluid, fabrics, or elasto-plastic objects have almost infinite degrees of freedoms and lack a low-dimension representation. As a result, it is difficult to perform state estimation and dynamics modeling for deformable objects. Finally, deformable objects often present large self-occlusion. This thesis aims to address these challenges emerged from deformable object manipulation by developing algorithms for learning the complex dynamics of deformable objects for planning. The challenges in modeling deformable objects force us to abandon the aforementioned assumptions and build a more general manipulation systems. Progress made in deformable object manipulation can in turn help with manipulating novel objects where a low-dimension object pose is not defined, and objects under occlusion.

To plan for manipulation, a dynamics model is needed to predict the future conditioned on the action. In this thesis, we parameterize the dynamics models with neural networks and learn these models from data. Such data-driven models have a few advantages compared to analytical models. First, the learned dynamics models are more flexible to the input

representation, allowing us to learn dynamics models at different levels of abstraction (Chapter 3, 4, 5), with different trade-off between accuracy and computation. Because of this, we are able to perform long-horizon planning with a learned dynamics model (Chapter 5). They are also flexible to predict dynamics from partial observation (Chapter 3). Second, neural networks are naturally differentiable and thus these differentiable dynamics models can better support system identification or become modules in a larger end-to-end system (Chapter 4, 5). Finally, an accurate analytical model is not always available, especially for deformable objects which have complex dynamics. Data-driven models allow us to learn complex dynamics of deformable objects from real world data, although we have not exploited this advantage in this thesis.

At what levels of details should we model the dynamics? In a very granular level, any physical systems can be modeled by a lot of atoms and their interactions with each other. They can also be modeled in a more abstract level, such as in the space of objects and their interactions. With a more abstract representation, the dynamics model is easier to learn and compute, as we do not need to estimate and predict the state of each atom, but is also less accurate. The choice of this spatial abstraction depends on the requirement of specific manipulation tasks. While for rigid objects, their 6D poses are often used as the level of spatial abstraction, a 6D pose is undefined for deformable objects. On the other hand, the data-driven models are flexible in a way that allow us to represent and abstract deformable objects in different ways to balance accuracy, computation and generalization of the learned dynamics models.

Orthogonal to spatial abstraction, we can model dynamics in different temporal abstraction. We can model how the atoms move at each infinitesimal time step, or at a longer time span. The latter allows us to predict the state of a physical system further into the future and thus make longer horizon plans. While typical physical simulators rely on small time steps to be numerically stable during integration, data-driven models can be trained to predict at different granularity in time. This thesis will introduce methods that allow long-horizon planning over skills with learned, temporally extended dynamics models. We term our learned dynamics models **Structured World Model**, as they incorporate spatial and temporal abstraction into data-driven dynamics models to enable a robot to learn about the physical world at different levels of details.

1.1 Thesis Organization

This thesis is organized as follows:

- **Benchmarking Deformable Object Manipulation** [Chapter 2] Prior works on deformable object manipulation often evaluate on different tasks, robot setups or simulators, making it difficult to compare different methods. This chapter introduces SoftGym, which is the first benchmark for various deforamble object manipulation tasks. Through benchmarking, we reveal the challenges for learning policies or dynamics models for manipulating deformable objects, which inspire our following works.
- **Plan with Spatial Abstraction** [Chapter 3] We start by considering the state representation for cloth manipulation tasks. Instead of modeling the cloth as a single latent vector or just the RGB pixels from the camera, we model cloth as a set of particles connected by a graph that representing different interactions among the particles. We show the benefit of this graph abstraction in learning a dynamics model that generalizes to different clothes of different shapes.
- **Plan with Temporal Abstraction** [Chapter 4] To effectively plan long-horizon manipulation sequences, we need to model the dynamics at a higher level temporally. In this chapter, we introduce a framework for planning over skills in a latent space. We learn skill abstraction modules that model the high-level skill dynamics, allowing us to solve a set of complex tasks of manipulating elasto-plastic objects with multiple tool-use skills.
- **Plan with Spatial-temporal Abstraction** [Chapter 5] Finally, in the last chapter, we combine both spatial and temporal abstraction to plan over skills for solving novel, multi-stage tasks of manipulating elasto-plastic objects involving complex tool-use skills. To support learning skill dynamics while maintaining a level of abstraction at the object level for compositional generalization, we use an object-level latent representation that first decompose the scene into entities and then separately encode each entity into a set of latent vectors. We can then perform long-horizon planning with this latent set representation.

Chapter 2

SoftGym: A Benchmark for Deformable Object Manipulation

2.1 Introduction

Robotic manipulation of deformable objects has wide application both in our daily lives, such as folding laundry and making food, and in industrial applications, such as packaging or handling cables. However, programming a robot to perform these tasks has long been a challenge in robotics due to the high dimensional state representation and complex dynamics (Maitin-Shepard et al., 2010; Miller et al., 2011; 2012).

One potential approach to enable a robot to perform these manipulation tasks is with deep reinforcement learning (RL), which has achieved many successes in recent years (Mnih et al., 2015; Vinyals et al., 2019; Schrittwieser et al., 2019; Hwangbo et al., 2019; Andrychowicz et al., 2020). Some recent works have used learning-based methods to explore the challenges of deformable object manipulation (Li et al., 2019a; Matas et al., 2018; Agrawal et al., 2016; Sermanet et al., 2018); however, these works often each evaluate on a different task variant with different simulators or robot setups, making it challenging to directly compare these approaches. There is currently no benchmark for evaluating and comparing different approaches for deformable object manipulation.

In contrast, there are a number of popular benchmarks for reinforcement learning with rigid or articulated objects (Brockman et al., 2016; Tassa et al., 2018; Fan et al., 2018).

Many of these benchmarks assume that the agent directly observes a low dimension state representation that fully describes the underlying dynamics of the environment, such as the joint angles and velocities of the robot (Brockman et al., 2016; Duan et al., 2016) or object state (Andrychowicz et al., 2020; Johannink et al., 2019). However, such low-dimensional sufficient state representations are difficult to perceive (or sometimes even define) for many deformable object tasks, such as laundry folding or dough manipulation. For deformable object manipulation, the robot must operate directly on its observations, which can include camera images and other sensors.

In this chapter, we present SoftGym, a set of open-source simulated benchmarks for manipulating deformable objects, with a standard OpenAI Gym API and Python interface for creating new environments. Currently, SoftGym includes 10 challenging environments involving manipulation of rope, cloth and fluid of variable properties, with different options for the state and action spaces. These environments highlight the difficulty in performing robot manipulation tasks in environments that have complex visual observations with partial observability and an inherently high dimensional underlying state representation for the dynamics. SoftGym provides a standardized set of environments that can be used to develop and compare new algorithms for deformable object manipulation, thus enabling fair comparisons and thereby faster progress in this domain.

We benchmark a range of algorithms on these environments assuming different observation spaces for the policy, including full knowledge of the ground-truth state of the deformable object, a low-dimension state representation, and only visual observations. Our results show that learning with visual observations leads to much worse performance compared to learning with ground-truth state observations in many deformable object manipulation tasks. The poor performance of image-based methods on these environments motivates the need for future algorithmic development in this area. We also provide an analysis to give some insight into why current methods that use visual observations might have suboptimal performance; this analysis can hopefully point the way towards better methods for deformable object manipulation. Videos and links to code can be found on our project website [1].

[1] https://sites.google.com/view/softgym

2.2 Related Works

Robotic manipulation of deformable objects has a rich history across various fields, such as folding laundry (Maitin-Shepard et al., 2010), preparing food (Bollini et al., 2013), or assistive dressing and feeding (Chen et al., 2013; Erickson et al., 2018). Early works used traditional vision algorithms to detect key features such as edges and corners (Maitin-Shepard et al., 2010; Willimon et al., 2011; Triantafyllou and Aspragathos, 2011). Motion planning is then adopted along with analytical models of the deformable objects (Saha and Isto, 2007; Rodriguez et al., 2006). However, these planning approaches often suffer from the large configuration space induced by the high degree of freedom of the deformable objects (Essahbi et al., 2012). We refer to (Sanchez et al., 2018; Khalil and Payeur, 2010) for a more detailed survey on prior methods for robot manipulation of deformable objects.

Recently, the success of deep learning has garnered increased interest in learning to solve perception and manipulation tasks in an end-to-end framework (Agrawal et al., 2016; Sermanet et al., 2018; Ebert et al., 2018; Pathak et al., 2018; Yan et al., 2020). In cloth manipulation, recent work uses demonstrations to learn an image-based policy for cloth folding and draping (Matas et al., 2018). Other works learn a pick-and-place policy to spread a towel (Seita et al., 2019; Wu et al., 2020). Due to the large number of samples required by reinforcement learning, as well as the difficulty in specifying a reward function, all these works start by training the policy in simulation and then transfer the policy to a real robot through domain randomization and dynamics randomization. However, these papers do not systematically compare different methods on a range of tasks.

Standard environments with benchmarks have played an important role in the RL community, such as the Arcade Learning environments (Bellemare et al., 2013) and the MuJoCo environments (Duan et al., 2016). A variety of new environments have been created recently to benchmark reinforcement learning algorithms (Fan et al., 2018; Osband et al., 2019; Yu et al., 2019; James et al., 2020; Lee et al., 2019). However, none of these benchmark environments incorporate deformable objects, and usually the full state of the system can be represented by a low-dimensional vector. Other recent environments built on top of the Nvidia PhysX simulator also have the ability to simulate of deformable objects (Xiang et al., 2020; Gan et al., 2021) but do not include any tasks or assets for manipulating deformable objects. As such, we believe that SoftGym would be a unique and valuable contribution to the reinforcement learning and robotics communities, by enabling new methods to be

quickly evaluated and compared to previous approaches in a standardized and reproducible set of simulated environments.

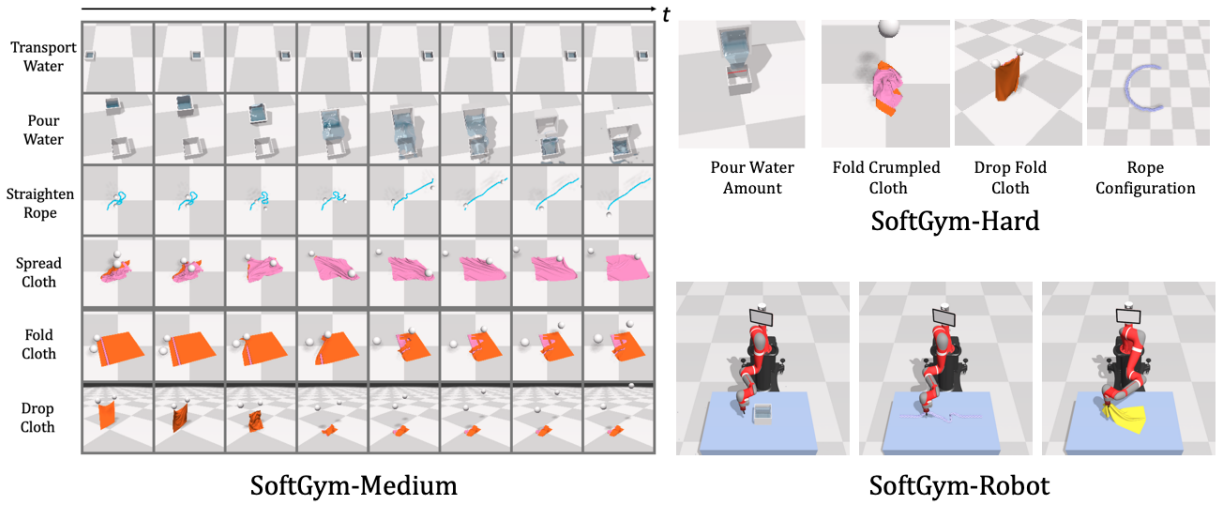


Figure 2.1: Visualizations of all tasks in SoftGym. These tasks can be used to evaluate how well an algorithm works on a variety of deformable object manipulation tasks.

2.3 Background

SoftGym builds on top of the Nvidia FleX physics simulator. Nvidia's FleX simulator models deformable objects in a particle and position based dynamical system (Müller et al., 2007; Macklin et al., 2014). Each object is represented by a set of particles and the internal constraints among these particles. Each particle p_i has at least three attributes: position x_i, velocity v_i, and inverse mass w_i. Different physical properties of the objects are characterized by the constraints. A constraint is represented in the form of $C(x_1, ..., x_n) \geq 0$ or $C(x_1, ..., x_n) = 0$, where $C(x)$ is a function of all the positions of the relevant particles. Given the current particle positions p_i and velocities v_i, FleX first computes a predicted position $\hat{p}_i = p_i + \Delta t v_i$ by integrating the velocity. The predicted positions are then projected onto the feasible set given all the constraints to obtain the new positions of the particles in the next step.

Fluids can naturally be modeled in a particle system, as detailed in Macklin and Müller (2013); specifically, a constant density constraint is applied to each particle to enforce the

incompressibility of the fluid. For each particle, the constant density constraint is based on the position of that particle, as well as the positions of its neighboring particles.

Rope is modeled as a sequence of particles, where each pair of neighbouring particles are connected by a spring. Cloth is modeled as a grid of particles. Each particle is connected to its eight neighbors by a spring, i.e. a stretching constraint. Additionally, for particles that are two-steps away from each other, a bending constraint is used to model the resistance against bending deformation. Additional constraints for modeling self-collision are applied. We refer the readers to Müller et al. (2007) for more details.

2.4 SoftGym

To advance research in reinforcement learning in complex environments with an inherently high dimensional state, we propose SoftGym. SoftGym includes a set of tasks related to manipulating deformable objects including rope, cloth, and fluids. As a result, the underlying state representation for the dynamics has a dimension ranging from hundreds to thousands, depending on the number of particles that are used.

SoftGym consists of three parts: SoftGym-Medium, SoftGym-Hard and SoftGym-Robot, visualized in Figure 2.1. SoftGym-Medium includes six tasks where we provide extensive benchmarking results. Four more challenging tasks are included in SoftGym-Hard. SoftGym-Medium and SoftGym-Hard using an abstract action space while SoftGym-Robot includes tasks with a Sawyer or Franka robot as the action space. [2] We describe the details of the action space below.

2.4.1 Action Space

We aim to decouple the challenges in learning low-level grasping skills from high-level planning. As such, we employ abstract action spaces for tasks in SoftGym-Medium and SoftGym-Hard. For rope and cloth manipulation, we use pickers, which are simplifications of a robot gripper and are modeled as spheres which can move freely in the space. A picker can be activated, in which case if it is close to any object, the particle on the object that is the closest to the picker will be attached to the picker and moves with it. More specifically,

[2] Unfortunately, we are not able to release SoftGym-Robot to the public due to license issue. However, we hope similar functions can be implemented in the new Nvidia Omniverse Kit.

the action of the agent is a vector of length $4n$, where n is the number of the pickers. For each picker, the agent outputs (d_x, d_y, d_z, p), where d_x, d_y, d_z determine the movement of the picker and p determines whether the picker is activated (picking cloth, $p \geq 0.5$) or deactivated (not picking cloth, $p < 0.5$). For fluid related tasks, we directly actuate the cup holding the fluid. This action space is designed to enable the user to focus on the challenges of high-level planning and to abstract away the low-level manipulation.

Still, in order to reflect the challenges of robotic manipulation, we also provide SoftGym-Robot, where either a Sawyer robot or a Franka robot is used to manipulate objects in the environment (Figure 2.1, bottom-right). Cartesian control of the end-effector is used.

2.4.2 Tasks

SoftGym-Medium includes six tasks (see Appendix for more details):

TransportWater Move a cup of water to a target position as fast as possible without spilling out the water. The movement of the cup is restricted in a straight line; thus the action of the agent is just a scalar $a = d_x$ that specifies the displacement of the cup. The reward of this task is the negative distance to the target position, with a penalty on the fraction of water that is spilled out.

PourWater Pour a cup of water into a target cup. The agent directly controls the position and rotation of the cup at hand. Specifically, the action space of the agent is a vector $a = (d_x, d_y, d_\theta)$, representing the displacement of the cup in the x, y dimension and its rotation around its geometric center. The reward is the fraction of water that is successfully poured into the target cup.

StraightenRope Straighten a rope starting from a random configuration. The agent controls two pickers. The reward is the negative absolute difference between the current distance of the two endpoints of the rope, and the rope length when it is straightened.

SpreadCloth Spread a crumpled cloth on the floor. The agent controls two pickers to spread the cloth. The reward for this task is the area covered by the cloth when viewed top-down.

FoldCloth Fold a piece of flattened cloth in half. The agent controls two pickers. The reward for this task is the distance of the corresponding particles between the two halves of the cloth; we also add a penalty based on the displacement of the cloth from its original position, i.e., we do not want to agent to drag the cloth too far away while folding it.

DropCloth This task begins with the agent holding two corners of a piece of cloth with two pickers in the air, and the goal is to lay the cloth flat on the floor. The action space of the agent is the same as that in SpreadCloth, with the additional constraint that the pickers cannot move too close to the floor (i.e. below a given height threshold). As such, a swinging and dropping motion is required to perform the task; dropping the cloth without swinging will result in the cloth being crumpled. The reward of this task is the mean particle-wise L_2 distance between the current cloth and a target cloth configuration flattened on the floor. SoftGym-Hard contains four more tasks:

PourWaterAmount This task is similar to PourWater but requires a specific amount of water poured into the target cup, indicated either in the state representation and marked by a red line in the visual observation.

FoldCrumpledCloth This task is similar to FoldCloth but the cloth is initially crumpled. Thus, the agent may need to spread and fold the cloth at the same time.

DropFoldCloth This task has the same initial state as DropCloth but requires the agent to fold the cloth instead of just laying it on the ground.

RopeConfiguration This task is similar to StraightenCloth but the agent needs to manipulate the rope into a specific configuration from different starting locations. Different goal configurations in the shape of letters of the alphabet can be specified. The reward is computed by finding the minimum bipartite matching distance between the current particle positions and the goal particle positions (West et al., 1996).

FleX uses a GPU for accelerating simulation. On a Nvidia 2080Ti GPU, all SoftGym tasks run about 4x faster than real time, with rendering. One million simulation steps takes 6 hours (wall-clock time) and corresponds to at least 35 hours for a real robot to collect. More details on the environments, including the variations for each task, can be found in the Appendix.

2.5 Experiments

In this section, we perform experiments with an aim to answer the following questions:

- Are SoftGym tasks challenging for current reinforcement learning algorithms?
- Is learning with state as sample efficient as learning from high-dimensional observations on SoftGym tasks?

- Are the environments realistic enough to reflect the difficulty of learning on a real deformable object dynamical system?

2.5.1 Methods Evaluated

We benchmark a few representative policy search algorithms on the tasks in SoftGym. We group these algorithms into categories which make different assumptions regarding knowledge about the underlying dynamics or position of particles in the environments. These algorithms allow us to analyze different aspects of the challenges in learning to manipulate deformable objects. We use the ground-truth reward function for all the baselines, although such rewards are not readily available outside of the simulation environment; computing rewards from high-dimensional observations is an additional challenge that is outside the scope of this section. We refer to Nair et al. (2018); Srinivas et al. (2018); Florensa et al. (2019); Lin et al. (2019b) for recent works towards this challenge.

2.5.1.1 Dynamics Oracle

The Dynamics Oracle has access to the ground-truth positions and velocities of all particles in the deformable objects, as well as access to the ground-truth dynamics model. This information is only accessible in simulation. Given this information, we can use gradient free optimization to maximize the return. In this category we benchmark the cross entropy method (CEM) (Rubinstein, 1999) as a representative random shooting algorithm. CEM optimizes over action sequences to find the trajectory with the highest return, given the action sequence and the ground-truth dynamics model. Because the ground-truth dynamics model is used, no training is required (i.e. no parameters are learned) for this method. We use model predictive control (MPC) for executing the planned trajectories. This baseline shows what can be achieved from a trajectory optimization approach given ground-truth particle positions, velocities, and dynamics; we would expect that methods that only have access to visual observations would have worse performance. However, the performance of the Dynamics Oracle may still be limited due to the exploration strategy employed by CEM.

2.5.1.2 State Oracle

Many robotic systems follow the paradigm of first performing state estimation and then using the estimated state as input to a policy. Deformable objects present a unique challenge for manipulation where the dynamical system has a high dimensional state representation, i.e. the position of all particles in the deformable objects. For such high dimensional systems, state estimation becomes harder; furthermore, even assuming perfect state estimation, the high dimensional state space is challenging for any reinforcement learning agent. We explore two state-based methods to explore these challenges.

Full State Oracle This method has access to the ground-truth positions of all of the particles in the target object as well as any proprioception information of the robot or the picker, but it does not have access to the ground-truth dynamics. We use these positions as input to a policy trained using SAC (Haarnoja et al., 2018); we use the standard multi-layer perceptron (MLP) as the architecture for the agent. As the observational dimension can vary, e.g. cloths with different sizes can have different numbers of particles, we take the maximum number of particles as the fixed input dimension and pad the observation with 0 when it has fewer dimensions. Generally, we expect this baseline to perform poorly, since concatenating the positions of all particles in a single vector as input forces the network to learn the geometric structure of the particles, which adds significantly to the complexity of the learning problem. Given additional task-specific information about the connectivity among the particles, alternative architectures such as graph neural networks may be better at capturing the structure of the particles (Li et al., 2019b).

Reduced State Oracle To avoid the challenges of RL from high-dimensional state spaces, this method uses a hand-defined reduced set of the full state as input to the policy. This baseline uses the same SAC algorithm as the Full State Oracle to train the policy. Estimating this reduced state from a high-dimensional observation of a deformable object, such as from an image of crumpled cloth, may be challenging; for this oracle baseline, we assume that such a reduced state is perfectly estimated. Thus, this baseline provides an upper bound on the performance of a real system, assuming that the reduced state captures all of the relevant information needed for the task. Unlike the Dynamics Oracle, this baseline does not assume access to the ground-truth system dynamics.

For all of the cloth related environments, the reduced state is the positions of the four corners of the cloth. For the StraightenRope environment, we pick 10 evenly-spaced

keypoints on the rope, including the two end points, and use the positions of these key points as the reduced state representation. For TransportWater, the reduced state is the size (width, length, height) of the cup, the target cup position, and the initial height of the water in the cup. For PourWater, the reduced state is the sizes of both cups, the position of the target cup, the position and rotation of the controlled cup, and the initial height of the water in the cup. For any environment with pickers, the positions of the pickers are also included as part of the reduced state. We note that the set of reduced state we picked may not be sufficient for performing all of the manipulation tasks. For example, knowing the positions of the four corners of a crumpled cloth is not sufficient to infer the full configuration of the cloth, so some information is lost in this reduced state representation.

2.5.1.3 Image Based Observations

We also evaluate state-of-the-art RL algorithms that directly operate on high dimensional observations. It is important to evaluate methods that use high dimensional observations as input, since it cannot be assumed that a low dimensional state representation (such as that used by the Reduced State Oracle) can always be accurately inferred.

Recent works (Laskin et al., 2020b; Kostrikov et al., 2020; Laskin et al., 2020a) show evidence that the gap between image-based RL and state-based RL can be closed on a range of tasks with the data augmentation in reinforcement learning. Among these, we benchmark CURL-SAC (Laskin et al., 2020b), which uses a model-free approach with a contrastive loss among randomly cropped images, and DrQ (Kostrikov et al., 2020), which applies data augmentation and regularization to standard RL. We also evaluate PlaNet (Hafner et al., 2019), which learns a latent state space dynamics model for planning. For SpreadCloth and FoldCloth, we additionally benchmark Wu et al. (2020), which learns a pick-and-place policy with model-free RL. For this method, we use only one picker and 20 pick-and-place during evaluation.

2.5.2 Benchmarking Results

Experimental Setup For each task, we compute a lower bound and upper bound on performance so that we can more easily analyze the performance of each method (see Appendix for details). The lower bound is obtained from a policy that always does nothing. Using these bounds, the performance of each method can then be normalized into $[0, 1]$,

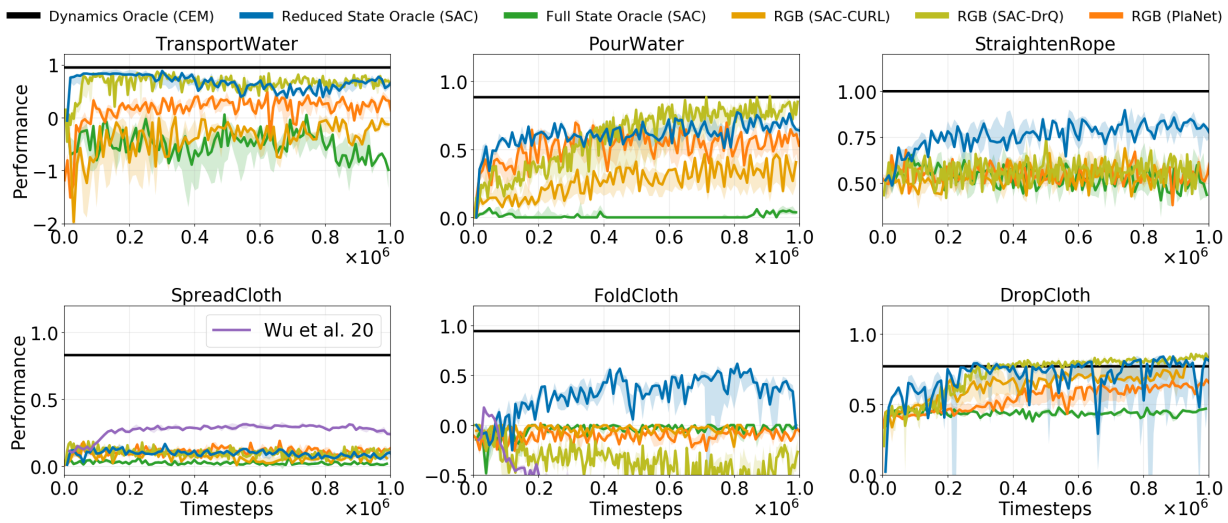


Figure 2.2: Normalized performance at the last time step of the episode of all the algorithms on the evaluation set. The x-axis shows the number of training time steps.

although the performance of a policy that performs worse than doing nothing can drop below 0. We run each algorithm for 5 random seeds and plot the median of the normalized performance. Any shaded area in the plots denotes the 25 and 75 percentile. In each task, we pre-sample 1000 variations of the environment. We then separate these task variations into a set of training tasks with 800 variations and a set of evaluation tasks of 200 variations. For CEM, no parameters are trained, so we modify this procedure: instead, we randomly sample 10 task variations from the evaluation set and compute the average performance across the variations. All methods are trained for 10^6 time steps. Please refer to the Appendix for more details of the algorithms and training procedure. Most of the experiments are run on an Nvidia 2080Ti GPU, with 4 virtual CPUs and 40G RAM.

2.5.2.1 Benchmarking results on SoftGym-Medium

A summary of the final normalized performance of all baselines on the evaluation set is shown in Figure 2.2. As expected, the Dynamics Oracle performs the best and is able to solve most of the tasks. As the dynamics and ground-truth position of all particles are usually unknown in the real world, this method serves as an upper bound on the performance.

The Reduced State Oracle performs well on tasks where the reduced state captures

the task relevant information, such as StraightenRope, TransportWater, FoldCloth, and performs poorly on the tasks which may require more information beyond the reduced state, such as in SpreadCloth, where the positions of the four cloth corners are not sufficient to reason about the configuration of the cloth. We note that the reduced states can be hard to obtain in the real world, such as in TransportWater, where the reduced state includes the amount of the water in the cup.

More interestingly, we also examine the performance of methods that assume that the agent only has access to image observations. A robot in the real world will not have access to ground-truth state information and must use these high dimensional observations as inputs to its policy. We observe that the performance of image-based reinforcement learning (PlaNet, SAC-CURL, or SAC-DrQ) is far below the optimal performance on many tasks. This is especially true for StraightenRope, SpreadCloth, and FoldCloth, and the learning curves for these tasks seem to imply that even with more training time, performance would still not improve. These methods also have a performance far below the upper bound of 1 on the other tasks (TransportWater, PourWater, DropCloth). Thus, this evaluation points to a clear need for new methods development for image-based robot manipulation of deformable objects. Compared to the reduced state oracle, image based methods have much worse performance in certain tasks such as FoldCloth or StraightenRope, indicating that there is still a gap between learning from high dimensional observation and learning from state representation.

The Full State Oracle, which uses the position of all particles in the object as input to a policy, performs poorly on all tasks. This further demonstrates the challenges for current RL methods in learning to manipulate deformable objects which have a variable size and high dimensional state representation.

For the SpreadCloth task, we additionally compare to previous work (Wu et al., 2020) that learns a model-free agent for spreading the cloth from image observation. A pick-and-place action space is used here. During training, for collecting data, a random location on the cloth is selected based on a segmentation map and the agent learns to select a place location. The picker will then move to the pick location above the cloth, pick up the cloth, move to the place location and then drop the cloth. In Figure 2.2, we show the final performance of this method with 20 pick-and-place steps for each episode. While it outperforms the rest of the baselines due to the use of the segmentation map and a better action space for exploration, the result shows that there still exists a large room for

improvement. On the other hand, this method does not perform very well on the FoldCloth task.

2.5.2.2 Difficult Future Prediction

Why is learning with deformable objects challenging? Since PlaNet (Hafner et al., 2019) learns an autoencoder and a dynamics model, we can visualize the future predicted observations of a trained PlaNet model. We input to the trained model the current frame and the planned action sequence and visualize the open-loop prediction of the future observation. Figure 2.3 shows that PlaNet fails to predict the spilled water or the shape of the cloth in deformable manipulation tasks. This provides evidence that since deformable objects have complex visual observations and dynamics, learning their dynamics is difficult.

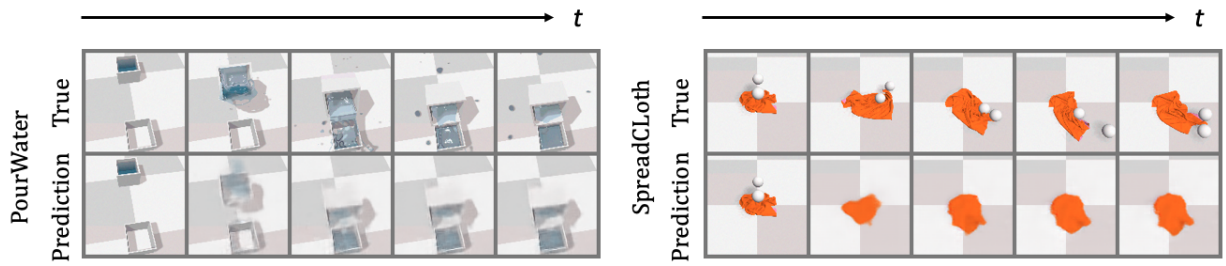


Figure 2.3: Bottom row: Open-loop prediction of PlaNet. Given an initial set of five frames, PlaNet predicts the following 30 frames. Here we show the last observed frame in the first column and four evenly spaced key frames out of the 30 predicted frames in the last four columns. Top row: Ground-truth future observations.

2.5.2.3 Reality Gap

Do SoftGym environments reflect the challenges in real world manipulation of deformable objects? Here we take the cloth environments as an example and show that both our cloth modeling and the picker abstraction can be transferred to the real world. We set up a real world cloth manipulation environment with a Sawyer robot with a Weiss gripper, as shown in Figure 2.4. We perform a series of pick and place actions both in simulation and on the real robot. We can see that the simulated cloth shows similar behaviour to the real one. This demonstration suggests that the simulation environment can reflect the complex dynamics in the real world and that algorithmic improvements of methods developed in

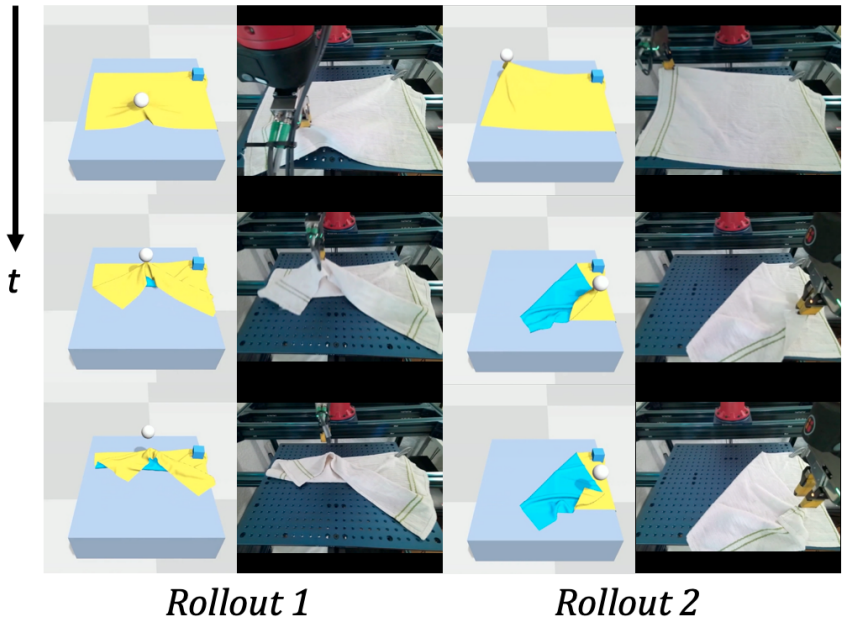


Figure 2.4: Two pick-and-place rollouts both in simulation and in the real world for a cloth manipulation task. For each rollout, the left column shows the simulation; the right shows the real world.

SoftGym are likely to correspond to improvements for methods trained in the real world; however, direct sim2real transfer of learned policies is still expected to present a challenge.

2.6 Conclusion

In this chapter, we present SoftGym, a set of benchmark environments for deformable object manipulation. We show that manipulating deformable objects presents great challenges in learning from high dimensional observations and complex dynamics. We believe that our benchmark presents challenges to current reinforcement learning algorithms; further, our benchmark should help to ease the comparison of different approaches for deformable object manipulation, which should assist with algorithmic development in this area. We are excited to see a series of amazing works in deformable object manipulation building on top of SoftGym shortly after its release (Ha and Song, 2021; Xu et al., 2022; Ma et al., 2022). In the next chapter, we will also use SoftGym for learning cloth manipulation skills.

Chapter 3

Planning with Spatial Abstraction for Cloth Smoothing

3.1 Introduction

Robotic manipulation of cloth has wide applications across both industrial and domestic tasks such as laundry folding and bed making. However, cloth manipulation remains challenging for robotics due to the complex cloth dynamics. Further, like most deformable objects, cloth cannot be easily described by low-dimensional state representations when placed in arbitrary configurations. Self-occlusions make state estimation especially difficult when the cloth is crumpled.

One approach to cloth manipulation explored by previous work, which we also adopt, is to learn a cloth dynamics model and then use the model for planning to determine the robot actions. However, given that a crumpled cloth has many self-occlusions and complex dynamics, it is unclear how to choose the appropriate state representation. One possible state representation is to use a mesh model of the entire cloth (Jiménez and Torras, 2020). However, fitting a full mesh model to an arbitrary crumpled cloth configuration is difficult. Recent work have approached fabric manipulation by either compressing the cloth representation into a fixed-size latent vector (Yan et al., 2020; Wu et al., 2020; Matas et al., 2018) or directly learning a visual dynamics model in pixel space (Hoque et al., 2020). However, these representations do not enforce any inductive bias of the cloth physics,

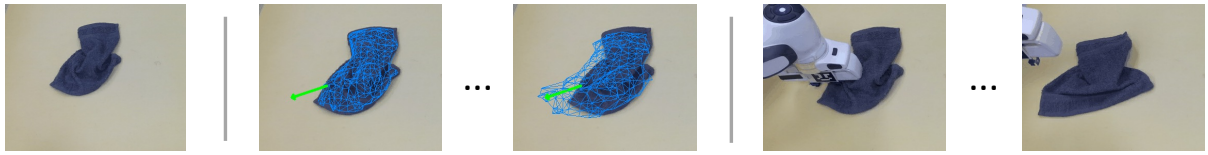


Figure 3.1: Cloth smoothing by planning using a dynamics model with a visible connectivity graph.

leading to suboptimal performance and generalization.

In contrast to a pixel-based or latent dynamics model, particle-based models have recently been shown to be able to learn dynamics for fluid and plastics (Li et al., 2019a; Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021). A particle-based dynamics representation has the following benefits: first, it captures the inductive bias of the underlying physics, since real-world objects are composed of underlying atoms that can be modeled on the micro-level by particles. Second, we can incorporate inductive bias by directly applying the effect of the robot gripper on the particle being grasped (though the effect on the other particles must still be inferred). Last, particle-based models are invariant to visual features such as object colors or patterns. As such, in this section we aim to learn a particle-based dynamics model for cloth. However, the challenges in applying the particle-based model to cloth are that we cannot directly observe the underlying particles composing the cloth nor their mesh connections. The problem is made even more challenging due to the partial observability of the cloth from self-occlusions when it is in a crumpled configuration.

Our insight into this problem is that, rather than fitting a mesh model to the observation, we should learn the *visible connectivity dynamics (VCD):* a dynamics model based on the connectivity structure of the visible portion of the cloth. To do so, we first learn to estimate the *visible connectivity graph*: we estimate which points in the point cloud observation are connected in the underlying cloth mesh. Estimating the mesh connectivity of the observation is a simplification of the problem of fitting a single full mesh model of the entire cloth to the observation; however, it is significantly easier to learn, since we do not need to find a globally consistent explanation of the observation which requires reasoning about occlusion; to estimate the mesh connectivity of the observation, we only need to consider the visible local cloth structure.

The downside of inferring the visible connectivity graph is that it cannot be used with

a physics simulator during the execution of the task, since it does not capture the occluded part of the object. We overcome this limitation by learning a dynamics model directly on the inferred visible connectivity graph (see Figure 3.1). Unlike a physics simulator, the learned dynamics model can be trained to be robust to the partial observability of the cloth.

In this chapter, we focus on the task of smoothing a piece of cloth from a crumpled configuration. We propose a method that infers the observable particles and their connections from the point cloud, learns a visible connectivity dynamics model for the observable portion of the cloth, and uses it for planning to smooth the cloth. We show that for smoothing, planning with a visible connectivity dynamics model greatly outperforms state-of-the-art model-based and model-free reinforcement learning methods that use a fixed-size latent vector representation or learn a pixel-based visual dynamics model. Furthermore, we demonstrate zero-shot sim-to-real transfer where we deploy the model trained in simulation on a Franka arm and show that the learned model can successfully smooth cloths of different materials, geometries, and colors from crumpled configurations. Videos and links to code can be found on our project website [1].

3.2 Related Work

Vision-based Cloth Manipulation: Some papers on cloth manipulation assume that the cloth is already lying flat on the table (Miller et al., 2011; Stria et al., 2014a;b). If the cloth starts in an unknown configuration, then one approach is to perform a sequence of actions that are designed to move the cloth into a set of known configurations from which perception can be performed more easily (Cusumano-Towner et al., 2011; Maitin-Shepard et al., 2010; Triantafyllou and Aspragathos, 2011). For example, the robot might first grasp the cloth by an arbitrary point and raise it into the air; it can then detect the lowest point, either while the cloth is held in the air (Cusumano-Towner et al., 2011; Osawa et al., 2007; Kita and Kita, 2002; Kita et al., 2004; 2009; Mariolis et al., 2015) or after throwing the cloth on the table (Triantafyllou and Aspragathos, 2011). By constraining the cloth to this configuration set, the task of perceiving the cloth or fitting a mesh model (Jiménez and Torras, 2020) is greatly simplified. However, these funneling actions are usually scripted and are not generalizable to different cloth shapes or configurations. In contrast, our work

[1] https://sites.google.com/view/vcd-cloth

aims to enable a robot to interact with cloth from arbitrary configurations and shapes.

Other early works designed vision systems for detecting cloth features that can be used for downstream tasks, such as a Harris Corner Detector (Willimon et al., 2011) or a wrinkle-detector (Sun et al., 2013). More examples of such approaches are described in (Jiménez and Torras, 2020). However, these approaches require a task-specific manual design of vision features and are typically not robust to different variations of the cloth configuration.

Policy Learning for Cloth Manipulation: Recently, there have been a number of learning based approaches to cloth folding and smoothing. One approach is to learn a policy to achieve a given manipulation task. Some papers approach this using learning from demonstration. The demonstrations can be obtained using a heuristic expert (Seita et al., 2020) or a scripted sequence of actions based on cloth descriptors (Ganapathi et al., 2021). Another approach to policy learning is model-free reinforcement learning (RL), which has been applied to cloth manipulation (Matas et al., 2018; Wu et al., 2020; Lee et al., 2020b). However, policy learning approaches often lack the ability to generalize to novel situations; this is especially problematic for cloth manipulation in which the cloth can be in a wide variety of crumpled configurations. We compare our method to a state-of-the-art policy learning approach (Wu et al., 2020) and show greatly improved performance.

Model-based RL for Cloth Manipulation: Model-based RL methods learn a dynamics model and then use it for planning. Model-based reinforcement learning methods have many benefits such as sample efficiency, interpretability, and generalizability to multiple tasks. Previous works have tried to learn a pixel-based dynamics model that directly predict the future cloth images after an action is applied (Ebert et al., 2018; Hoque et al., 2020). However, learning a visual model for image prediction is difficult and the predicted images are usually blurry, unable to capture the details of the cloth. Another approach is to represent the cloth with a fixed-size latent vector representation (Yan et al., 2020) and to plan in that latent space. However, cloth has an intrinsic high dimension state representation; thus, such compressed representations typically lose the fine-grained details of their environment and are unsuitable for capturing the low-level details of the cloth's shape, such as folds or wrinkles, which can be important for folding or other manipulation tasks. Our method also falls into the model-based RL category; unlike previous works, we learn a particle based dynamics model (Li et al., 2019a; Sanchez-Gonzalez et al., 2020), which can better capture the cloth dynamics due to the inductive bias of the particle

representation. Additionally, the particle representation is invariant to visual features and enables easier sim-to-real transfer.

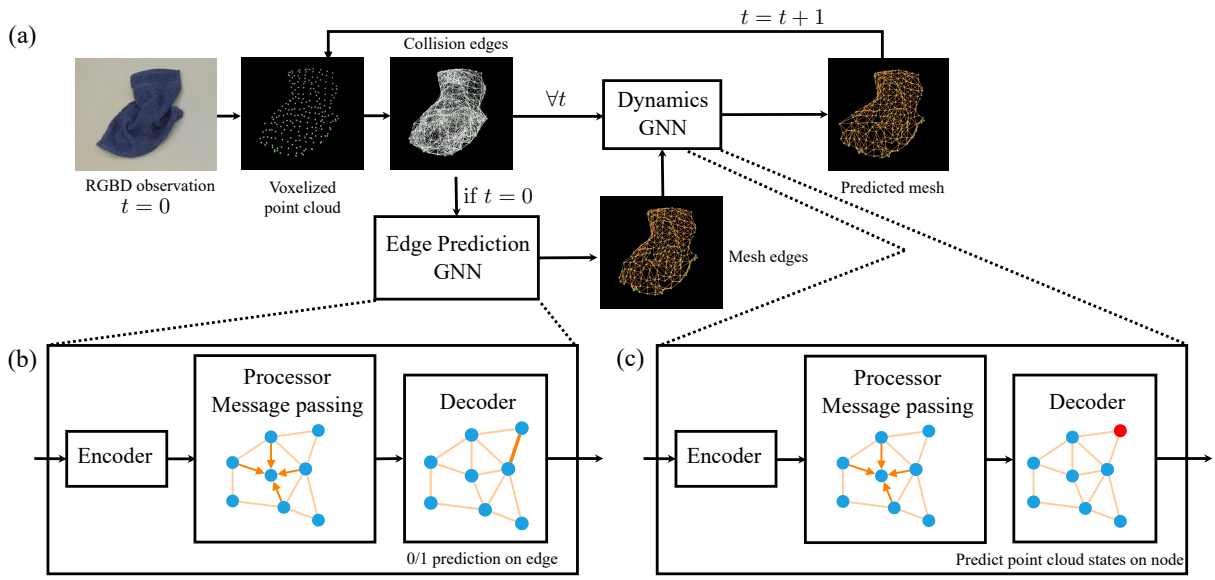


Figure 3.2: (a) Overview of our visible connectivity dynamics model. It takes in the voxelized point cloud, constructs the mesh and predicts the dynamics for the point cloud. (b) Architecture for the edge prediction GNN which takes in the point cloud connected by the collision edges and predicts for each collision edge whether it is a mesh edge. (c) Architecture for the dynamics GNN which takes in the point cloud connected by both the collision edges and the mesh edges and predict the acceleration of each point in the point cloud.

3.3 Method

An overview of our method, VCD (Visible Connectivity Dynamics), can be found in Figure 3.2. We represent the cloth using a Visible Connectivity Graph, in which we infer the collision and mesh edges between points of a partial point cloud. Next, we learn a dynamics model over this graph, and finally we use this dynamics model for planning robot actions.

3.3.1 Graph Representation of Cloth Dynamics

We represent the state of a cloth with a graph $\langle V, E \rangle$. The nodes $V = \{v_i\}_{i=1...N}$ represent the particles that compose the cloth, where $v_i = (x_i, \dot{x}_i)$ denotes the particle's current position and velocity, respectively. There are two types of edges E in the graph, representing two types of interactions between the particles: mesh edges and collision edges. The mesh edges, E^M, represent the connections among the particles on the underlying cloth mesh. The mesh connectivity is determined by the structure of the cloth and does not change throughout time. Each edge $e_{ij} = (v_i, v_j) \in E^M$ connects nodes v_i to v_j and models the mesh connection between them. The other type of edges are collision edges, E^C, which model the collision dynamics among two particles that are nearby in space. These can be different from the mesh edges due to the folded configuration of the cloth, which can bring two particles close to each other even if they are not connected by a mesh edge. Unlike the mesh edges which stay the same throughout time, these collision edges are dynamically constructed at each time step based on the following criteria:

$$E_t^C = \{e_{ij} \big| \ ||x_{i,t} - x_{j,t}||_2 < R\}, \tag{3.1}$$

where R is a distance threshold and $x_{i,t}, x_{j,t}$ are the positions of particles i, j at time step t. Throughout this section, we use the subscript t to denotes the state of a variable at time step t if the variable changes with time. Additionally, we assume that $E^M \subset E^C$, since a mesh edge connects nodes that are close to each other and hence should also satisfy Eqn. 3.1.

3.3.2 Inferring Visible Connectivity from a Partial Point Cloud

In the real world, we observe the cloth in the form of a partial point cloud. In this case, we represent the nodes of the graph using the partial point cloud and infer the connectivity among these observed points. We denote the raw point cloud observation as $P_{raw} = \{x_i\}_{i=1..N_{raw}}$, where x_i is the position of each point and N_{raw} is the number of points. We first pre-process the point cloud by filtering it with a voxel grid filter: we overlay a 3d voxel grid over the observed point cloud and then take the centroid of the points inside each voxel to obtain a voxelized point cloud $P = \{x_i\}_{i=1,...,N_p}$. This preprocessing step is done both in simulation training and in the real world, which makes our method agnostic to the density of the observed point cloud and more robust during sim2real transfer.

We create a graph node v_i for each point x_i in the voxelized point cloud P. The collision edges are then inferred by applying the criterion from Eqn. 3.1. However, inferring the mesh edges is less straightforward, since in the real world we cannot directly perceive the underlying cloth mesh connectivity. To overcome this challenge, we use a graph neural network (GNN) (Battaglia et al., 2018) to infer the mesh edges from the voxelized point cloud. Given the positions of the points in P, we first construct a graph $\langle P, E^C \rangle$ with only the collision edges based on Eqn. 3.1. As we assume $E^M \subset E^C$, we then train a classifier, which is a GNN, to estimate whether each collision edge $e \in E^C$ is also a mesh edge. We denote this edge GNN as G_{edge}. The edge GNN takes as input the graph $\langle P, E^C \rangle$, propagates information along the graph edges in a latent vector space, and finally decodes the latent vectors into a binary prediction for each edge $e \in E^C$ (predicting whether the edge is also a mesh edge). For the edge GNN, we use the network architecture in previous work (Sanchez-Gonzalez et al., 2020) (referred to as GNS). See Appendix A.1 for the detailed architecture. The edge GNN is trained in simulation, where we obtain labels for the mesh edges based on the ground-truth mesh of the simulated cloth. After training, it can then be deployed in the real world to infer the mesh edges from the point cloud. We defer the description of how we obtain the ground-truth mesh labels in Sec. 3.3.5.

3.3.3 Modeling Visible Connectivity Dynamics with a GNN

In order to predict the effect of a robot's action on the cloth, we must model the cloth dynamics. While there exists various physics simulators that support simulation of cloth dynamics (Coumans and Bai, 2016–2019; Lin et al., 2020; Narain et al., 2012), applying these simulators for a real cloth is still challenging due to two difficulties: first, only a partial point cloud of a crumpled cloth is observed in the real world, usually with many self-occlusions. Second, the estimated mesh edges from Sec. 3.3.2 may not all be accurate. To handle these challenges, we learn a dynamics model based on the voxelized partial point cloud and its inferred visible connectivity (Sec. 3.3.2). Formally, given the cloth graph $G_t = \langle V, E \rangle$, a dynamics GNN G_{dyn} predicts the particle accelerations in the next time step, which can then be integrated to update the particle positions and velocities. Here, V refers to the voxelized point cloud, and E refers to inferred visible connectivity that includes both the predicted mesh edges E^M as well as the collision edges. Our dynamics GNN G_{dyn} uses the similar GNS architecture as the G_{edge}. It takes a cloth mesh as input

with state information on each node, propagates the information along the graph edges in a latent vector space, and finally decodes the latent vectors into the predicted acceleration on each node. See Appendix A.1 for the detailed architecture of the GNN.

3.3.4 Planning with Pick-and-place Actions

We plan in a high-level, pick-and-place action space over the VCD model. For each action $a = \{x_{pick}, x_{place}\}$, the gripper grasps the cloth at x_{pick}, moves to x_{place}, and then drops the cloth. As the GNN dynamics model is only trained to predict the changes of the particle states in small time intervals in order to accurately model the interactions among particles, we decompose each high-level action into a sequence of low-level movements, where each low-level movement is a small delta movement of the gripper and can be achieved in a short time. Specifically, we generate a sequence of small delta movements $\Delta x_1, ..., \Delta x_H$ from the high-level action, where $x_{pick} + \sum_{i=1}^{H} \Delta x_i = x_{place}$. Each delta movement Δx_i moves the gripper a small distance along the pick-and-place direction and the motion can be predicted by the dynamics GNN in a single step. When the gripper is grasping the cloth, we denote the picked point as u. We assume that the picked point is rigidly attached to the gripper; thus, when considering the effect of the tth low-level movement of the robot gripper, we modify the graph by directly setting the picked point u's position $x_{u,t} = x_{pick} + \sum_{i=1}^{t} \Delta x_i$ and velocity $\dot{x}_{u,t} = \Delta x_i / \Delta t$, where Δt is the time for one low-level movement step. The dynamics GNN will then propagate the effect of the action along the graph when predicting future states. For the initial steps where the historic velocities are not available, we pad them with zeros for input to the dynamics GNN. If no point is picked, e.g., after the gripper releases the picked point, then the dynamics model is rolled out without manually setting any particle state.

Our goal is to smooth a piece of cloth from a crumpled configuration. To compute the reward r based on either the observed or the predicted point cloud, we treat each point in the point cloud as a sphere with radius R and compute the covered area of these spheres when projected onto the ground plane. Due to computational limitations, we greedily optimize this reward over the predicted states of the point cloud after a one-step high-level pick-and-place action rather than optimizing over a sequence of pick-and-place actions. Given the current voxelized point cloud of a crumpled cloth P, we first estimate the mesh edges using the edge predictor $E^M = G_{edge}(\langle P, E^C \rangle)$. We keep the mesh edges

fixed throughout the rollout of a pick-and-place trajectory since the structure of the cloth is fixed. In theory, it could be helpful to update the mesh edges based on the newly observed point cloud at each low-level step, but this is challenging due to the heavy occlusion from the robot's arm during the execution of a pick-and-place action. After the execution of each pick-and-place action, new particles may be revealed and we update the mesh edges when re-planning the next action. The pseudocode of the planning procedure can be found in the appendix.

3.3.5 Training in Simulation

The simulator we use for training is Nvidia Flex, a particle-based simulator with position-based dynamics (Müller et al., 2007; Macklin et al., 2014), wrapped in SoftGym (Lin et al., 2020). In Flex, a cloth is modeled as a grid of particles, with spring connections between particles to model the bending and stretching constraints.

One challenge that we must address is that the points in the observed partial point cloud do not directly correspond to the underlying grid of particles in the cloth simulator. This presents a challenge for obtaining the ground-truth labels used for training the dynamics GNN and the edge GNN, including the acceleration for each point in the observed point cloud and the mesh edges among them. To address this issue, we perform bipartite graph matching to match each point in the voxelized point cloud to a simulated particle by minimizing the Euclidean distance between the matched pairs. Details about the matching can be found in the appendix. After we get the mapping from the points to the simulator particles, the ground-truth acceleration of each point is simply assigned to be the acceleration of its mapped particle, which is used for training the dynamics GNN. For training the edge GNN, a collision edge is assumed to be a mesh edge if the mapped simulation particles of the edge's both end points are connected by a spring in the simulator.

3.3.6 Shape Specific Training

When a shape prior of the cloth is known, we further introduce two techniques to leverage such information to improve the performance. The first is graph-based privileged imitation learning, which transfers features from a teacher model trained on the full cloth to a student model trained on the partial cloth observation, to better allow our model to reason about occlusions. This idea is related to other recent work which trains a student with partial

observations to imitate a teacher with full-state information (Chen et al., 2020; Lee et al., 2020a; Warrington et al., 2020). Specifically, we first train a privileged teacher dynamics model with ground-truth information of the particle state of the full cloth, i.e., it takes as input all particles (including the occluded particles). Next, we train the student model, which takes a partial point cloud as input. The student is trained to imitate the features of the corresponding nodes in the teacher (Lee et al., 2020a). Graph imitation offers direct supervision on the intermediate features and enables the student model to implicitly reason about the occluded part of the cloth, by imitating the features from the teacher which has full state information. This approach is shape-specific, since the occluded parts of the cloth can only be inferred if the cloth shape is known. The second shape-specific approach that we introduce is adding an auxiliary task of reward prediction (Jaderberg et al., 2016), where the reward is the covered area of the cloth after taking the current action. Details on our shape specific training methods can be found in the appendix. Since both methods are shape-specific, we only use them when we train and test on the same (or similar) cloth shape; on the other hand, we remove them when we want to train on one shape (e.g. square cloth) and test on a very different shape (e.g. T-shirt).

3.4 Experiments

3.4.1 Experimental Setup

Simulation Setup As mentioned, we use the Nvidia Flex simulator wrapped in SoftGym (Lin et al., 2020) for training. The robot gripper is modeled as a spherical picker that can move freely in 3D space and can be activated so the nearest particle will be attached to it. For training, we generate random pick-and-place trajectories on a square cloth. The side length of the cloth varies from 25 to 28 cm. For evaluation, we consider three different shapes: 1) the same type of square cloth as used in training; 2) Rectangular cloth, with its length and width sampled from $[19, 21] \times [31, 34]$ cm. 3) Two layered T-shirt (the square cloth used for training was single-layered). For each shape, the experiment was run 40 times, each time with a different initial configuration of the fabric. We report the 25%, 50% and 75% (Q_{25}, Q_{50}, Q_{75}) percentiles of the performance. For all our quantitative results, numbers after $\pm$ denotes $\max(|Q_{50} - Q_{25}|, |Q_{75}, -Q_{50}|)$.

Our goal for cloth smoothing is to maximize the covered area of the cloth in the

top-down view. We report two performance metrics: Normalized improvement (NI) and normalized coverage (NC). NI computes the increased covered area normalized by the maximum possible improvement $NI = \frac{s-s_0}{s_{max}-s_0}$, where s_0, s, s_{max} are the initial, achieved, and maximum possible covered area of the cloth. Similarly, $NC = \frac{s}{s_{max}}$ computes the achieved covered area normalized by the maximum possible covered area. We report NI in this section and NC in the appendix.

We evaluates two variants of our method: Visible Connectivity Dynamics (VCD) and VCD with shape-specific training. We compare with previous state-of-the art methods for cloth smoothing: VisuoSpatial Foresight (VSF) (Hoque et al., 2020), which learns a visual dynamics model using RGBD data; Contrastive forward model (CFM) (Yan et al., 2020), which learns a latent dynamics model via contrastive learning; Maximal Value under Placing (MVP) (Wu et al., 2020), which uses model-free reinforcement learning with a specially designed action space. More implementation details can be found in the appendix.

Real World Setup We use our dynamics model trained in simulation to smooth cloth in the real world with a Franka Emika Panda robot arm and a standard panda gripper. We obtain RGBD images from a side view Azure Kinect camera. We use color thresholding for segmenting the cloth and obtain the cloth point cloud. We evaluate on three pieces of cloth: Two square towels made of cotton and silk respectively, and one t-shirt made of cotton. We use our dynamics model trained in simulation without any fine-tuning. More details are in the appendix.

3.4.2 Simulation Results

For each method, we report the NI after different numbers of pick-and-place actions. A smoothing trajectory ends early when NI>0.95. We note that the edge GNN can achieve a high prediction accuracy of 0.91 on the validation dataset. See appendix for visualizations of the edge GNN prediction.

We first test all methods on the same type of square cloth used in training. The results are shown in Figure 3.3 (left). Under any given number of pick-and-place actions, VCD greatly outperforms all of the baselines. The shape-specific training approaches described in Section 3.3.6 further improves the performance. To test the generalization of these methods to novel cloth shapes that are not seen during training, we further evaluate on a rectangular cloth and a t-shirt. For this experiment we only compare VCD to VSF,

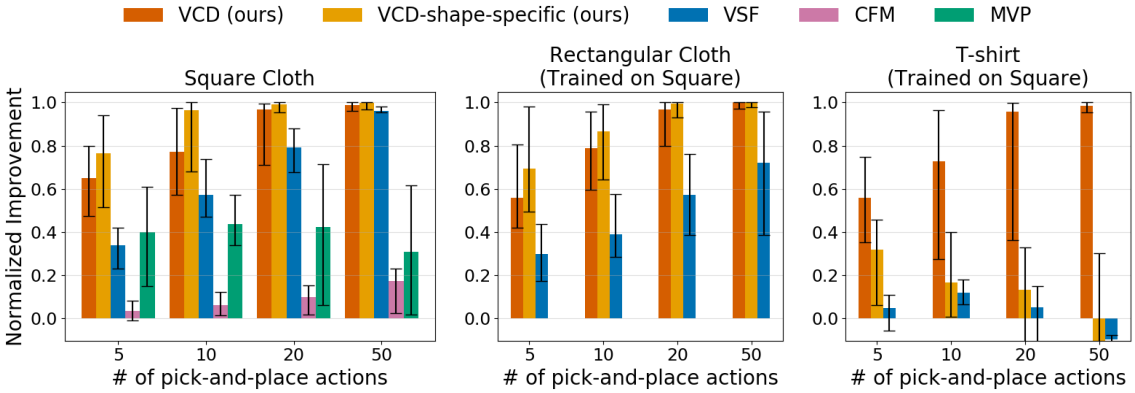


Figure 3.3: Normalized improvement on square cloth (left), rectangular cloth (middle), and t-shirt (right) for varying number of pick-and-place actions. The height of the bars show the median while the error bars show the 25 and 75 percentile. For detailed numbers, see the appendix.

since VSF achieves the best performance on the square cloth among all the baselines. The results are summarized in Figure 3.3 (middle and right). VCD shows a larger improvement over VSF on the rectangular cloth. T-shirt is more different from the training square cloth and VSF completely fails, while VCD still shows good generalization. The shape specific training still leads to marginal improvement and better stability on rectangular since it has a similar shape to the square cloth. However, as the t-shirt has very different shape compared to the square cloth, VCD-shape-specific cannot generalize to it.

Since VCD learns a particle-based dynamics model, it incorporates the inductive bias of the cloth structure, which leads to better performance and stronger generalization across cloth shapes, compared to RGB based method like VSF. Please see the appendix for examples of some planned pick-and-place action sequences of our method on all cloth shapes as well as visualizations of the predictions of our model.

Material \ # of pick-and-place actions	5	10	20	Best
Cotton Square Cloth	0.342 ± 0.265	0.725 ± 0.445	0.941 ± 0.360	0.941 ± 0.153
Silk Square Cloth	0.456 ± 0.197	0.643 ± 0.391	0.952 ± 0.229	0.952 ± 0.095
Cotton T-Shirt	0.265 ± 0.119	0.356 ± 0.096	0.502 ± 0.135	0.619 ± 0.155

Table 3.1: Normalized improvement of VCD in the real world.

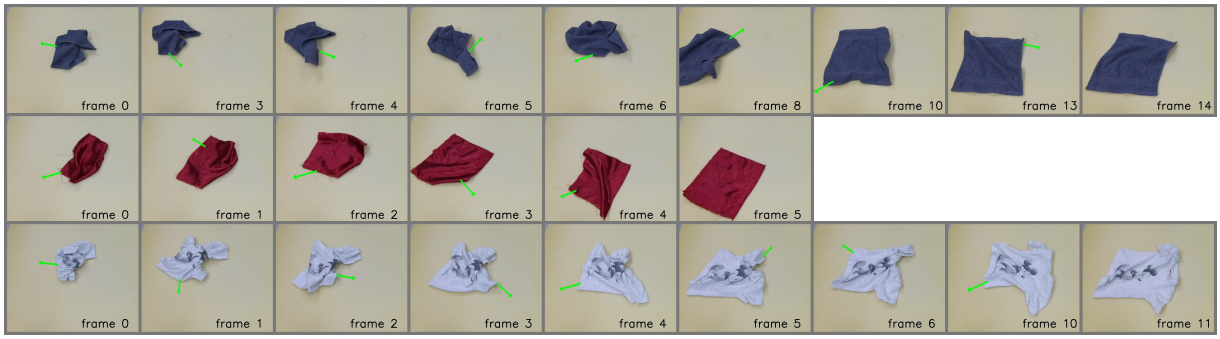


Figure 3.4: Smoothing cloths of different colors, materials and shapes with our method on a Franka robot: square cotton (top), square silk (middle), cotton t-shirt (bottom). Each row shows one trajectory. Frame 0 shows the initial configuration of the cloth, and each frame after shows the observation after some number of pick-and-place actions, with the number labeled on the frame. The green arrow shows the 2D projection of the pick-and-place action executed.

3.4.3 Real-world Results

We also evaluate our method for smoothing in the real world. We only evaluate VCD (i.e., without shape specific training) since it works well on all cloth shapes in simulation. Unfortunately, we were not able to evaluate the baselines in the real world due to the difficulties of transferring their RGB-based policies from simulation. All of the baselines use RGB data as direct input to the dynamics model or the learned policy, making them sensitive to the camera view and visual features. In contrast, our method uses a point cloud as input, which makes it robust to the camera position as well as variation in visual features such as the cloth color or patterns. The point cloud representation allows our method to easily transfer to the real world.

We evaluate 12 trajectories for each cloth. The quantitative results are in Table 3.1 and a visualization of smoothing sequences is shown in Figure 3.4. Despite the drastic differences of the cotton and silk cloths in visual appearances, shapes, as well as the different dynamics, our model is able to smooth the cotton and silk cloths and generalize well to t-shirt. We also report the performance if our method is able to terminate optimally in hindsight and choose the frame with the highest performance in each trajectory; the result is shown in the last column of Table 3.1. Videos of complete trajectories and the model predicted rollouts can be found on our project website.

3.4.4 Ablation Studies

Table 3.2: Normalized improvement of all ablations in simulation after 10 pick-and-place actions.

Algorithm	Normalized Improvement
VCD (Our method)	$\mathbf{0.770 \pm 0.206}$
Replace GNN with Flex	0.371 ± 0.144
Use only collision edges	0.605 ± 0.183
Use only mesh edges	0.405 ± 0.252
Remove edge GNN at test time	0.189 ± 0.102

We perform the following ablations to study the contribution of each component of our method. The first ablation replaces the learned GNN dynamics model with the Flex simulator to test whether a learned dynamics model performs better for our task than the physical simulator. In more detail, after we use the edge GNN to infer the mesh edges on the point cloud, we create a cloth using Flex where a particle is created at each location of the voxelized points and a spring connection is added for each inferred mesh edge. The results is shown in Table 3.2, row 2. We see that using the Flex simulator instead of the dynamics GNN produces significantly worse performance. The main reason is that the cloth created from the partial point cloud with the inferred mesh edges deviates from the common cloth mesh structure used in Flex; thus, using the Flex simulator under this condition does not create realistic dynamics. On the other hand, the dynamics GNN is trained directly on the partial point cloud; therefore it can learn to compensate for the partial observability when predicting the cloth dynamics. This ablation validates the importance of using a dynamics GNN to learn the dynamics of the partially observable point cloud.

The next set of ablations aims to test whether using an edge GNN to infer the mesh edges as described in Section 3.3.1 is necessary for learning a good dynamics model. First, we train a dynamics GNN without using the edge GNN, where the edges are constructed solely based on distance by Eqn. (3.1). Since this ablation does not use an edge GNN, it cannot have two different edge types (collision vs mesh edges). Thus at test time, all edges can either be kept fixed throughout the trajectory (similar to the mesh edges in our model), or dynamically reconstructed using Eqn. (3.1) at each time step (similar to the collision edges in our model). The results of these two ablations are shown in Table 3.2, rows 3 and 4. As can be seen, the performance is worse without the edge GNN.

Additionally, we perform another ablation where we train with both collision and mesh

edges, but at test time, we do not use an edge GNN to infer the edge type; instead we consider the edges that satisfy the criteria of Eqn. (3.1) in the first time step as the mesh edges. The result of this ablation is shown in Table 3.2, row 5. The performance is again much worse. All these ablations validate the importance of using an edge GNN to infer the mesh edges.

3.5 Conclusion

In this section, we propose the visible connectivity dynamics (VCD) model, that infers a visibility connectivity graph from the partial point cloud and learns a particle-based dynamics model over the graph for planning to perform cloth smoothing. VCD has the advantage of posing strong inductive bias that fits the underlying cloth physics, being invariant to visual features, and being interpretable. We show that VCD greatly outperforms previous state-of-the-art methods for cloth smoothing, and achieves zero-shot sim-to-real transfer on a Franka arm for smoothing various types of cloth.

Our work demonstrates the importance of the choice of state representation for efficient and generalizable manipulation, as well as the benefits of a graph-based representation. While there may not be a universal representation suitable for all objects, we believe that a graph representation can be an important alternative to raw images or latent vectors, especially for deformable objects such as cloth. In the future, we hope VCD can also be applied to different types of object manipulation tasks, such as manipulation of cables, bags, and food.

Additionally, while VCD only models the points on the visible surface of the graph, in a follow-up work Huang et al. (2022), we show that it can be further improved by training an additional module to estimate the full mesh of a cloth from crumpled configuration.

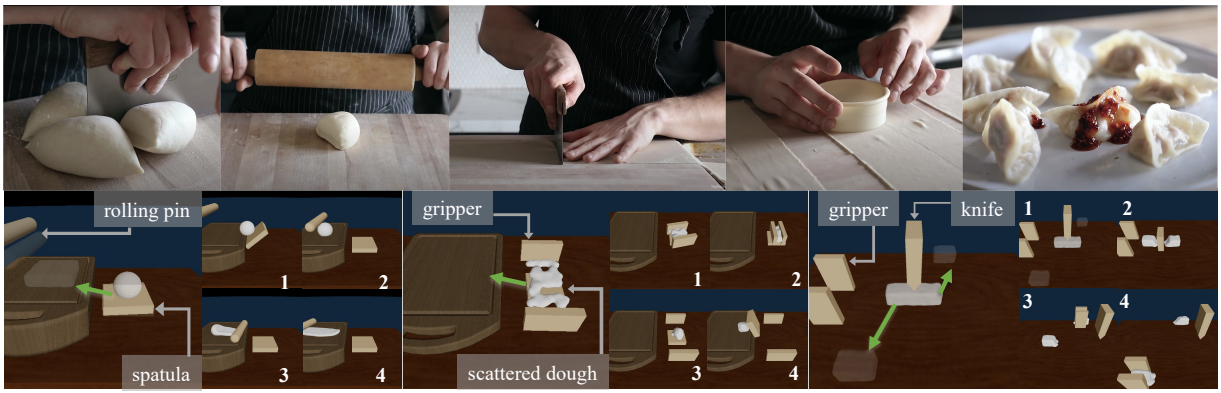
Chapter 4

Planning with Temporal Abstraction for Dough Manipulation

4.1 Introduction

Robot manipulation of deformable objects is a fundamental research problem in the field of robotics and AI and has many real-world applications such as laundry making (Maitin-Shepard et al., 2010), cooking (Bollini et al., 2013), and caregiving (Park et al., 2020). The recent development of differentiable physics simulators for deformable objects has shown promising results for solving soft-body control problems (Hu et al., 2019b; Heiden et al., 2021; Huang et al., 2021). These differentiable simulators have facilitated gradient-based trajectory optimizers to find a motion trajectory with much fewer samples, compared with black box optimizers such as CEM or reinforcement learning algorithms (Huang et al., 2021; Heiden et al., 2021; Geilinger et al., 2020).

However, the usage of these simulators is often limited in two ways. First, most differentiable physics simulators are only applied to solve short-horizon, single-skill tasks, such as cutting (Heiden et al., 2021), ball throwing (Geilinger et al., 2020), or locomotion (Hu et al., 2019b). This is partly because the gradient only provides local information, and thus, gradient-based optimizers often get stuck in local optima, preventing them from solving long horizon tasks. For example, consider a two-stage task of first gathering scattered dough on a table and then put it on the cutting board using a spatula (Figure 4.1). If



Task 1: LiftSpread Task 2: GatherTransport Task 3: CutRearrange

Figure 4.1: Humans use various tools to manipulate deformable objects much more effectively than state-of-the-art robotic systems. This work aims to narrow the gap and develop a method named DiffSkill that learns to use tools like a rolling pin, spatula, knife, etc., to accomplish complicated dough manipulation tasks. Our method learns skill abstraction using a differentiable physics simulator, composes the skills for long-horizon manipulation of the dough, and evalulated in three challenging sequential deformable object manipulations tasks: LiftSpread, GatherTransport, and CutRearrange.

the objective function is to minimize the distance between all the dough and the cutting board, a gradient-based trajectory optimizer would directly use the spatula to put a small amount of dough on the cutting board without ever using the gathering operation. Second, the agent needs to know the full simulation state and relevant physical parameters in order to perform gradient-based trajectory optimization, which is a great limiting factor for real-world generalization, as reliable full state estimation of deformable objects from sensory data like RGB-D images can often be challenging and sometimes impossible due to ambiguities caused by self-occlusions.

Planning over the temporal abstraction of skills is a powerful approach to tackle the first issue of solving long horizon tasks. However, finding a suitable set of skills is challenging. For example, while standard skills such as grasping an object or moving the robot arm from one pose to another may be manually specified (Toussaint et al., 2018), more complex skills such as cutting or gathering can be difficult to define manually. Therefore, it is desirable to learn these skills automatically.

In this chapter, we aim to solve a novel set of sequential dough manipulation tasks that involve operating various tools and multiple stages of actions (Figure 4.1), including lifting

and spreading, gathering and transporting, cutting and rearranging the dough. To extend the use of differentiable physics models to these long-horizon tasks and enable the agent to directly consume visual observations, we propose DiffSkill: a novel framework where the agent learns skill abstraction using the differentiable physics model and composes them to accomplish complicated manipulation tasks. Our method consists of three components, (1) a trajectory optimizer that acts as an expert that applies gradient-based optimization on the differentiable simulator to obtain demonstration trajectories, (2) a neural skill abstractor that is instantiated as a goal-conditioned policy, taking visual observations as input and imitating the demonstration trajectories, and (3) a planner that learns to assemble the abstracted skills to solve the long horizon task. Experiments show that our method, operating on the high-dimensional RGB-D images, can successfully accomplish a set of the dough manipulation tasks, which greatly outperforms the model-free RL baselines and the standalone gradient-based trajectory optimizer.

4.2 Related Work

Motion Planning with Differentiable Physics. Differentiable physics models provide the gradients of the future states with respect to the initial state and the input actions. Compared with black-box dynamics models, planning with differentiable physics models can often make more accurate updates on the action sequences and deliver better sample efficiency (Huang et al., 2021). Over the past few years, researchers have built differentiable physics models from first principles rooted in our understanding of physics for various physical systems, ranging from multi-body systems (Degrave et al., 2019; Tedrake and the Drake Development Team, 2019; de Avila Belbute-Peres et al., 2018; Geilinger et al., 2020), articulated bodies (Werling et al., 2021; Qiao et al., 2021), cloth (Qiao et al., 2020; Liang et al., 2019), fluid (Schenck and Fox, 2018), plasticine (Huang et al., 2021), to soft robots (Hu et al., 2019b;a; Du et al., 2021). They have shown impressive performance in using the gradient to solve inverse problems like parameter identification and control synthesis. A complement thread of methods tried to relax the assumption that we have to know the full state and the physics equations; instead, they employ deep neural networks to learn the dynamics models directly from observation data and use the gradients from the learned models to aid motion planning or policy learning (Battaglia et al., 2016; Chang et al., 2016; Mrowca et al., 2018; Ummenhofer et al., 2019; Sanchez-Gonzalez et al., 2020;

Pfaff et al., 2021; Sanchez-Gonzalez et al., 2018; Li et al., 2019b;a). Our method differs from prior works in that we use the differentiable physics model for skill abstract and compose the learned skills for long-horizon manipulation of deformable objects.

Deformable Object Manipulation. Deformable objects have more degrees of freedom and are typically more challenging to manipulate than rigid objects. Extensive work has been done in this area, focusing on different types of deformable objects, including plasticine (Li et al., 2019a; Huang et al., 2021), fluid (Li et al., 2022), cloth (Maitin-Shepard et al., 2010; Lin et al., 2021; Weng et al., 2021; Ha and Song, 2021; Wu et al., 2020; Yan et al., 2020; Ganapathi et al., 2020), rope (Nair et al., 2017; Sundaresan et al., 2020), and object piles (Suh and Tedrake, 2020). People have also come up with standard benchmarks to establish a fairer comparison between various algorithms (Lin et al., 2020). Our method aims to manipulate dough using tools similar to how humans would do in a kitchen environment (Figure 5.1). There are also a series of works on tool-using for object manipulation (Toussaint et al., 2018; Qin et al., 2020; Fang et al., 2020a; Xie et al., 2019), but most of them focus on rigid objects, whereas we take a step further and tackle deformable objects.

Solving Long Horizon Tasks. Our work aims to solve long-horizon manipulation tasks; thus, it is also closely related to task and motion planning (TAMP) that contains elements of discrete task planning and continuous motion planning. Typical solutions to TAMP involve formulating the problem as mixed-integer programming (MIP) over state or belief space and solving the program using branch-and-bound techniques (Gravot et al., 2005; Kaelbling and Lozano-Pérez, 2010; 2013; Srivastava et al., 2014; Toussaint, 2015; Toussaint et al., 2018; Driess et al., 2020; Garrett et al., 2020b). Please refer to Garrett et al. (2021) for a detailed and comprehensive survey of the related works. Differs from prior works, our method directly operates on the pixel space and generates motion trajectories and skill abstractions by applying gradient-based trajectory optimization techniques using the differentiable physics model. The abstracted skills are then composed sequentially to accomplish long-horizon manipulation tasks according to a higher-level task planner.

On the other hand, hierarchical reinforcement learning provides a general framework for learning both the low-level policy and a high-level controller at the same time through exploration (Dietterich, 2000; Bacon et al., 2017; Levy et al., 2017), where the two-level controllers make decisions at a different temporal scale. However, learning the temporal structure proves difficult, especially from high-dimension visual observation. Compared to hierarchical RL, our method takes advantage of the skills learned with differentiable

physics and greatly reduces the difficulty of learning a high-level controller or planner.

4.3 Method

Our goal is to learn a policy to perform sequential deformable object manipulation using tools from sensory observations. We assume that we are learning these skills in a differentiable physics simulator. Since it is not feasible to directly use a standalone differentiable physics solver to find an optimal solution for long-horizontal tasks, we propose to first learn to abstract primitive skills from this differentiable physics simulator; we then plan on top of these skills to solve long-horizon tasks. An overview of our framework is shown in Figure 4.2.

4.3.1 Problem formulation

We consider a Markov Decision Process (MDP) defined by a set of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$ and a deterministic, differentiable transition dynamics $s_{t+1} = p(s_t, a_t)$, with t indexing the discrete time. At each timestep, the agent only has access to an observation $o \in \mathcal{O}$ (such as an image) instead of directly observing the state. For any goal state $s_g \in \mathcal{G}$, a distance function from the state s is given as $D(s, s_g)$. The objective is to find a policy $a_t = \pi(o, o_g)$ that minimizes the final distance to the goal $D(s_T, s_g)$, where T is the length of an episode.

4.3.2 Demonstration Trajectories from Differentiable Physics

Previous work has shown that differentiable physics solvers can acquire short-horizon skills for deformable object manipulation in tens of iterations (Huang et al., 2021). Inspired by this work, we first collect demonstration trajectories of using each tool to achieve a short-term goal. Concretely, given an initial state s_0, a goal state s_g and the transition dynamics p of a differentiable simulator, we use gradient-based trajectory optimization to solve for an open-loop action sequence (Kelley, 1960). Specifically, we solve

$$\arg\min_{a_0,\dots a_{T-1}} L(a_0, \dots a_{T-1}) = \arg\min_{a_0,\dots a_{T-1}} \sum_{t=1}^{T} D(s_t, s_g) + \lambda \sum_{t=1}^{T} E(m(s_t), d(s_t)), \tag{4.1}$$

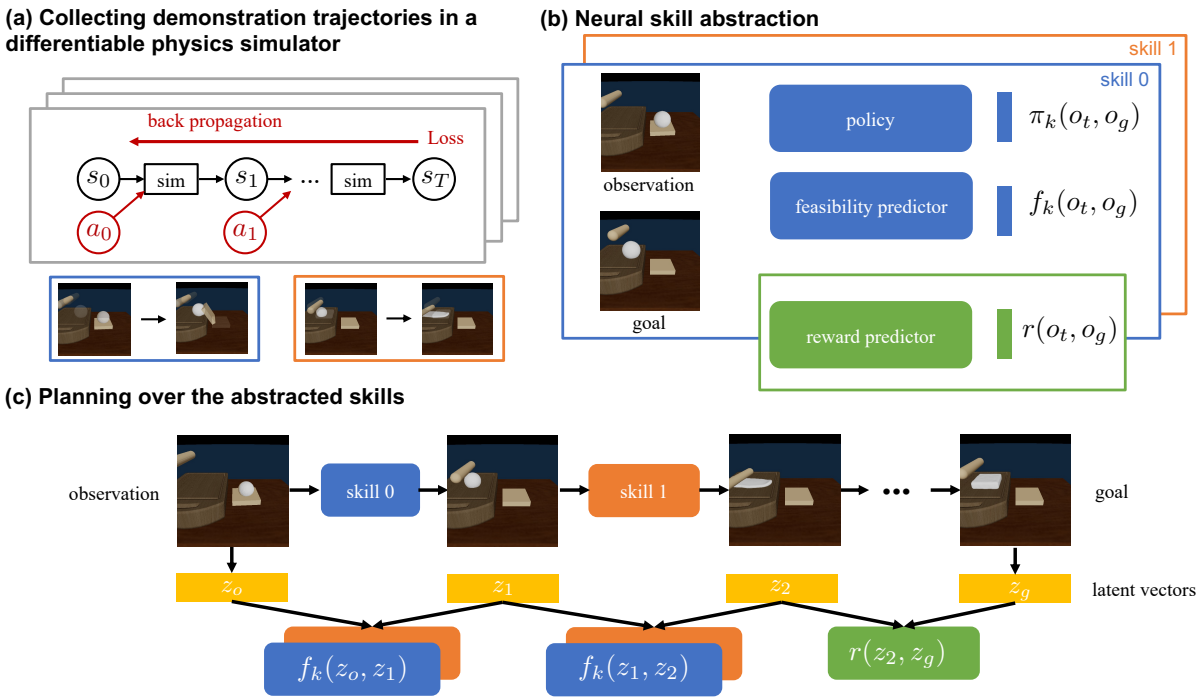


Figure 4.2: (a) Collecting demonstration trajectories by running a gradient-based trajectory optimizer in a differentiable simulator. (b) Neural abstraction by imitating the expert demonstration, which consist of a goal-conditioned policy, a feasibility predictor and a reward predictor. (c) Planning for both skill combination and the intermediate goals to solve long-horizon tasks.

where $s_{t+1} = p(s_t, a_t)$. In the case of deformable object manipulation, we represent the current and target shape of the deformable object as two sets of particles. We use the Earth Mover Distance (EMD) between the two particle sets as the distance metric $D(s_t, s_g)$, which we approximate with the Sinkhorn Divergence (Séjourné et al., 2019) . To encourage the manipulator to approach the deformable object, we additionally add into the objective the Euclidean distance between the manipulator and the deformable object $E(m(s_t), d(s_t))$, with a weight λ, where $m(s_t)$ and $d(s_t)$ are the positions of the center of mass of the manipulator and deformable object at time t, respectively. We solve Equation 4.1 by updating the action sequence using $\nabla_{a_t} L, t = 0 \dots T$, with an Adam optimizer (Kingma and Ba, 2014), initialized using an action sequence of all zeros.

In this paper, we define a "skill" as using a single tool to achieve a short-horizon goal s_g, starting from an initial state s_0. Assuming that we have K tools, the action space over all tools can be written as $[\mathcal{A}^{(1)}, \dots \mathcal{A}^{(K)}]$, where $\mathcal{A}^{(k)}$ is the action space of the kth tool. During demonstration collection, for each short-horizon goal s_g, we run the trajectory optimizer for each tool separately, by masking the actions for other tools to be zero at each timestep.

4.3.3 Neural Skill Abstraction

Although the trajectory optimizer is able to provide solutions for short-horizon tasks, it is unable to solve long-horizon tasks due to local optima. Furthermore, running the trajectory optimizer requires knowing the full state of the environment, including particle positions and mass distributions of the deformable objects and the physical parameters of these objects. This information is difficult to obtain during real-world deployment, where the observations available to a robot are from sensory information such as RGB-D images. Additionally, the trajectory optimizer takes minutes to run, which is too slow during evaluation for real-time applications.

As such, we propose to learn a neural skill abstractor that learns skills from the demonstration videos of a trajectory optimizer; we will then leverage these skills for solving long horizon tasks. Our neural skill abstraction consists of a goal-conditioned policy that takes a sensory observation (RGB-D images in our case) as input, a feasibility and reward predictor, as well as a variational auto-encoder (VAE).

Goal conditioned policy: For each tool, we learn a goal-conditioned neural policy

$a_t = \pi_k(o_t, o_g) \in \mathcal{A}^{(k)}$, using behavior cloning from the corresponding demonstration trajectories collected for the kth tool. Given the demonstration trajectory $(s_0, a_0, \ldots, s_T)$ and the corresponding RGB-D sensory observations $(o_0, \ldots o_T)$, as well as an observation of the goal o_g, we train a policy using the MSE loss $L_{policy} = ||\pi(o_t, o_g) - a_t||_2^2$. Furthermore, to make the policy more robust to goal observations outside of the training goal images, we adopt hindsight relabeling (Andrychowicz et al., 2017). When sampling from the demonstration buffer, with a probability p_{hind}, we will relabel the original goal o_g with a hindsight goal $\bar{o}_i$, where $\bar{o}_i \sim \text{Uniform}\{o_{t+1}, \ldots o_T\}$. The intuition is that, for any goal that will be achieved by the current action sequence, the policy should imitate the action at the current step.

Feasibility Predictor: For each skill, we also learn a feasibility predictor $f_k(o_t, o_g) \in \mathbb{R}$ to determine the feasibility of reaching from o_t to o_g using the trajectory optimizer. For training the feasibility predictor, we obtain positive pairs by sampling (o_{t_1}, o_{t_2}) from the same demonstration trajectory and negative pairs by sampling pairs of observations from different trajectories. We assign a label of 1 for positive pairs and a label of 0 for negative pairs. We use an MSE loss L_{fea} for model training, which was shown empirically to work slightly better than a cross-entropy loss. While there may be false negative pairs, we find the feasibility predictor to perform reasonably well, as different trajectories have different initial and goal configurations and it is less likely that the state of a different configuration (e.g. mass of the dough) can be achieved from the current state within one skill. This intuition was similarly used for goal relabeling in previous work by Lin et al. (2019b).

Reward Predictor: We further train a reward predictor $r(o_t, o_g) \in \mathbb{R}$ that predicts the negative of the Sinkhorn divergence between the corresponding states $-D(s_t, s_g)$ using an MSE loss L_r. The reward predictor does not depend on any specific skill or tool.

Variational Auto-Encoder: As explained in Section 4.3.4, we will plan to compose skills in a latent space instead of optimizing directly in the image space. To learn the latent space, we train a generative model of the observation space, specifically a VAE. The VAE includes an encoder $z = Q(o)$ that encodes an observation into a fixed length latent vector, and a decoder $o = G(z)$ that decodes back into the observation space. The VAE is trained such that $G(z), z \sim \mathcal{N}(\mathbf{0}, I)$ reproduces the observation distribution. The feasibility and reward predictor shares the encoder with the VAE. Thus, we can also write the feasibility

and reward predictor as

$$f_k(o_t, o_g) = \bar{f}_k(Q(o_t), Q(o_g)) = \bar{f}_k(z_t, z_g), \quad r(o_t, o_g) = \bar{r}(Q(o_t), Q(o_g)) = \bar{r}(z_t, z_g).$$

As such, the feasibility and reward predictor can be used with only latent vectors from the VAE encoder as input.

Training We jointly optimize the loss function $L_{skill} = L_{policy} + \lambda_{fea} L_{fea} + \lambda_r L_r + \lambda_{vae} L_{vae}$, where L_{vae} is the standard ELBO objective for VAE and $\lambda_{fea}, \lambda_r, \lambda_{vae}$ are hyperparameters.

4.3.4 Long-horizon Planning with Skill Abstraction

To apply the skill abstraction learned above sequentially, we need to determine 1) which skill to use at each stage; 2) what intermediate goal to specify for each skill. Given o_0, o_g the initial and goal observations, we plan over H steps. By using the feasibility and score predictor, we formulate our problem as a hybrid discrete and continuous optimization problem:

$$\underset{k_1, z_1, \ldots, k_H, z_H}{\arg\min} \quad C(\mathbf{k}, \mathbf{z}) = \prod_{i=0}^{H-1} \bar{f}_{k_i}(z_i, z_{i+1}) \bar{r}(z_H, z_g), \text{ s.t. } \quad ||z_i||_2^2 \leq M, \ \forall i. \tag{4.2}$$

Here, k_i is the index of the tool used at step i, $\bar{f}, \bar{r}$ are the feasibility and reward predictors that take latent vectors as input, and z_0, z_g are the VAE encoded latent vectors of the initial and goal observations (RGB-D image) o_0, o_g. Here, the optimization variables include the discrete variables $\mathbf{k} = k_1, \ldots, k_H$, representing the index of the skills to use at each step and the continuous variables $\mathbf{z} = z_1, \ldots, z_H$ that are latent vectors that represent the intermediate goals. The term $||z_i||_2^2$ in the constraint is proportional to the log likelihood of the latent vectors under a unit normal distribution and M is a threshold to ensure that the latent vectors correspond to actual intermediate goals in the observation space.

To optimize for both the discrete and continuous variables in Eqn. 4.2, we use exhaustive search over all possible combinations of the discrete variables. For the continuous variables, we start with N initial solutions $\{z_1, \ldots z_H\}_j, j = 1, \ldots, N$ and use Adam with projected gradient on the loss for all the initial solutions in parallel, since the constraint is convex. Specifically, after each gradient update step of Adam, we project the current z_i to the

Algorithm 1 Solve long-horizon planning with DiffSkill

Input: Trajectory optimizer, skill horizon T, planning horizon H
  Initialize modules for neural skill abstraction π_k, f_k, r, G, Q
  Generate N demonstration trajectories in differentiable physics $\tau = \{(o_i, a_i, r_i, o_g)\}$
  Train neural skill abstraction π_k, f_k, r, G, Q using loss L_{skill} until convergence
  for $\mathbf{k} = 0$ **to** K^H **do**
    Initialize $\mathbf{z} = [z_1, \dots z_H] \sim \mathcal{N}(\mathbf{0}, I)$
    Optimize $z_1, \dots z_H$ according to Eqn. 4.2 to obtain cost $C(\mathbf{k}, \mathbf{z})$
  Choose $\mathbf{k}, \mathbf{z}$ that minimizes $C(\mathbf{k}, \mathbf{z})$
  for $i = 0$ **to** H **do**
    Reset tools to initial poses
    Decode intermediate goal images: $o_{g,i} \leftarrow G(z_i)$
    Execute policy $\pi_{k_i}(\cdot,\ o_{g,i})$ in the environment for T steps

constraint set by setting $z_i = \frac{z_i}{\max(||z_i||_2/\sqrt{M}),1)}$. Once we have solved for $k_1, z_1, \dots, k_H, z_H$, we can decode the latent vectors back to images $o_1, \dots, o_H = G(z_1), \dots, G(z_H)$, and call the corresponding goal-conditioned policies sequentially: $\pi_{k_1}(o_0, o_1), \dots, \pi_{k_H}(\cdot, o_H)$. To simplify the execution, we move each tool to its initial pose at the beginning of executing each skill. We also present the psedocode of our method in Algorithm 1.

4.4 Experiments

In this section, we will discuss our experimental setup, implementation details, baselines, and comparison results.

4.4.1 Experimental setup

Tasks and environments We experiment with a set of sequential deformable object manipulation tasks with dough. We build our simulation environments on top of PlasticineLab (Huang et al., 2021), a differentiable physics benchmark using the DiffTaichi system (Hu et al., 2019a) that could simulate plasticine-like objects based on the MLS-MPM algorithm (Hu et al., 2018). Inspired by the dumpling making process, we design three novel tasks that require long-horizon planning and usage of multiple tools:

- **LiftSpread**: The agent needs to first use a spatula (modeled as a thin surface) to lift a dough onto the cutting board and then adopt a rolling pin to roll over the dough to

flatten it. The rolling pin is simulated as a 3-Dof capsule that can rotate along the long axis and the vertical axis and translate along the vertical axis to press the dough.

- **GatherTransport**: Initially, residual of dough is scattered over the table. First, the agent needs to gather the dough with an extended parallel gripper and place it on top of a spatula. Then the agent needs to use the spatula to transport the dough onto the cutting board. The spatula can translate and rotate along the vertical plane. The gripper can translate along the horizontal plane, rotate around its center and open or close the gripper.
- **CutRearrange**: This is a three-step task. Given an initial pile of dough, the agent needs to first cut the dough in half using a knife. Inspired by the recent cutting simulation (Heiden et al., 2021), we model the knife using a thin surface as the body and a prism as the blade. Next, the agent needs to use the gripper to transport each piece of the cut dough to target locations. The knife can translate in 3D.

A visualization of these tasks can be found in Figure 5.1. For all the taks, the agent receives RGBD image resized to 64x64 from a camera and return velocity control commands directly on the tools.

Evaluation metric We report the normalize decrease in Sinkhorn diverge computed as $s(t) = \frac{s_0 - s_t}{s_0}$, where s_0, s_t are the initial and current Sinkhorn divergence. In this way, a normalized performance of 0 representing a policy that does nothing and a normalized performance of 1 representing an upper bound of a policy that perfectly match the two distributions. Note that the normalized performance can be negative if the policy makes the Sinkhorn divergence larger compared to the initial state. The maximum normalized performance of 1 may not always be possible. For example, due to the incompressibility of the dough, certain target shape may be too small for a large dough to fit into. As such, we additionally set a threshold on the Sinkhorn divergence for each task such that the task is assume to be completed successfully when EMD is below the threshold. This threshold is manually picked by observing the performance gap between successful and failed trajectories. For each task, 5 initial and target configurations are given for evaluation, which all require multi-stage manipulation of the dough. We report both the normalized performance metric and the success rate for comparisons.

4.4.2 Implementation Details

For each task, we first generate 1000 different initial and goal configurations, varying the initial and target shape of the dough as well as poses of the manipulators. For each configuration, we run the optimizer for each tool separately to generate trajectories of length 50. The Adam optimizer starts with a initial action sequence of all zero and is then run for 200 iterations to optimize each trajectory.

We then jointly train our VAE, policy, feasibility and score predictors over this demonstration video dataset. These modules share the same encoder architecture: 4 convolutional layers with a kernel size of 4 and a stride of 2 and a channel size of $(32, 64, 128, 256)$, followed by an MLP that maps the feature into a fixed 16 dimension vector. The VAE, feasibility and the score predictor also share the same weights for the encoder.

After training, we find the feasibility and score predictor to perform well over the held out trajectories, achieving a L2 error of less than 0.05 for the score predictor and an accuracy of over 0.95 for the feasibility trajectory, across different environments. We found behavior cloning to be sufficient for learning short-horizon skills from the demonstration dataset. In Table 4.3, we can see that the learned skills (labeled as Behavior Cloning) approach the normalized performance of the trajectory optimization (Trajectory Opt) on single-tool use, even though they are unable to solve the long-horizon tasks.

Given the skill abstraction, DiffSkill iterates through all skill combination. For each skill combination, we randomly sample 400 initial solutions for the latent vectors $z_1, \ldots, z_H$, and then perform 1500 iterations of Adam optimization for each of them. Since we are optimizing in the latent space, and with the help of parallel computation of GPU, this optimization can be done efficiently in around 10 seconds for each skill combination on a single NVIDIA 2080Ti GPU.

4.4.3 Baselines

We compare with there strong baselines:

Model-free Reinforcement Learning (RL) We compare with TD3 (Fujimoto et al., 2018), a state-of-the-art actor-critic RL method with a deterministic policy. The RL method uses the same encoder architecture as DiffSkill. The RL agent receives the negative of the cost function used for the trajectory optimizer as the reward function. We train the RL agent with either a single tool, or with both tools where the agent controls them

simultaneously.

Behavior Cloning We compare with another baseline that directly trains a goal-conditioned policy with Behavior Cloning (BC) using all tools. The agent is trained with the same dataset and jointly controls all tools.

Trajectory Opt (Oracle) We compare with the trajectory optimizer used to generate the demonstration, with the same parameters. Note that this method requires full state of the simulation and multiple forward and backward passes through the simulator during evaluation time.

For all baselines that sequentially apply multiple skills (BC, DiffSkill), we move any used tools to its initial pose after executing each skill with a manual policy.

4.4.4 Results

We show that DiffSkill is able to solve the challenging long-horizon, tool-use tasks from the sensory observation (RGBD) while the baselines cannot. The quantitative results can be found in Table 4.3. Tool A refers to the spatula for LiftSpread, the parallel gripper for GatherTransport and Knife for the CutRearrange environment, while tool B refers to the rolling pin, the spatula and the gripper for each task respectively.

First, we can compare the rows for single-tool use. Trajectory Opt is able to solve one-stage of the task, achieving a reasonable performance, if the correct tool is used (e.g. tool A for LiftSpread and GatherTransport), although for cutting, the performance is still close to zero with cutting only, since cutting itself does not reduce the distance between the current dough and the desired dough locations. BC can well approach the performance of the corresponding trajectory optimizer despite being a policy that only receives RGBD input. On the other hand, RL performs poorly on tasks, showing that it is difficult to learn A good policy directly from visual input.

Second, with multiple tools, we can see that DiffSkill significantly outperforms the single-skill policy. Remarkably, DiffSkill even outperforms the trajectory optimizer that controls both tools at the same time, which uses the full simulation state during evaluation time. This indicates that high-level planning over skills is necessary for achieving these sequential manipulation tasks. Furthermore, we show visualization of the found solutions of DiffSkill in Figure 4.3. We can see that DiffSkill is able to find reasonable intermediate goal state such as first putting the dough on the cutting board, or gather dough onto the spatula.

CutRearrange is shown to be a harder task, with all method performing poorly. This is because it requires three stages of manipulation and both the cutting and transporting dough without deforming the dough too much are non-trivial skills. Still, DiffSkill is able to achieve higher normalized improvement.

	Method \ Task (H)	LiftSpread (2)	GatherTransport (2)	CutRearrange (3)
Tool A only	Trajectory Opt	0.307 / 0%	0.544 / 0%	0.030 / 0%
	Behavior Cloning	0.291 / 0%	0.540 / 0%	0.018 / 0%
	Model-free RL	-0.423 / 0%	-0.081 / 0%	-0.010 / 0%
Tool B only	Trajectory Opt	0.029 / 0%	0.495 / 20%	0.264 / 20%
	Behavior Cloning	0.038 / 0%	0.438 / 0 %	0.203 / 20%
	Model-free RL	-0.423 / 0%	-0.008 / 0%	-0.471 / 0%
Multi-tool	Trajectory Opt	0.343 / 0%	0.385/ 20%	0.312 / 20%
	Model-free RL	-1.769/ 0%	0.231 / 0%	-2.434 / 0%
	DiffSkill (Ours)	**0.45 / 100%**	**0.663 / 60%**	**0.367 / 20%**

Table 4.1: Normalized improvement of all methods and the success rate on different tasks. Each entry shows the normalized improvement / success rate. The top bar shows H, the planning horizon for each environment. Note that the normalized improvemnet metric can be negative if the method makes the Sinkhorn divergence larger compared to the initial state.

4.4.5 Ablation Analysis

We perform two ablations on DiffSkill. First, we try removing the planning over the discrete variables that decides which tool to use at each step. Instead, we train a tool-independent policy as well as the feasibility and score predictor, where the action space is the joint-tool action space. Then during planning, we only need to optimize for the intermediate goals. This ablation is labeled as *No Discrete Planning*.

Second, we try to remove the planning over the continuous variables, i.e. the intermediate goals. Without the intermediate goals, we directly use the final target image as the goal for sequentially executing each skill. This ablation is labeled as Direct Execution. Since we do not have the intermediate goals anymore, we try two different ways of choosing the skills to execute at each stage: Randomly pick one skill and an oracle that execute each combination of skills in the simulator and choose the best one in hindsight. The results are shown in Figure 4.2. Without discrete planning, the policy performs poorly. This is probably

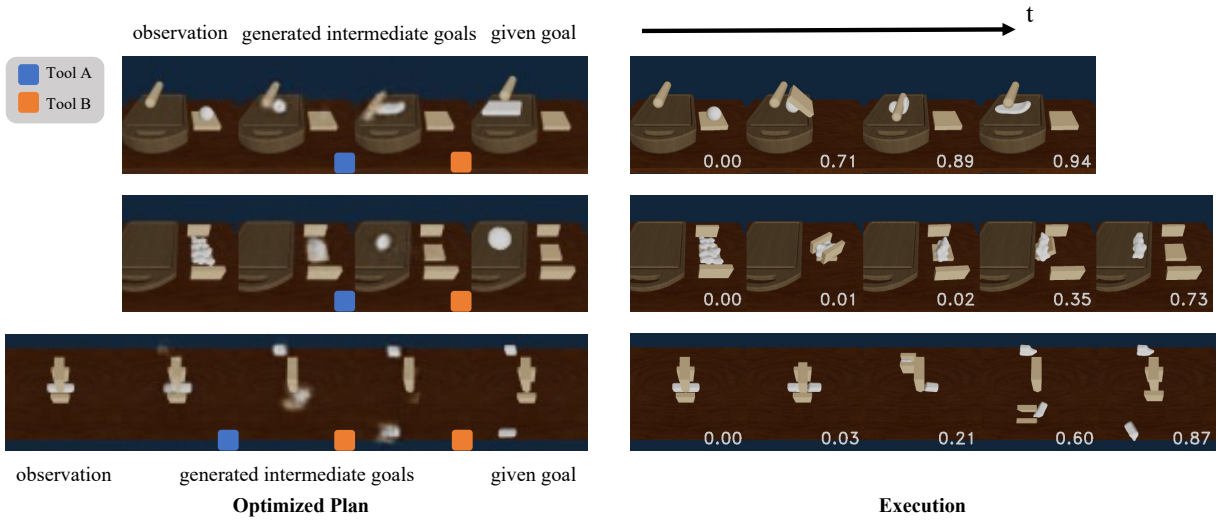


Figure 4.3: Visualization of the plan and the execution. The top plan generated by DiffSkill is shown in the left, where the first and the last image are the given initial and goal observation and in between are the decoded intermediate goals from the solution with the smallest loss during planning. The right shows sampled frames during the execution of the generated plan using the corresponding goal conditioned policy. The numbers on the bottom right shows the achieved normalized improvement metric at that time.

because during training, a single policy and feasibility predictor is used for learning two modes of skills that are very different and the policy is unable to differentiable for different modes or decide when to switch modes. On the other hand, if we do not optimize for the intermediate goals, we also cannot determine which tools to use at evaluation time, since both the feasibility predictor requires intermediate goals as input. In this case, we can see that using random skills at each stage results in poor performance. Even if the oracle skill order is used, there is still a drop in performance as the policy may not work well given only a future goal instead of an immediate goal.

Table 4.2: Normalized improvement and success rate of ablation methods.

Method \ Task	LiftSpread	GatherTransport	CutRearrange
No Discrete Planning	0.329 / 20%	0.421 / 0%	0.034 / 0%
Direct Execution (Random)	0.263 / 15%	0.419 / 20%	0.037 / 0%
Direct Execution (Oracle)	0.395 / 60%	0.633 / 60%	0.238 / 20%
DiffSkill (Ours)	**0.45 / 100%**	**0.663 / 60%**	**0.367 / 20%**

4.5 Conclusions

In this section, we propose DiffSkill, a novel framework for learning skill abstraction from differentiable physics and compose them to solve long-horizontal deformable object manipulations tasks from sensory observation. We evaluate our model on a series of challenging tasks for sequential dough manipulation using different tools and demonstrate that it outperforms both reinforcement learning and standalone differentiable physics solver. We hope our work will be a first step towards more common application of differentiable simulators beyond solving short-horizon tasks. We are looking forward to combining the learned skills to solve more complex tasks which requires further investigation on the planning method.

In some cases, we can find simple methods to sidestep the local optima in a gradient-based trajectory optimizer by designing a differentiable reset module that resets the tool to a new pose (Qi et al., 2022). In this work, we further transfer a point cloud policy learned in the same simulator to the real world.

In Chapter 5, you will see how we combine the DiffSKill framework with spatial abstraction and use a 3D latent set representation to plan for more complex tasks and transfer it to the real world.

Chapter 5

Planning with Spatial-Temporal Abstraction for Dough Manipulation

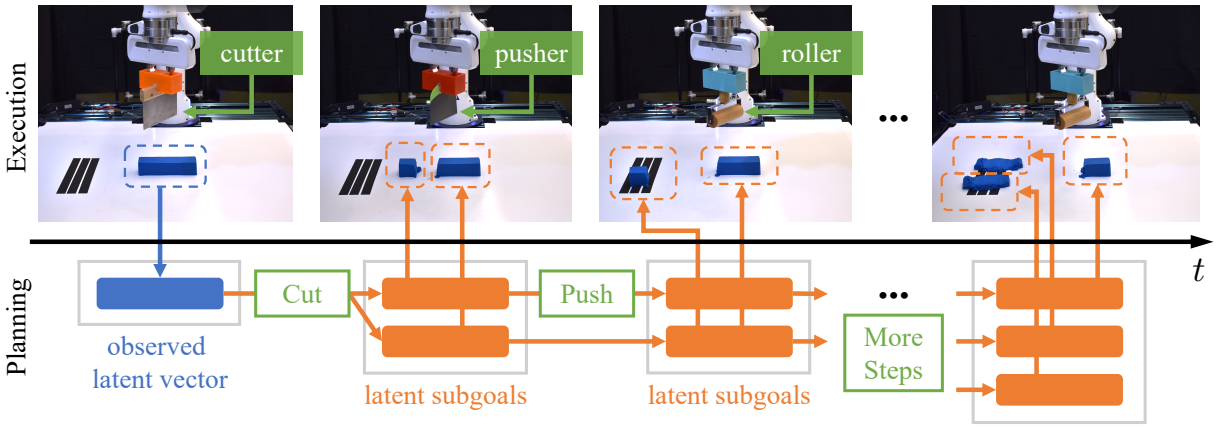


Figure 5.1: **Long-horizon dough manipulation with diverse tools.** Our framework is able to solve long-horizon, multi-tool, deformable object manipulation tasks that the agent has not seen during training. The illustrated task here is to cut a piece of dough into two with a cutter, transport the pieces to the spreading area on the left (with a high-friction surface) using a pusher, and then flatten both pieces with a roller.

5.1 Introduction

Consider a typical cooking task of making dumplings from dough. People plan over which piece of dough to manipulate and which tool to use in sequence, incorporating both spatial and temporal abstractions. A spatial abstraction reasons about objects, parts, and their relations to each other, such as reasoning about pieces of dough instead of reasoning about individual dough atoms; such a spatial abstraction enables efficient planning and compositional generalization. On the other hand, a temporal abstraction incorporates abstract actions represented as a set of skills such as deciding which tool to use at different task stages, instead of making plans at low-level actions such as joint torques. Temporal abstractions allow planning at the skill level, enabling more efficient optimization for solving long-horizon tasks. An autonomous robot that operates in unstructured environments should be able to reason about world dynamics using high-level spatial and temporal abstractions instead of reasoning only over the atoms, infinitesimally small timesteps, and low-level robot actions.

The research community is making rapid progress towards developing state abstractions for manipulating complex objects, including key points (Manuelli et al., 2019), graphs (Li et al., 2019a; Lin et al., 2021), dense object descriptors (Florence et al., 2018), or implicit functions (Li et al., 2022; Driess et al., 2022). However, most of these approaches do not make abstractions at the temporal level, limiting their use to short-horizon tasks. Methods are also being developed with temporal abstractions, planning over a set of skills to solve long-horizon tasks (Lin et al., 2022a; Nasiriany et al., 2022; Dalal et al., 2021). However, the lack of spatial abstraction severely limits their generalization ability. Therefore, it remains a key question in robot learning on how to learn spatial and temporal abstractions within a unified framework for complex and long-horizon manipulation tasks.

In this work, we focus on the challenging task of sequential deformable object manipulation, as shown in Figure 5.1. We consider a set of dough manipulation tasks that require sequentially applying different skills using multiple tools to manipulate dough, such as spreading using a roller, cutting using a knife and pushing using a pusher, where the longest task requires applying 6 skills in sequence. Deformable objects like dough have nearly infinite degrees of freedom. As such, in this work, we dynamically cluster points in a point cloud into different groups and learn a point cloud encoder to map each element in the group into a latent vector. In this way, we obtain a compositional 3D set representation

of the state space. Given an observation and a target point cloud, we then sample skill sequences with subgoals generated in this latent space. We learn skill abstraction modules to determine the feasibility and score of each skill sequence to plan for the skill sequences and the subgoals to reach the goal.

Our contribution of this paper is a framework PASTA, PlAnning with Spatial-Temporal Abstraction, that learns a set of skill abstraction modules over a 3D set representation, which enables our method to efficiently plan over both a spatial and temporal abstraction of the environment. With this spatial-temporal abstraction, PASTA can compose a set of skills to solve complex tasks with more entities and longer-horizon than what was seen during training. We show that PASTA significantly outperforms an ablation that performs planning with a flat representation without a spatial abstraction (e.g. without a 3D set representation). Finally, our planner can be trained in simulation and transferred to the real world for solving sequential tasks of deformable object manipulation.

5.2 Related Work

Model-based Planning for Sequential Manipulation. Model-based systems for sequential manipulation generally take one of two approaches to construct the dynamics model. The first approach, Task and Motion Planning (TAMP), constructs the model based on physical rules. These systems typically assume known object states and known effects for the action operators (Garrett et al., 2020a; Toussaint et al., 2018; Fikes and Nilsson, 1971; McDermott et al., 1998; Toussaint, 2015). However, it is difficult to estimate states and dynamics for unknown objects, deformable objects, or from partial observations. While recent works have made progress in learning certain components of the system, such as the logical states (Yuan et al., 2022) from high dimensional observations or learning action models (Liang et al., 2022; Ugur and Piater, 2015; Wang et al., 2021) from interactions, they still require either known states or known action operators. In contrast, our work learns a 3D set representation as well as the action model on the representation.

Another approach learns dynamics directly from visual observations (Li et al., 2019a; Ebert et al., 2018; Hafner et al., 2019). Most of these works focus on learning a one-step dynamics model for planning short-horizon tasks. A few works learn the dynamics model over a set of skills and use it for sequential manipulation of rigid objects (Fang et al., 2020b; Simeonov et al., 2020) or deformable objects (Lin et al., 2022a). However, these works does

not use an object-centric representation and thus cannot easily generalize to more complex scenes. In contrast, our framework unifies both temporal and spatial abstraction and can perform long-horizon manipulation for complex tasks with more objects than in previous work, as we will show.

Planning with Spatial Abstraction. Many prior works leverage spatial abstraction to facilitate solving tasks that involve complex dynamics and high-dimensional observations. These works either model a compositional system with Graph Neural Networks (GNN) (Li et al., 2019a; Lin et al., 2021; Driess et al., 2022; Pfaff et al., 2021; Ma et al., 2022) or learn policies directly from object-centric representations (Heravi et al., 2022; Devin et al., 2018). These works demonstrate compositional generalization, but they only learn policies or one-step dynamics models for planning, which can be difficult for solving long-horizon tasks. In contrast, our framework connects temporally extended spatial abstractions with a feasibility predictor, which allows us to plan over a longer time horizon.

Deformable Object Manipulation. Deformable objects have nearly infinite degrees of freedom and complex dynamics, making them very challenging to manipulate. Previous works have explored pouring liquid (Li et al., 2022; Schenck and Fox, 2017; Gautham et al., 2022), rope manipulation (Sundaresan et al., 2020; Mitrano et al., 2021), and cloth manipulation (Maitin-Shepard et al., 2010; Hoque et al., 2021; Lin et al., 2021; Huang et al., 2022; Weng et al., 2021). Other papers have also explored manipulating elasto-plastic objects such as deforming them by grasping (Li et al., 2019a; Shi et al., 2022), rolling (Figueroa et al., 2016; Matl and Bajcsy, 2021), or cutting (Heiden et al., 2021). However, these works mostly only consider manipulation with one skill at a time. In contrast, we consider the task of sequential manipulation using multiple tools. The one exception is DiffSkill (Lin et al., 2022a), where multiple skills are chained together. However, DiffSkill uses RGB-D images to represent the scene. In contrast, we use a 3D set representation extracted from point clouds, which enables us to generalize compositionally to tasks with more objects and longer-horizon and also to transfer the planning method to the real world.

5.3 Method

Given a point cloud of the dough P^{obs} and a goal point cloud P^{goal}, our objective is to execute a sequence of actions $a_1, ..., a_T$ that minimizes the distance between the final observed point cloud and the goal $D(P_T^{obs}, P^{goal})$ where P_i^{obs} is the segmented observation

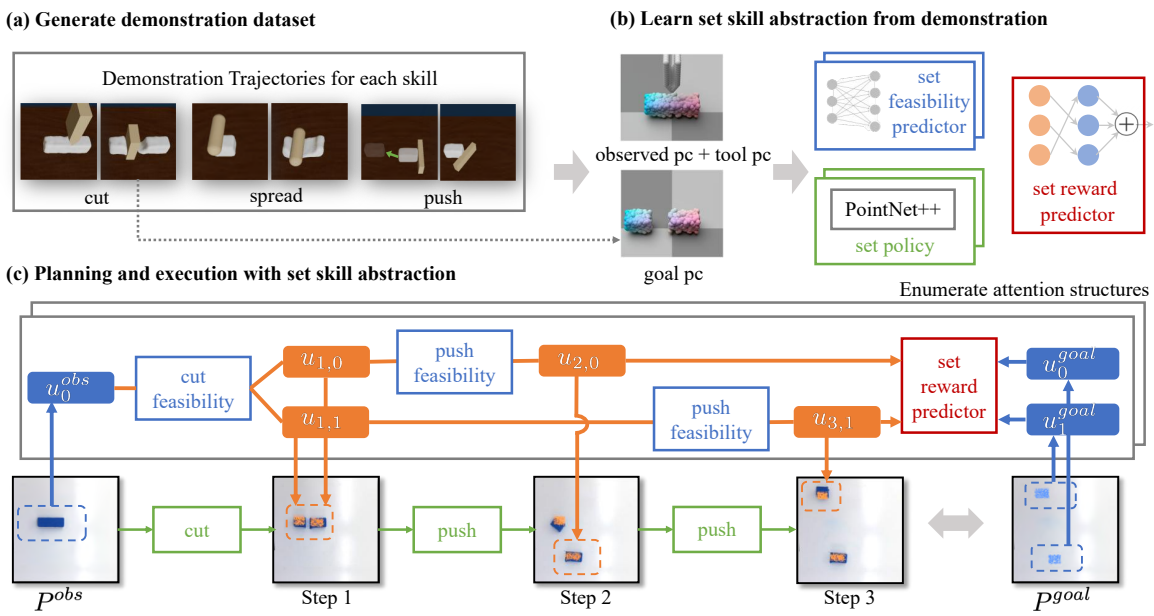


Figure 5.2: **Overview of our proposed framework PASTA.** (a) We first generate demonstration trajectories for each skill in a differentiable simulator using different tools. (b) We then sample point clouds (pc) from the demonstration trajectories to train our set skill abstraction modules. (c) We map point clouds into a latent set representation and plan over tool-use skills to perform long-horizon deformable object manipulation tasks. P^{obs}, P^{goal} are the observation and target pc; $u_{i,j}$ denotes component j at step i. The example shows our method performs the CutRearrange task, which requires cutting the dough into two pieces with a knife and transporting each piece to its target location.

point cloud at time i. We aim to solve long-horizon tasks that require chaining multiple skills in novel scenes with more objects than training. To do so, we present a general framework that incorporates spatial and temporal abstractions for learning and planning with skills from high-dimensional observations, as summarized in Figure 5.2. We use point clouds as input to all our modules to enable easier transfer from simulation to the real world and to enable robustness to changes in viewpoint.

We assume access to an offline dataset of demonstration trajectories $\mathcal{D}_{demo}$, where each trajectory demonstrates one of the K skills using one tool. We can learn skill policies by imitation learning from these demonstration. To chain these skills to solve long-horizon tasks, we train a set of skill abstraction modules (Sec. 5.4), which can be used for efficient planning in a latent space. Below, we first describe how we generate this latent space and use an object-centric representation to generalize our planner to scenes with more objects.

5.3.1 Spatial Abstraction from Point Clouds

Scene Decomposition: First, we describe the representation that we use as a spatial abstraction of the point cloud observation. Given a point cloud $P \in \mathbb{R}^{N\times 3}$, we first cluster particles into different components based on their proximity in space. In this paper, we apply DBSCAN (Ester et al., 1996) to P and group points into a set of entity point clouds $\{P_i \in \mathbb{R}^{N_i\times 3}\}_{i=1\ldots C}$ by separating points from high-density regions into different clusters. While other works on scene decomposition can also be used (Locatello et al., 2020), we find this simple method to be sufficient for our tasks.

Entity Encoding: Planning directly over the $\mathbb{R}^{N\times 3}$ dimensional space of point clouds is too inefficient. Instead, to enable efficient planning in a latent space, we train a point cloud variational autoencoder (Yang et al., 2019). The variational autoencoder model includes three modules: A point cloud encoder $\phi : \mathbb{R}^{N\times 3} \to \mathcal{U}$ that maps a point cloud to a latent vector, a decoder $\psi : \mathcal{U} \to \mathbb{R}^{N\times 3}$ that maps from a latent space back into a point cloud, and a prior distribution over the latent space $p_u : \mathcal{U} \to [0, 1]$ which can be used to generate samples from the latent space during planning. We can the encode each entity point cloud P_i into a set of latent vectors: $\{u_i\}_{i=1\ldots C}$ as our set latent representation. We further achieve translation invariance by separating the translation from its shape embedding. See our appendix for details.

5.4 Neural Spatial-Temporal Abstraction

Given a demonstration dataset, we can learn a set of goal-conditioned policies as the skills. However, we assume the demonstration trajectories only show how to perform a single-stage task with a single tool and can be just for a single object, so our skills are also limited to single-stage, single-object tasks. In order to plan with these skills to solve longer-horizon tasks, we further learn a feasibility predictor and a reward predictor. They can be used to plan subgoals that chain the skills into a sequence, such that each subgoal is feasible for the corresponding policy to reach and the final subgoal reaches a given goal. Additionally, all these modules take in the set representation $\{P_1 \dots P_C\}$ as input such that the planner also generalizes to tasks with more objects.

Set Point Cloud Policy The set point cloud policy for the kth skill π_k takes in an observed point cloud P^{obs}, a goal point cloud P^{goal}, and a tool point cloud P_k^{tool} and outputs an action at each timestep to control the tool directly. The policy only sees a subset of the dough point cloud and goal point cloud in the scene to enable compositional generalization to scenes with more objects. For example, a dough spreading policy will only see the dough being spread. To achieve this, we train each set policy with behavior cloning and hindsight relabeling on the demonstration dataset with an attention mask that filters out the non-relevant entity point clouds. During planning, this attention mask will be provided by the planner.

Set Feasibility Predictor Similar to DiffSkill (Lin et al., 2022a), we train a feasibility predictor $f_k(U^o, U^g)$ for each skill, where $U^o = \{u_i^o\}_{i=1\dots N_o}, U^g = \{u_j^g\}_{j=1\dots N_g}$ are latent set representations of an observation and goal point cloud respectively. The feasibility predictor outputs a value in $[0, 1]$ denoting whether the goal can be reached from the observation by executing the kth skill. In DiffSkill, the feasibility predictor uses a flat representation that takes in a single latent vector for all objects in the scene as input. However, as our skills such as cutting or spreading only need to take in a subset of the objects as input, we use the same attention method for the feasibility and assume that the feasibility predictor only takes as input a subset of the full set representation $\hat{U}^o \subseteq U^o, \hat{U}^g \subseteq U^g$, where $\hat{U}^o = \{u_i^o\}_{i=1\dots N_k}, \hat{U}^g = \{u_j^g\}_{j=1\dots M_k}$, where N_k and M_k are the number of components in the observation and goal for skill k. The number of components in the observation and goal can be different since the number of components can change before and after executing a skill; for example, the cut skill takes one component as observation and cuts it into

two components. As another example for robot assembly (Lee et al., 2021), the number of entities increases when a piece is disassembled into parts and the number of entities decreases when the parts are assembled. In this work, we manually define the number of entities N_k, M_k per skill. Determining which subset to attend to when executing each skill can be difficult; we make this decision during planning and defer the details to Sec. 5.4.1. We parameterize f_k to be invariant to permutation of the set by first max-pooling the transformed latent vectors into a single vector for both $\hat{U}^o$, and $\hat{U}^g$ (after filtering the elements that are not attended to) followed by a Multi-Layer Perceptron (MLP).

We train the feasibility predictor of skill k with positive examples $\hat{U}^o, \hat{U}^g$ (examples in which the goal $\hat{U}^g$ can be reached from the observation $\hat{U}^o$ within T timesteps by executing skill k) and negative examples (in which the goal $\hat{U}^g$ cannot be reached from the observation $\hat{U}^o$ using skill k). During training, we obtain positive pairs for the feasibility predictor by sampling two point clouds (P^{obs}, P^{goal}) from the same trajectory in the demonstration set. To find $\hat{U}^o, \hat{U}^g$, we first cluster the observation and goal point clouds into two sets $\{P_i^o\}, \{P_j^g\}$ respectively. Then, we match point clouds in the observation set to those in the goal set by finding the pairs of point clouds that are within a Chamfer distance of ϵ: $\{(P_i^o, P_j^g) \mid D_{Chamfer}(P_i^o, P_j^g) < \epsilon\}$. We then remove these point clouds from the corresponding set, since these are the point clouds that have already been moved to the target location in the goal. We can then encode the remaining point clouds into $\hat{U}^o, \hat{U}^g$ as explained above. We generate hard negative samples by randomly replacing one of the entities in the positive examples; see the appendix for details.

Set Reward Predictor As we do planning in a latent space, we train a set reward predictor as our planning objective which determines how close a plan is to a given goal. The set reward predictor R takes two latent set representation as input U^o, U^g. Since our tasks focus on matching each entity in the observation with one in the goal, we assume they have the same number of components, i.e. $|U^o| = |U^g| = N_r$. To compute the reward, we try to find the matching entity with the minimal matching cost: $R\Big(\{u_i^o\}, \{u_j^g\}\Big) = \text{argmax}_\sigma \sum_{i=1}^{N_r} r_\theta(u_i^o, u_{\sigma(i)}^g)$, where σ is a permutation and r_θ is a reward prediction network parameterized by an MLP trained to predict the negative Chamfer distance between the latent vectors of the two point clouds. Optimization of the reward computation is done by performing Hungarian matching between the two sets, where the matching cost is the negative of the predicted reward.

5.4.1 Planning with Set Representation

Given an observation and a goal point cloud P^{obs}, P^{goal}, we can plan subgoals and the sequence of skills using our trained abstraction modules, such that we can use our skills to follow each subgoal sequentially to reach a given target. To do so, we need to optimize for the sequence of skills to apply, the attention for each skill (i.e. find $\hat{U}^o \subseteq U^o$), as well as the latent subgoals for each skill (i.e. the exact value for each latent vector in $\hat{U}^o$). For our most simple approach, we run a three-level nested optimization: In the top-level, we exhaustively search over the combinations of skills to apply at each step, i.e. $k_1 \ldots, k_H$, where k_h is the index of the skill to apply at the high-level step h. We only keep the sequences that end with the same set cardinality as the goal by ensuring that $\sum_{h=1}^{H} M_{k_h} - N_{k_h} = N_g - N_o$, where M_{k_h} and N_{k_h} are the number of observation and goal components for the skill applied at step h and N_o and N_g are the number of components in the observed and target point clouds.

In the second-level optimization, we search over different attention structures. Assume that we have N_h components before applying skill k_h, i.e. $|U_h^o| = N_h$ and skill k_h takes K_h components as its observation. We can search over all $C_{N_h}^{K_h}$ combinations of attention structures.. For components not considered by the skill, its latent vector will remain the same at step h. Each choice of each skill attention yields an attention structure, as illustrated in Fig. 5.2(c). Since all latent subgoals u are initialized independently, we can filter out attention structures that are topologically identical. See the appendix for how we do this efficiently.

In the low-level optimization, for each attention structure, we follow the optimization in DiffSkill (Lin et al., 2022a). We first sample multiple initializations for all components u, where each node is initialized from our generative model. We can then perform gradient descent to further optimize the latent subgoals on the following objective:

$$\arg\min_{\mathbf{k},\mathbf{U}} C(\mathbf{k}, \mathbf{U}) = \prod_{h=1}^{H} f_{k_h}(U^{h-1}, U^h) \exp(-R(U^H, U^g)), \tag{5.1}$$

where $U^h = \{u_{h,1}, \ldots u_{h,N_h}\}$ are the latent subgoals at step h. $U^0 = U^o$ are the observed set while U^g are the goal set. Finally, we can use our policy to execute our plan by following each subgoal. A summary of our method can be found in Algorithm 2.

Algorithm 2 Planning with Spatial-Temporal Abstraction (PASTA)

Input: Demonstration Dataset D_{demo}, skill horizon T, planning horizon H, modules for neural skill abstraction π_k, f_k, r_θ, Point Cloud VAE with encoder ϕ, decoder ψ, prior p_u
Initialize modules for neural skill abstraction π_k, f_k, r_θ and PointFlow model
Generate N demonstration trajectories in differentiable physics
Train PointFlow model using all clustered point clouds $\{P_i\}$
Train set neural skill abstraction π_k, f_k, r_θ
for each valid skill sequence K_1 **to** K^H **do**
  for each valid attention structure U_1 **to** U^H **do**
    Initialize different latent subgoals $U^1, \dots U^H$ from p_u
    Optimize $U_1, \dots U_H$ according to Sec. 5.4.1
Choose skill sequence $\mathbf{k}$, attention structure $\mathbf{U}$ that minimizes $C(\mathbf{k}, \mathbf{U})$
for $h = 0$ **to** $H - 1$ **do**
  Decode the subgoal from U^h using the decoder ψ
  Execute policy π_{k_i} following the subgoal

5.5 Experiments

Our experiments are categorized into three parts: In Sec. 5.5.1, we describe the experimental setups and the baselines we consider. In Sec. 5.5.2 and Sec. 5.5.3, we show that PASTA outperforms the baselines and ablate different components in our framework. In Sec. 5.5.4, we demonstrate that PASTA can be effectively transferred to the real world without any fine-tuning.

5.5.1 Simulation Tasks and Baselines

Environment setups We consider several long-horizon dough manipulation tasks and divide them into two categories. First, we consider the three tasks from Diffskill (Lin et al., 2022a): LiftSpread, GatherTransport, and CutRearrange. These tasks require the agent to sequentially compose at most two skills to spread, cut or transport the dough. For these tasks, the number of entities in training and testing are the same. We further propose two new generalization tasks: CutRearrangeSpread (CRS) and CRS-Twice, where there are more entities during testing than during training..

Generalization tasks The CRS task provides a number of demonstration trajectories performing one of the three skills: Cutting with a knife, pushing with a pusher, and

spreading with a roller. The demonstration of each skill only shows a tool manipulating a single piece of dough. During testing, the agent needs to cut a dough into two, transport one piece to a spreading area with high table friction and then spread it. Generalization to more entities is required during the pushing and the spreading as there will be two entities in the scene , whereas during training of pushing and spreading there was only one entity. Can we do even more generalization? In the CRS-Twice task, we use the same agent trained on the CRS dataset and ask it to cut two pieces of dough from a chunk, transport both of them to a spreading area and spread them both. This is a 6-horizon task with up to 3 entities in the scene, much more complex than the skill demonstrations the agent is trained on. Due to the long-horizon nature of CRS-Twice, we specify the skill skeleton and use receding horizon planning for all the planning-based methods. See appendix for details.

Baselines We consider several baselines in simulation: First, gradient-based trajectory optimizer with oracle information (Traj-Opt), which can solve single-stage tasks for deformable object manipulation as shown in prior works (Huang et al., 2020). Second, model-free RL with Soft Actor Critic (Haarnoja et al., 2018) with RGB-D image input (SAC-Image). Third, SAC agent that takes in the dough, target dough, and tool point clouds as input, the same as our method (SAC-Point). Fourth is DiffSkill, model-based planning method from Lin *et al.* (Lin et al., 2022a), which takes RGB-D images as input and has no spatial abstraction. The last one is Flat 3D, which extends DiffSkill to use 3D point clouds as input, with a "flat" 3D representation that encodes the whole scene to a single latent vector without any spatial abstraction.

Metric Following DiffSkill, we report the normalized decrease in the Earth Mover Distance (EMD) approximated by the Sinkhorn diverge (Séjourné et al., 2019) computed as $s(t) = \frac{s_0 - s_t}{s_0}$, where s_0, s_t are the initial and current EMD. We additionally set a threshold for the score to determine the success of a trial.

5.5.2 Comparison with Baselines in Simulation

Table 5.1 shows the quantitative results of simulation tasks. First, we show that using a 3D representation is beneficial to planning and complex manipulation, as PASTA matches DiffSkill in LiftSpread and outperforms it in all the other tasks. Second, we highlight the compositional generalization power of PASTA in CRS and CRS-Twice, where the testing configuration differs vastly from training. In both environments, there exist additional

Table 5.1: Normalized improvement and success rate of all methods on two sets of tasks: tasks in DiffSkill and tasks that require generalization to more steps and entities. For CRS and CRS-Twice, training data only contains skills operating on **one** component of dough but at test time there are more than two components. Only the best performing baselines in CRS are evaluated on CRS-Twice. For LiftSpread and GatherMove, we consider the whole scene as a single spatial abstraction, so Flat 3D is equivalent to PASTA.

Method \ Task (Horizon)	DiffSkill tasks			Generalization tasks	
	LiftSpread (2)	GatherMove (2)	CutRearrange (3)	CRS (3)	CRS-Twice (6)
Traj-Opt (Oracle) (Huang et al., 2020)	0.818 / 40%	0.403 / 0%	0.511 / 20%	0.312 / 0%	0.227 / 0%
SAC-Image (Haarnoja et al., 2018)	0.797 / 0%	0.567 / 20%	0.103 / 0%	0.562 / 0%	0.365 / 0%
SAC-Point (Haarnoja et al., 2018)	0.796 / 0%	0.603 / 40%	0.147 / 0%	0.573 / 0%	0.353 / 0%
DiffSkill-Image (Lin et al., 2022a)	**0.920 / 100%**	0.683 / 60%	0.249 / 20%	-0.505 / 0%	-
Flat 3D (Ours)	*	*	0.797 / 60%	-0.712 / 0%	-
PASTA (Ours)	0.904 / **100%**	**0.715 / 100%**	**0.837 / 80%**	**0.896 / 100%**	**0.604 / 40%**

Ablation Method	Performance / Success
No Hard Negatives Feasibility	0.740 / 40%
No Sampling Planning	-0.455 / 0%
No Gradient Planning	0.329 / 0%
PASTA (Ours)	**0.837 / 80%**

Table 5.2: Ablation results from CutRearrange.

components of dough in the scene at test time, and in CRS-Twice, the planner needs to optimize a six-step sequence. Effectively using spatial abstraction to model the scene, PASTA achieves 100% success rate in CRS, and retains a good performance in CRS-Twice. All the baselines, especially the planning baselines (DiffSkill, Flat 3D) fail dramatically, as they can only produce plans that consist of scenes seen in training. Impressively, PASTA is the only approach that reaches non-zero success rate on these tasks that require compositional generalization.

5.5.3 Ablation analysis

Table 5.2 shows the quantitative performance of each ablation and PASTA in CutRearrange. First, we consider a variant of feasibility predictor's training, which removes the hard negative samples and only uses random negative sampling (No Hard Negatives), which halves the success rate. Second, we consider two variants of the planner, one without gradient-descent (No Gradient Planning) and one without sampling (No Sampling Planning). The results show that both components are crucial to planning. We provide more ablations on the

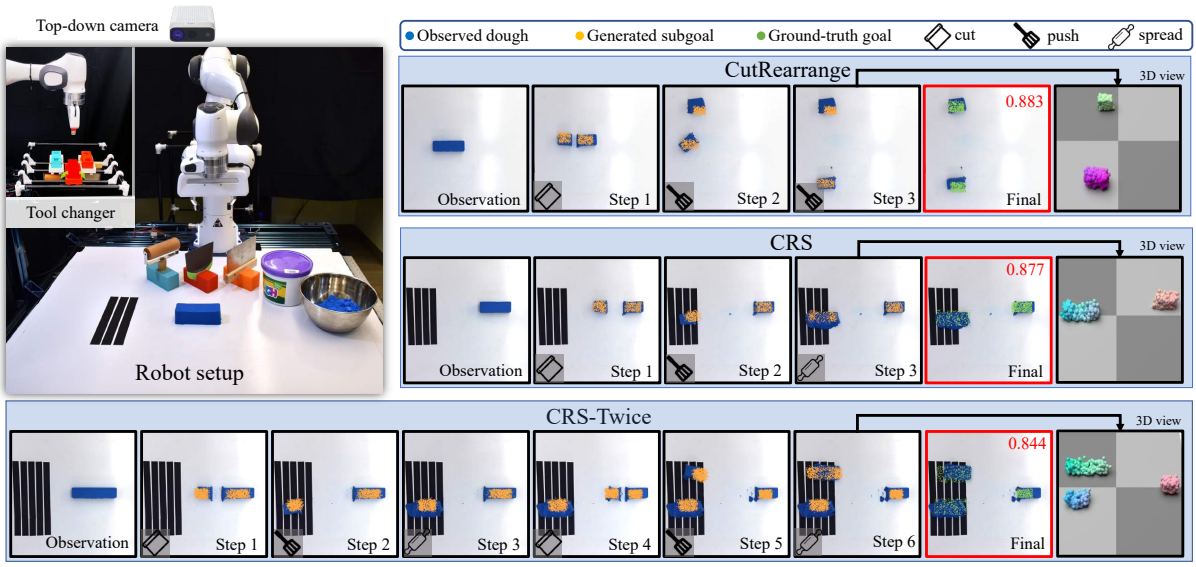


Figure 5.3: **Real world setup and execution with planned subgoals**. Our workspace consists of a Franka robot, a top-down camera, and a novel tool changer behind the robot that allows the robot to automatically switch tools. For each task, we show frames after executing a skill overlaid with the decoded point cloud subgoal; we report the final performance in red and overlay the ground truth target in green in the final frame. Additionally, we include a 3D view of the last generated subgoal to show the shape variations.

policy and the planner in the appendix.

5.5.4 Real World Experiments

Figure 5.3 shows our real world setup. We use a Franka robot with a top-down Azure Kinect camera capturing the RGB-D observation of the workspace. The robot is equipped with a tool station that allows an automatic change of tools. For real world "dough", we use Kinect Sand as a proxy because of its stable physical property. We transfer the feasibility predictor and reward predictor of PASTA directly from simulation and define heuristic controllers for the skills. For evaluation, we first generate a desired target point cloud and then reset the dough to its initial shape and record its point cloud. Next, given the current and the target point cloud, we use the planner to generate a sequence of skills and subgoals and then execute the plan with our controller. Finally, we record the achieved point cloud and report the normalized improvement EMD. We compare with the Flat3D method and also report performance of human on these tasks.

We evaluate on three of the simulation tasks: CutRearrange, CRS, and CRS-Twice. For each task we evaluate on the same four initial and target shapes for each method and report the performance in Table 5.3. Figure 5.3 shows the key frames from the execution of PASTA. We overlay the planned subgoals as well as the final goal for qualitative comparison. PASTA performs on-par with human in the real world, highlighting the robustness of our planner and the advantage of using 3D representation for sim2real transfer.

Method \ Task (Horizon)	CutRearrange (3)	CRS (3)	CRS-Twice (6)
Flat 3D	0.351 ± 0.478	0.007 ± 0.429	-
PASTA (Ours)	**0.836 ± 0.029**	**0.854 ± 0.016**	**0.795 ± 0.035**
Human	0.910 ± 0.014	0.863 ± 0.018	0.895 ± 0.013

Table 5.3: Normalized improvement on real world tasks. Each entry shows the mean and std of the performance over 4 runs. Flat 3D does not produce any meaningful plan for CRS, so we do not evaluate it on CRS-Twice.

5.6 Conclusions and Limitations

In this work, we propose a planning framework named PASTA that incoporates both spatial and temporal abstraction by planning with a 3D latent set representation with attention structure. We demonstrate a manipulation system in the real world that uses PASTA to plan with multiple tool-use skills to solve the challenging deformable object manipulation tasks, and we show that it significantly outperforms a flat 3D representation, especially when generalizing to more complex tasks.

Limitations: First, we rely on an unsupervised clustering method for entity decomposition and a point cloud VAE for mapping an observation to our latent set representation. This design is specific to the tasks we show and may not generalize to other tasks without retraining or parameter tuning. We hope that our framework can be incorporated in the future with self-supervised methods for learning the spatial abstraction. Second, manipulating deformable objects like dough is very challenging with a significant sim2real gap. This paper provides a starting point for planning with multiple tools towards this challenge; we hope our work can inspire more works in this exciting area.

Chapter 6

Discussion and Future Works

In this thesis, we tackle the challenges of manipulating deformable objects. These tasks challenge the common assumptions made in robotics, including low-dimension state representation, known dynamics and minimal occlusion. As such, we propose methods for learning structured world models for deformable objects from data, which are then used for planning. These data-driven models parameterized by neural networks and thus are flexible and allow us to model dynamics in different levels of details. At the same time, these dynamics models incorporate different inductive bias in a physical system, giving our models better generalization ability compared to models without any inductive bias.

Deformable object manipulation is an active area of research in robotics, with numerous exciting progress in recent years, expanding the capabilities of robots to various new tasks (Ha and Song, 2021; Xu et al., 2022; Viswanath et al., 2022; Lin et al., 2022b). Manipulating deformable objects have wide, direct application in domestic environments, such as giving care to the elderly, making food, doing laundry and more. But more importantly, these tasks force us to think about manipulation problems with fewer assumptions and thus build more general, autonomous manipulation systems that can work under partial observation, no accurate dynamics model and from high-dimension visual observation.

6.1 Future Directions

Below I list a few directions that are closely related to the contribution of this thesis:

Learning from Real Dynamics One of the advantages of learning dynamics model

from data is that, we can directly learn from real data, avoiding the sim2real gap of learning from simulation data. The challenges are getting supervision data from the high-dimension visual observation. The most common modality is video. While we can directly learn video prediction model (Ebert et al., 2018), we have shown in Chapter 3 that the lack of inductive bias limits its generalization. Without inductive bias, models will confuse the visual appearance from physics. A promising direction is to combine a learned structured dynamics model with a differentiable renderer, which enable us to train dynamics models directly from real world videos.

Adaptive Spatial Abstraction We can model the dynamics at different levels of details. In this thesis, we have modeled a graph-based dynamics (Chapter 3), an object-centric dynamics (Chapter 5) and a latent dynamics (Chapter 5). However, so far the spatial abstraction are manually designed based on the task. To build a truly general manipulation framework, we need to derive principles for making such choices. Alternatively, we can learn models that can learn to pick the right abstraction in an adaptive way, or learn spatial abstraction that can make smooth trade-offs between accuracy and computation.

Learning Policies with World Models While planning is a powerful method that can solve novel tasks given a sufficiently accurate dynamics model, it can be slow. For more dynamic tasks, learning policies directly is the more common approach (Ha and Song, 2021). My PhD study has also explored general methods for making policy learning more efficient (Lin et al., 2019a;b;c), But learning policies directly from high-dimension visual observation is difficult (Chapter 2). Prior works have shown success in combining the best of both worlds (Hafner et al., 2020). I believe learning policies on top of a structured representation which are also used for the dynamics model will help to significantly reduce the planning time for model-based approaches.

Appendix A

Details on SoftGym

A.1 Environment Details

A.1.1 Observation Space

Each task supports three types of observation space: Full state of the particles, reduced states and image based observation. For image-based observation, the agent receives an RGB image of the environment rendered by the Flex simulator, with a size $d \times d \times 3$, where d is a controllable parameter. For all our image-based experiments we choose $d = 128$. For the full state observation, the state is the positions of all particles, as well as any state of the action space or other rigid objects in the scene.

We now detail the reduced state representation for each task in SoftGym.

TransportWater: the reduced states are the size (width, length, height) of the cup, the target cup position, height of the water in the cup, amount of water inside and outside of the cup.

PourWater and PourWaterAmount: the reduced states are the sizes of both cups, the x, y-position and rotation of the controlled cup, the initial distance between the controlled cup and the current cup, the height of the water in the cup and the amount of water in both cups. For PourWaterAmount, we have an additional value indicating the amount of water to be poured.

Rope Environments: For rope enviornments, including the StraightenRope and RopeConfiguration, we pick 10 evenly-spaced keypoints on the rope, including the two end

points, and use the positions of these key points as the reduced state.

Cloth Environments: For all of the cloth related environments (SpreadCloth, FoldCloth(Crumpled), DropCloth, DropFoldCltoh), the reduced states are the positions of the four corners of the cloth.

For environments using any pickers or robots, the positions of the pickers or joint positions of the robot are included in the reduced state.

A.1.2 Action Space

For all environments, we normalize the action space to be within $[-1, 1]$ for the agents. Below we will describe the un-normalized action range for each environment, using meter or radian as the unit by default.

TransportWater: The motion of the cup is constrained to be in one dimension. The action is also in one dimension and is the increment of the the position of the cup along the dimension. The action range is $[-0.011, 0.011]$.

PourWater, PourWaterAmount: The action is $a = (dx, dy, d\theta)$, denoting the change of the position and rotation of the cup. $dx, dy \in [-0.01, 0.01]$ and $d\theta \in [-0.015, 0.015]$.

Picker: For the cloth and rope environments, we use either two pickers or one robot. A picker abstracts a controller which can pick and place objects. A picker is modeled as a sphere with a radius of 0.05. For each picker, the action is $a = (dx, dy, dz, d)$. $dx, dy, dz \in [-0.01, 0.01]$ indicate the change of the position of the picker. $d \in [0, 1]$ indicates the picking state of the picker. If $d > 0.5$, the particle closest to the picker will be attached to the picker and follow its movement. When $d < 0.5$, the picked particle will be released.

A.1.3 Task Variations

In this section we detail how we generate the task variations. Figure A.1 shows some of the task variations. Most of the practical tasks related to deformable object manipulation, such as laundry folding, require the agent to deal with variations of the objects in the environment, such as the size, shape, and physical properties. We summarize the variations of each task that we include in SoftGym in Table A.1.

PourWater, PourWaterAmount: For this task, both the controlled cup and the target cup are modeled as cuboids without the top face. We vary the height, length, and

Environment	Variations
TransportWater	cup size, water volume
PourWater	size of both cups, target cup position, water volume
PourWaterAmount	size of both cups, target cup position, water volume, goal volume
StraightenRope, RopeConfiguration	Initial rope configuration
SpreadCloth, FoldClothCrumpled	Cloth size, initial configuration
FoldCloth	Cloth size, random rotation of the inital configuration
DropCloth, DropFoldCloth	Cloth size, initial height

Table A.1: Different task variations in all tasks. Refer to the appendix for more details of the ranges of the variations and how they are generated.

width of both cups, the distance between the controlled cup and the target cup, as well as the volume of water in the controlled cup. The water is initially generated in a shape of cuboid. Denote the number of particles along each dimension of the water cuboid as l_w, w_w, h_w respectively. We vary the width w_w and the length l_w of the water cuboid in the range $[4, 13]$. For the height h_w, we first randomly select a water level between 'medium' and 'large'. Let $m = \min(w_w, l_w)$. For level 'medium', the height is $h_w = \lfloor 3.5m \rfloor$. For level 'large', the height is $h_w = 4m$. The total number of water particles is $v = w_w \cdot h_w \cdot l_w$.

Given the volume of water, we then create a cup for holding that amount of water. Denote the radius of the water particles as $r = 0.033$. The width and length of the controlled cup is $w_{cc} = w_w \cdot r + 0.1$ and $l_{cc} = l_w \cdot r + 0.1$, and the width and length of the target cup is $w_{tc} = w_w \cdot r + 0.07$ and $l_{tc} = l_w \cdot r + 0.07$. Let $h = v/((w_w + 1)(l_w + 1))$ be the number of particles in the height of the water cuboid. For medium volume of water, we have $h_{cc} = h \cdot r/2 + 0.001 \cdot \text{Unif}[-0.5, 0.5]$. For large volume of water, we have $h_{cc} = h \cdot r/3 + 0.001 \cdot \text{Unif}[0, 1]$. The height of the target cup is simply computed as $h_{tc} = h_{cc} + \text{Unif}[0, 0.1]$.

The distance between the controlled cup and target cup is sampled

$$m \cdot \text{Unif}[0.05m, 0.09m+](w_w + 4)r/2.$$

For PourWaterAmount, the goal volume is sampled from $0.1 + \text{Unif}[0, 1] \times 0.9$.

TransportWater: We vary the volume of water and size of cup in this task. The variation is generated almost exactly the same as in PourWater, with the following exceptions. For the medium volume of water, the cup height is computed as $h_{cc} = h \cdot r/2$, and for the large volume of water, the cup height is computed as $h_{cc} = h \cdot r/3 + 0.0015m$.

SpreadCloth, FoldClothCrumpled: We vary the size of the cloth. The cloth is

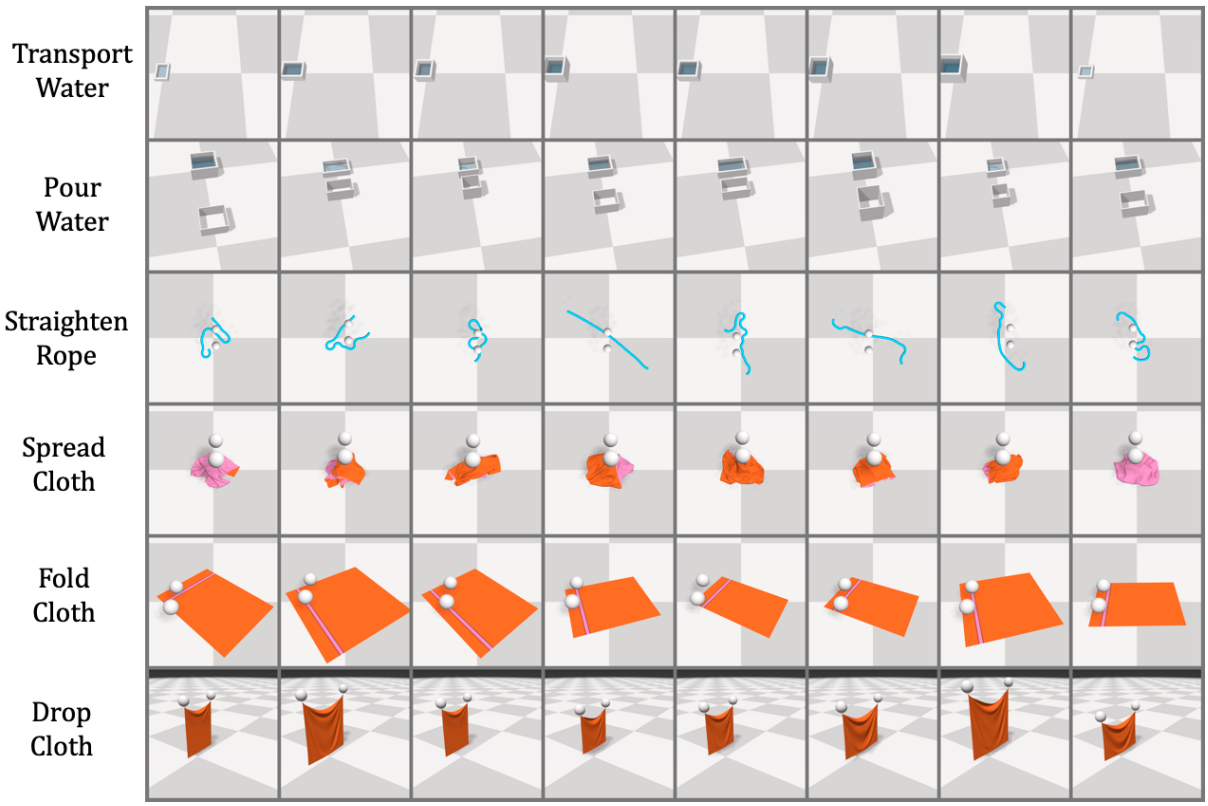


Figure A.1: Illustration of task variations. Each image shows the task after the initial reset. Variations of tasks in SoftGym-Hard are omitted here due to similarity to the ones shown.

modeled as a particle grid with width w and length l (the number of particles). We sample w and l from randint$(60, 120)$. We also vary the initial crumpled shape of the cloth. This is done by first randomly picking a particle on the cloth, lifting it up to a random height sampled from Unif$[0, 0.5]$, and then dropping it.

FoldCloth: In this task we only vary the size of the cloth. Similar to the SpreadCloth case, the width of the cloth w is sampled from randint$(60, 120)$. The initial state of the cloth is always flattened and centered at the origin.

DropCloth, DropFoldCloth: In this task we vary the size of the cloth in the same way in SpreadCloth. We lift the cloth up by picking up its two corners.

StraightenRope: In this task we use a rope with a fixed length and only vary its initial twisted shape. We generate different twisted shapes by randomly choosing a particle on the rope, picking it up, and then dropping it. We repeat this process for 4 times to

make sure the generated shapes are different.

A.1.4 Training and Evaluation

For computation efficiency, we pre-compute 1000 task variations and their initial states for each environment. Out of the 1000 task variations, 800 variations are used during training and 200 variations are used for evaluation.

Performance Metric Besides the reward, we compute a performance metric for each task at each time step, which is the same as the reward without any scaling. At the beginning of each episode, we compute an upper-bound and a lower-bound for the performance metric. For example, for SpreadCloth task, the performance is the covered area of the cloth and the upper-bound is when the cloth is flattened. For any task where the performance is a negative distance function, its upper-bound would be zero. For all tasks except StraightenRope, the lower-bound is the performance achieved at the first time step, which corresponds to the achieved performance when the policy does nothing. For StraightenRope, the lower bound is the possible minimal reward, which is the negative value of the straightened rope's length. Given the upper-bound and lower-bound u, l, we normalize the performance at each time step by

$$\hat{s} = \frac{s - l}{u - l},$$

where $\hat{s}$ is the normalized performance. The normalized performnace at the last time step is reported throughout the paper unless explicitly specified.

A.2 Algorithm Details

For all the tasks and algorithms, we use a discounting factor of $\gamma = 0.99$ when it applies. The action repetition and task horizon are summarized in table A.2.

A.2.1 CEM with Dynamics Oracle

For CEM, we use 10 optimization iteration. Model predictive control is used. Different planning horizon is used for different environments, as summarized in Table A.3. A total of 21K environment steps are used for making each decision. The number of candidate

Parameter	Transport Water	Pour Water	Straighten Rope	Spread Cloth	Fold Cloth	Drop Cloth
Action Repetition	8	8	8	8	8	32
Task Horizon	75	100	75	100	100	15

Table A.2: Action repetition and task horizon.

trajectories during each planning is thus $21K/planning\_horizon$. The top 10% candidates are selected as the elites for fitting the posterier distribution within each optimization iteration.

Parameter	Transport Water	Pour Water	Straighten Rope	Spread Cloth	Fold Cloth	Drop Cloth
Planning Horizon	7	40	15	15	30	15

Table A.3: Task specific planning horizon for CEM

A.2.2 SAC and CURL-SAC

We use the CURL-SAC implementation from the released code[1]. Both Q-value network and the policy network are MLPs with 2 hidden layers of 1024 neurons with ReLU as activation function. The hyper-parameters of SAC are summarized in Table A.4. To achieve learning stability, we tuned the reward scaling and learning rate for both SAC and CURL-SAC, for each environment. The parameters are summarized in Table A.5.

Table A.4: General hyper-parameters for SAC.

Parameter	SAC
batch size	128
initial steps	1000
replay buffer size	1e5
target smoothing coefficient	0.01
alpha	automatic tuning
delayed policy update period	2
target update interval	2

[1] https://github.com/MishaLaskin/curl

Table A.5: SAC task dependent hyper-parameters. If learning rate decay is applied, the actor learning rate is halved every 75K steps and the critic learning rate is halved every 100K steps.

Parameter	Transport Water	Pour Water	Straighten Rope	Spread Cloth	Fold Cloth	Drop Cloth
Reduced State						
learning rate	1e-3	1e-3	1e-3	1e-3	5e-4	1e-3
reward scaling	20	20	50	50	50	50
learning rate decay	-	-	yes	-	-	-
Image						
learning rate	3e-4	3e-4	3e-4	3e-4	1e-4	3e-4
learning rate decay	-	-	yes	yes	-	-
reward scaling	20	20	50	50	50	50

A.2.3 DrQ

We use the author released code[2] for the benchmarking with mostly default hyper-parameters. The only change in the hyper-parameter is that we use images of size 128×128 instead of 84×84 as in the released code, so we change the padding of the image from 4 to 6. We also tune the reward scaling parameter for different tasks, as summarized in Table A.5.

A.2.4 PlaNet

PlaNet takes the image observation as input. The image is first processed by a convolutional neural network to produce an embedding vector. The architecture of the encoding CNN is shown in table A.8. After the final convolution layer, the extracted features are flattened to be a vector, then transformed to an embedding vector by a linear layer. Different algorithms use different sizes for the embedding vector. For PlaNet and RIG, the size is 1024; For SAC and TD3, the size is 256. Different algorithms then process this embedding vector in different ways.

We use a GRU (Cho et al., 2014) with 200 hidden nodes as the deterministic path in the dynamics model. All functions are implemented as a two-layer MLP with 200 nodes for each layer and ReLU as the activation function. We refer to (Hafner et al., 2018) for

[2] https://github.com/denisyarats/drq

Table A.6: Hyper-parameters for PlaNet

Parameter	Value
training	
optimizer	Adam (Kingma and Ba, 2014)
learning rate	0.001
Adam ϵ	0.0001
experience replay size	10^6
explore noise	0.3
batch size	50
dynamics chunk size	50
free nats	3
CEM planning	
planning horizon	24
optimization iteration	10
candidate samples	1000
top candidate	100

layer	input channel	output channel	kernel size	stride
1	1024	128	5	2
2	128	64	5	2
3	64	32	5	2
4	32	16	6	2
5	16	3	6	2

Table A.7: Architecture of the deconvolutional neural network (VAE decoder) in PlaNet.

more details. We do not include the latent over-shooting in our experiment as it does not improve much over the one-step case.

During training, we first collect 5 episodes from a random policy to warm-up the replay buffer. Then, for each training epoch, we first store 900 time steps of experiences collected from the current planning agent and perform 100 gradient updates. The full hyper-parameters are listed in Table A.6.

On an Nvidia 2080Ti GPU with 4 virtual CPUs and 40G RAM, training PlaNet for 1M steps takes around 120 hours.

layer	input channel	output channel	kernel size	stride
1	3	16	4	2
2	16	32	4	2
3	32	64	4	2
4	64	128	4	2
5	128	256	4	2

Table A.8: Architecture of the encoding CNN.

A.2.5 Wu et al. 20

For the SpreadCloth and FoldCLoth task, we additionally compare to previous work (Wu et al., 2020) that learns a model-free agent for spreading the cloth from image observation. We take the official implementation from the authors [3]. Here, the action space is pick-and-place. Followed the approach in the paper, during exploration, a random point on the cloth is selected (with a heuristic method for cloth segmentation). The picker then goes to the picked location, picks up the cloth and moves to a place location given by the agent, waits for 20 steps and then drops the cloth. Default hyper-parameters in the original code are used.

A.3 CEM with Different Planning Horizons

We additionally evaluate CEM with different planning horizons for each task. The results are shown in Figure A.2. We see that the performance of CEM is sensitive to the planning horizon in TransportWater, FoldCloth and DropCloth, whereas the performance is relatively stable in the other tasks. The black bar is the performance that we report in the main chapter.

[3] https://github.com/wilson1yan/rlpyt

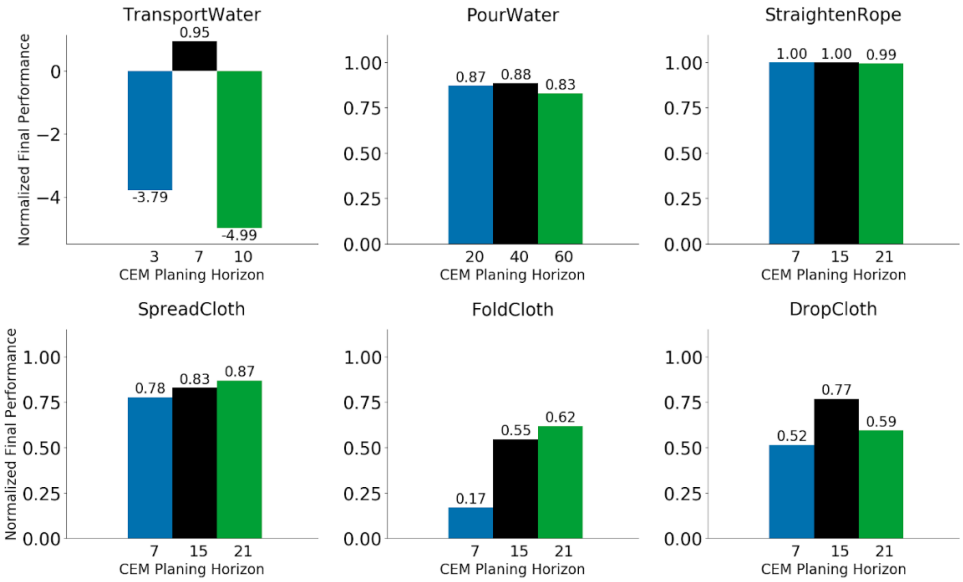


Figure A.2: Performance of CEM with different planning horizons for each task.

Appendix B

Details on Planning with Spatial Abstraction for Cloth Smoothing

Contents

B.1 VCD Implementation

B.1.1 GNN Architecture

As mentioned in the main chapter, we take the network architecture in previous work Sanchez-Gonzalez et al. (2020) (referred to as GNS) for our dynamics GNN G_{dyn} and the edge GNN G_{edge}. Both GNN consists of three parts: encoder, processor and decoder. Since the dynamics and edge GNNs have very similar architectures, we first describe the architecture of the dynamics GNN, and then describe how the edge GNN architecture differs from that.

Input: The input to the dynamics GNN is a graph, where the nodes are the points in the voxelized point cloud of the cloth, and the edges consist of the collision edges (built using Eq. (1)) and mesh edges (inferred by a trained edge GNN). The node feature for a point v_i consists of the concatenation of its past m velocities, a one-hot encoding of the point type (picked or unpicked - see details about picking in Section 3.4 in the main chapter and Section B.1.4 in the appendix), and the distance to the table plane. For edge e_{jk} that connects nodes v_j and v_k, its edge feature consists of the distance vector $(x_j - x_k)$, its norm $||x_j - x_k||$, a one-hot encoding of the edge type (mesh edge or collision edge), and the current displacement from the rest position $||x_j - x_k|| - r_{jk}$, where r_{jk} is the distance between x_j and x_k at the rest positions. The displacement from the rest positions are set to zero for collision edges which do not have rest positions.

We now describe how the robot action is incorporated into the input graph of the dynamics GNN as follows. As mentioned in Section 3.4 of the main chapter, when we

want to use the dynamics GNN to predict the effect of a pick-and-place robot action $a = \{a_{pick}, a_{place}\}$ on the current cloth, we first decompose the high-level action into a sequence of low-level movements, where each low-level movement is a small delta movement of the gripper and can be achieved in a short time. Specifically, we generate a sequence of small delta movements $\Delta x_1, ..., \Delta x_H$ from the high-level action, where $x_{pick} + \sum_{i=1}^{H} \Delta x_i = x_{place}$. Each delta movement Δx_i moves the gripper a small distance along the pick-and-place direction and the motion can be predicted by the dynamics GNN in a single step. We then incorporate the small delta movement into the input graph as follows. When the gripper is grasping the cloth, we denote the picked point as u. We assume that the picked point is rigidly attached to the gripper; thus, when considering the effect of the tth low-level movement of the robot gripper, we modify the input graph by directly setting the picked point u's position $x_{u,t} = x_{pick} + \sum_{i=1}^{t} \Delta x_i$ and velocity $\dot{x}_{u,t} = \Delta x_i / \Delta t$, where Δt is the time for one low-level movement step. The dynamics GNN will then propagate the effect of the robot action along the graph when predicting future states.

Encoder: The encoder consists of two separate multi-layer perceptrons (MLP), denoted as ϕ_p, ϕ_e, that map the node and edge feature, respectively, into latent embedding. Specifically, the node encoder ϕ_p maps the node feature for node v_i into the node embedding h_i, and the edge encoder ϕ_e maps the edge feature for edge e_{jk} into the edge embedding g_{jk}.

Processor: The processor consists of L stacked Graph Network (GN) blocks Battaglia et al. (2018) that update the node and edge embedding, with residual connections between blocks. We use $L = 10$ in both edge GNN G_{edge} and dynamics GNN G_{dyn}. The lth GN block contains an edge update MLP f_e^l and a node update MLP f_p^l that take as input the edge and node embedding g^l and h^l respectively and outputs updated embedding g^{l+1} and h^{l+1} (we denote g^0 and h^0 as the edge and node embedding output by the encoder). It also contains a global update MLP f_c^l that takes as input a global vector embedding c^l, and outputs the updated global embedding c^{l+1}. The initial global embedding c^0 is set to be 0. For each GN block, first the edge update MLP updates the edge embedding; it takes as input the current edge embedding g_{jk}^l, the node embedding h_j^l, h_k^l for the nodes that it connects, as well as the global embedding c^l: $g_{jk}^{l+1} = f_e^l(h_j^l, h_k^l, g_{jk}^l, c^l) + g_{jk}^l, \quad \forall e_{jk} \in E$. The node update MLP then updates the node embedding; its input consists of the current node embedding h_i^l, the sum of the updated edge embedding for the edges that connect to the node, and the global embedding c^l: $h_i^{l+1} = f_p^l(h_i^l, \sum_j g_{ji}^{l+1}, c^l) + h_i^l, \quad \forall i = 1, ..., N_p$. Note the edge and node updates both have residual connections between consecutive

blocks. Finally, the global update MLP takes as input the current global embedding c^l, the mean of the updated node and edge embedding, and updates the global embedding as: $c^{l+1} = f_c^l(c^l, \frac{1}{|V|}\sum_{i=1}^{|V|} h_i^{l+1}, \frac{1}{|E|}\sum_{e_{jk}} g_{jk}^{l+1})$.

Decoder: The decoder is an MLP ψ that takes as input the final node embedding h_i^L output by the processor for each point v_i; the decoder outputs the acceleration for each point: $\ddot{x}_i = \psi(h_i^L)$. The acceleration can then be integrated using the Euler method to update the node position x_i. We train the graph GNN G_{dyn} using the L2 loss between the predicted point acceleration $\ddot{x}_i$ and the ground-truth acceleration obtained by the simulator; see Sec. B.1.2 for details.

Edge GNN: The edge GNN G_{edge} has nearly the same architecture as the dynamics GNN, with the following differences: first, the input graph to the edge GNN encoder consists of only the voxelized point cloud and the collision edges $\langle P, E^C \rangle$; the edge GNN aims to infer which collision edges are also mesh edges. The node feature is 0 for all nodes. The edge feature for edge e_{jk} consists of the distance vector $(x_j - x_k)$ and its norm $||x_j - x_k||$ (without the edge type, since this must be inferred by the edge GNN). The processor is exactly the same as that in the dynamics GNN. The decoder is an MLP that takes as input the final edge embedding output by the processor and outputs the probability of the collision edge being a mesh edge. We use a binary classification loss on the prediction of the mesh edge for training.

Hyperparameters In simulator, we set the radius of particles to be 0.00625, an All MLPs that we use has three hidden layers with 128 neurons each and use ReLU as the activation function. The detailed parameters of the GNN architecture, as well as the simulator parameters, can be found in Table B.1.

B.1.2 VCD Training Details

Training in Simulation: We trian the dynamics GNN with one-step prediction loss: suppose that we sample a transition (V_t, a_t, V_{t+1}), where a_t is a low-level action. Then we assign the velocity at timestep t that is input to the network to be the ground-truth velocity obtained from the simulator (after matching the points to their corresponding simulator particles). This strategy enables us to sample arbitrary timesteps for training rather than needing to always simulate the dynamics from the first timestep.

For training the edge GNN, we need to obtain the ground-truth of which collision edges

are also mesh edges. During simulation training, a collision edge is assumed to be a mesh edge if the mapped simulation particles of the edge's both end points are connected by a spring in the simulator.

We train our dynamics GNN with the ground-truth mesh edges, and directly use it with the mesh edges predicted by the edge GNN at test time. We find this to work well without fine-tuning the dynamics GNN on mesh edges predicted by the edge GNN, due to the high prediction accuracy(91%) of the edge GNN.

Bipartite Graph Matching: As mentioned in the main chapter, we need bi-partite graph matching to find a mapping from the voxelized point cloud to the simulation particles, in order to obtain the state and connectivity of the voxelized point cloud for training the dynamics and edge GNN. Given N points in the voxelized point cloud $p_i, i = 1 \dots N$ and M simulated particles of the cloth in simulation $x_j, j = 1 \dots M$, the goal of the bipartite graph matching here is to match each point in the point cloud to a simulated particles. The simulated cloth mesh is downsampled by three times to improve computation efficiency, e.g., a cloth composed of 40×40 particles is downsampled to be of size 13×13. The bi-partite matching is only performed on the downsampled particles. We build the bipartite graph by connecting an edge from each p_i to x_j, with the cost of the edge being the distance between the two points. In our experiments, we always have $M > N$ since we use a large grid size for the voxelization.

Training Data: We collect 2000 trajectories, each consisting of 1 pick-and-place action. The pick point is randomly chosen among the locally highest points on the cloth; this is only done to generate the training data for the dynamics model, not for planning (we do this for training the VSF and CFM baselines as well; the MVP baseline uses the behavioral policy to generate its training data). The unnormalized direction vector $p = (\Delta x, \Delta y, \Delta z)$ for the pick-and-place action is uniformly sampled as follows: $\Delta x, \Delta z \in [-0.5, 0.5]$, $\Delta y \in [0, 0.5]$. The direction vector is then normalized and the move distance is sampled uniformly from $[0.15, 0.4]$. The high-level pick-and-place action is decomposed into 100 low-level steps: the pick-and-place is executed in the 60 low-level actions, and then we wait 40 steps for the cloth to stablize. We train our dynamics model in terms of low-level actions.

We choose the voxel size (0.0216) to be three times of the particle radius (0.00625) to keep it consistent with the downsampled mesh. The neighbor radius, which determines the construction of collision edges, is set to be roughly two times of the voxel size, so as to ensure that particles in adjacent voxels are connected.

Model parameter	Value
Encoder(same for both node encoder and edge encoder)	
number of hidden layers	3
size of hidden layers	128
Processor	
number of message passing steps	10
number of hidden layers in each edge/node update MLP	3
size of hidden layers	128
Decoder	
number of hidden layers	3
size of hidden layers	128
Training parameters	**Value**
learning rate	0.0001
batch size	16
training epoch	120
optimizer	Adam
beta1	0.9
beta2	0.999
weight decay	0
Others	**Value**
dt	0.05 second
particle radius	0.00625 m
downsample scale	3
voxel size	0.0216 m
neighbor radius R	0.045 m

Table B.1: Summary of all hyper-parameters.

Training Parameters: We use Adam Kingma and Ba (2014) with an initial learning rate of 0.0001 and reduce it by a factor of 0.8 if the training plateaus. We train with a batch size of 16. The training of the dynamics GNN takes roughly 4 days to converge on a RTX 2080 Ti. The training of the edge GNN usually converges in 1 or 2 days. Detailed training parameters can be found in Table B.1.

B.1.3 Shape-Specific Training Details

Although VCD performs decently under partial observability, we found dynamics model trained on full mesh model usually converges faster and obtains better asymptotic performance. This is well expected since incomplete information caused by self-occlusion results

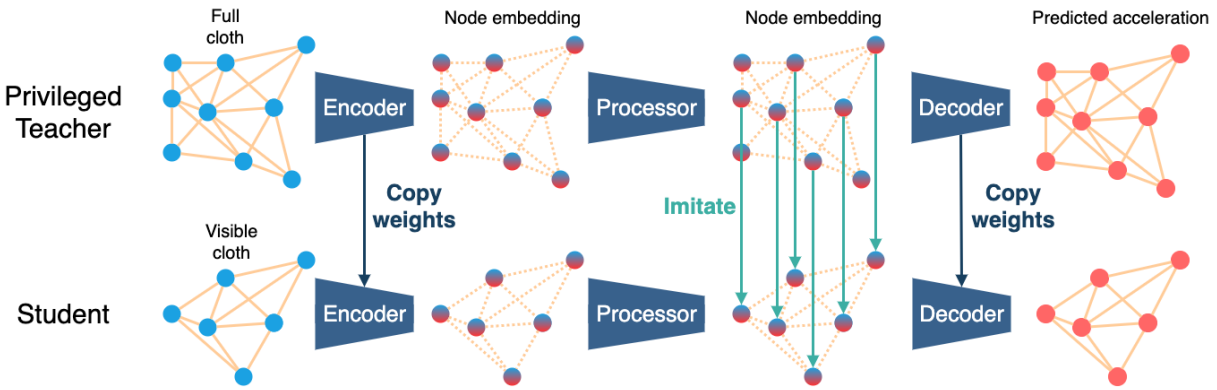


Figure B.1: A graphical illustration of privileged graph imitation learning. The privileged teacher has the same model architecture as student, but takes full cloth as input. Following Lee et al. (2020a), we initialize the encoder and decoder of student model by weights of pretrained teacher. Then we freeze the teacher and transfer the privileged information by matching the node embedding and global embedding of two models. The target nodes to imitate are obtained by bi-partite matching as described in B.1.2.

in ambiguity of state estimation.

Therefore, we introduce shape-specific training to inject prior knowledge of the full cloth into the dynamics model. The prior encodes structure of the full cloth and incentivize the model to reason about occlusion implicitly.

Auxiliary Reward Prediction Following Jaderberg et al. (2016), we additionally train our dynamics model to predict reward in order to regularize the model. The groundtruth reward, which is the coverage of cloth after one time step, is calculated by approximation as described in B.1.4. The coverage is calculated over all particles, thus it provides information from a global view to the model. The reward model is a three-layer MLP which takes global embedding c^L as input. We use mean square error to train the model.

It should be noted that at test time, we still use the heuristic reward function, which models particles as spheres and calculate an approximate coverage on the partial point cloud. Although the learned reward model predicts a global reward, which theoretically will take into the newly revealed occluded regions into account, we found it perform slightly worse than the heuristic reward function.

Privileged Graph Imitation Learning The second technique we introduce is privileged graph imitation learning, where we train a student model which takes partial point cloud as input, to imitate a privileged agent which has access to privileged information.

We hope the student to learn a recover function that recovers true states from partial information. A visual illustration is shown in Figure B.1.

To do so, we first train a privileged agent with all simulated particles(including the occluded ones) and ground-truth mesh edges. The privileged teacher model shares identical architecture as the student model, but with complete information. We train the teacher model with acceleration loss and the auxiliary reward prediction loss.

Graph-based imitation learning is not straightforward because the graphs of two models have very different structures. Typically, the graph of teacher model will have more vertices since it can observe occluded particles while the student only observes the voxelized partial point cloud. To tackle with this challenge, we conduct bipartite matching to match student nodes with teacher nodes as described in B.1.2.

Once we have the node correspondence, we retrieve the intermediate node features of both teacher and student model, h_T^L and h_S^L, and force the node feature of student h_S^L to be similar to h_T^L. The final output is still supervised by groundtruth acceleration. We copy the weights of encoder and decoder from teacher model to initialize student since we find it accelerate training. The teacher is frozen during imitation learning. By imitating the intermediate node features, we provide high capacity training signal to the student to recover groundtruth acceleration by proper message passing. To successfully imitate the teacher, the student have to conduct occlusion reasoning to some extent, and take the effects of occluded particles and erroneous mesh edges into account. In addition to node features, the student model also mimic the global embedding of teacher model to make more accurate reward prediction. We use mean square error for imitation learning.

B.1.4 VCD Planning Details

We sample K high-level pick-and-place actions. For each sampled high-level action, we roll out our dynamics model using that action for H low-level steps and obtain the sequence of predicted point positions.

Action sampling during planning in simulation As described in the main text, we sample 500 pick-and-place actions, where the pick point is first uniformly sampled from a bounding box of the cloth and then projected to be on the cloth mask. For generating the bounding box, we first obtain the cloth mask from the simulator. We then obtain the minimal and maximal pixel coordinates u, v value of the cloth mask. The

bounding box is the rectangle with corners $(min(u) - padding, min(v) - padding)$ and $(max(u)+padding, max(v)+padding)$, where padding is set to be 30 pixels for the 360×360 image size we use. We use rejection sampling to make sure the place point is within the image to keep the action within the depth camera view. The unnormalized direction vector $p = (\Delta x, \Delta y, \Delta z)$ (y is the up axis) of the pick-and-place is uniformly sampled as follows: $\Delta x, \Delta z \in [-0.5, 0.5]$, and $\Delta y \in [0, 0.5]$. The vector is normalized and then the distance is separately sampled from $[0.05, 0.2]$ meters. We decompose the pick-and-place action into 10 low-level actions and wait for another 6 steps for the cloth to stabilize.

Action sampling during planning in the real world The robot action space is pick-and-place with a top down pinch grasp. For each action, we sample 100 pick-and-place actions to be evaluated by our model. Each action sample is generated as follows: We first sample a pick-point location corresponding to the segmented cloth, denoted as (p_x, p_y). We then generate a random direction $\theta \in [0, 2\pi]$ and distance $l \in [0.02, 0.1]$ meters. Then the place point will be $(p_x + l\cos\theta), p_y + l\sin\theta$. We only accept an action if both the pick and the place points are within the work space of the robot. We additionally filter out actions whose place points are overlapping with the cloth. This heuristic saves computation time without sacrificing performance.

Reward computation in planning: As described in the main text, to compute the reward function r for planning, we treat each node in the graph as a sphere with radius R and compute the covered area of these spheres when projected onto the ground plane. To prevent the planner from exploiting the model inaccuracies, we do the following: if the model predicts that there are still points above a certain height threshold after executing the pick-and-place action and waiting the cloth to stabilize, then the model must be predicting inaccurately and we set the reward of such actions to 0. The threshold we use is computed as 15×0.00625 meters, where 0.00625 is the radius of the cloth particle used in the simulation.

B.2 Baselines Implementation

For all the baselines, we try our best to adjust the SoftGym cloth environment to match the cloth environment used in the original papers. For VSF, we place the camera to be top-down and zoomed in so that the cloth covers the entire image when fully flattened. We also changed the color of the cloth to be bluish as in the original paper. We collect 7115 trajectories, each consisting of 15 pick-and-place actions for training the VSF model (same

as in the VSF paper). For CFM, we also use a top-down camera and change the color of the cloth to be the same on both sides, following the suggestion of the authors (personal communication). We collect 8000 trajectories each consisting of 50 pick-and-place actions for training the contrastive forward model (same as in the CFM paper). For MVP, we collect 5000 trajectories each with 50 pick-and-place actions and report the performance of the best performing model during training. We trained each of the baselines for at least as many pick-and-place actions as they were trained in their original papers. For training our method, VCD, we collect 2000 trajectories, each consisting of 1 pick-and-place action decomposed into 20 low-level actions for training. Note that this is fewer pick-and-place actions than any of the baselines used for training. We now describe each compared baseline in more details below:

B.2.1 VisuoSpatial Forsight (VSF)

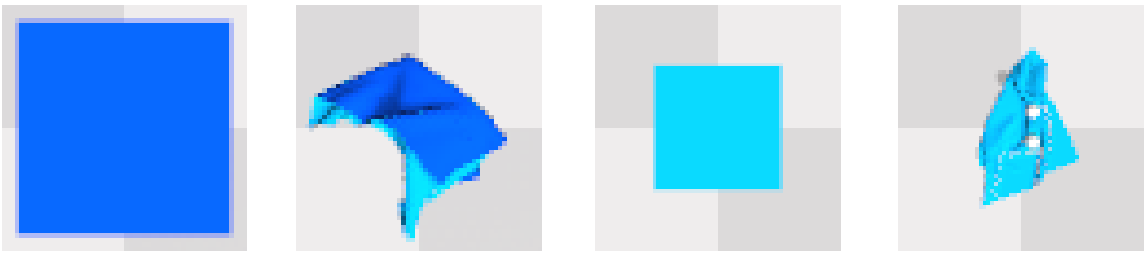
We use the official code of VSF provided by the authors[1].

Image: Following the original paper, we use images of size 56×56. we place the camera to be top-down and zoomed in so that the cloth covers the entire image when fully flattened. An example goal image of the smoothed cloth for VSF is shown in Figure B.2.

Training data & Procedure: For training the VSF model, we collect 7115 trajectories for training (same as in the VSF paper), each consisting of 15 pick-and-place actions. Following the VSF paper, the pick-and-place action first moves the cloth up to a fixed height, which is set to be 0.02 m in our case, and then moves horizontally. The horizontal movement vector is sampled from $[-0.07, 0.07] \times [-0.07, 0.07]$ m. This range is smaller than what is used for VCD, as we follow the original paper to set the maximal move distance roughly half of the cloth/workspace size. We use rejection sampling to ensure the after the movement, the place point is within the camera view. Similar to VCD, the pick point is uniformly sampled among the locally highest points on cloth (only during training). It takes 2 weeks for VSF to converge on this dataset.

Action sampling during planning: Similarly to VCD, the pick point is sampled uniformly from a bounding box around the cloth and then projected to the cloth mask. The padding for the bounding box here we use is 6. Other than the pick point, other elements of the pick-and-place action is sampled following the exact same distribution as in the training

[1] https://github.com/ryanhoque/fabric-vsf



(a) VSF goal image. (b) VSF observation. (c) CFM goal image. (d) CFM observation.

Figure B.2: Images of cloth configurations used by the baseline methods.

data collection.

B.2.2 Contrastive Forward Model (CFM)

We use the official code of CFM provided by the authors[2].

Image: Following the original paper, we use images of size 64×64. We also place the camera to be top-down and adjust the camera height so the cloth contains a similar portion of the image as in the original paper. Following the suggestions from the authors (personal communication), we also set the color of the cloth to be the same on both sides. See Figure B.2 for an example of the images we use.

Training data: For training, we collect 8000 trajectories each consisting of 50 pick-and-place actions, which is the same as in the original paper. Similar to VCD and VSF, the pick point is sampled among the locally highest points on the cloth (only during training). The movement vector is sampled from $[-0.04, 0.04] \times [0, 0.04] \times [-0.04, 0.04]$ m, where the y-axis is the negative gravity direction. We use pick-and-place actions with such small distances following the original paper. We also use rejection sampling to ensure the place point is within the camera view.

Action sampling during planning: Similar to VCD, the pick point is sampled uniformly from a bounding box around the cloth and then projected to the cloth mask. The padding size here we use for the bounding box is 5. Other than the pick point, other elements of the pick-and-place action are sampled following the exact same distribution as in training data collection.

[2]https://github.com/wilson1yan/contrastive-forward-model

B.2.3 Maximal Value under Placing (MVP)

We use the official code of MVP provided by the authors[3].

Image: Following the original paper, we use images of size 64×64. We also place the camera to be top-down.

Training data: For training, we collect 8000 trajectories each consisting of 50 pick-and-place actions, which is the same as the original paper. However, the Q function starts to diverge after 5000 trajectories and the performance starts to drop. Thus we report the best policy performance when it has been trained for 5000 trajectories. This corresponds to around 15000 training iterations.

Action space: The action space for the MVP policy is in 5 dimension:

$$(u, v, \Delta x, \Delta y, \Delta z)$$

, where u, v is the image coordinate of the pick point and is sampled for the segmented cloth pixel. We use the depth information to back project the pick point to 3d space. $(\Delta x, \Delta y, \Delta z)$ is the displacement of the place location relative to the pick point and is clipped to be within 0.5. Additionally, the height Δy is clipped to be non-negative.

B.3 Details on Experiments

B.3.1 Simulation Setup

We use the Nvidia Flex simulator, wrapped in SoftGym Lin et al. (2020), for training. In SoftGym, the robot gripper is modeled using a spherical picker that can move freely in 3D space and can be activated so the nearest particle will be attached to it. For the simulation experiments, we use a nearly square cloth, composed of a variable number of particles sampled from $[40, 45] \times [40, 45]$; this corresponds to a cloth of size in the range of $[25, 28] \times [25, 28]$ cm. Detailed cloth parameters such as stiffness are listed in the appendix. For all methods, we randomly generate 20 initial cloth configurations for training. The initial configurations are generated by picking the cloth up and then dropping it on the table in simulation. For evaluation, we consider three different geometries: 1) the same

[3]https://github.com/wilson1yan/rlpyt

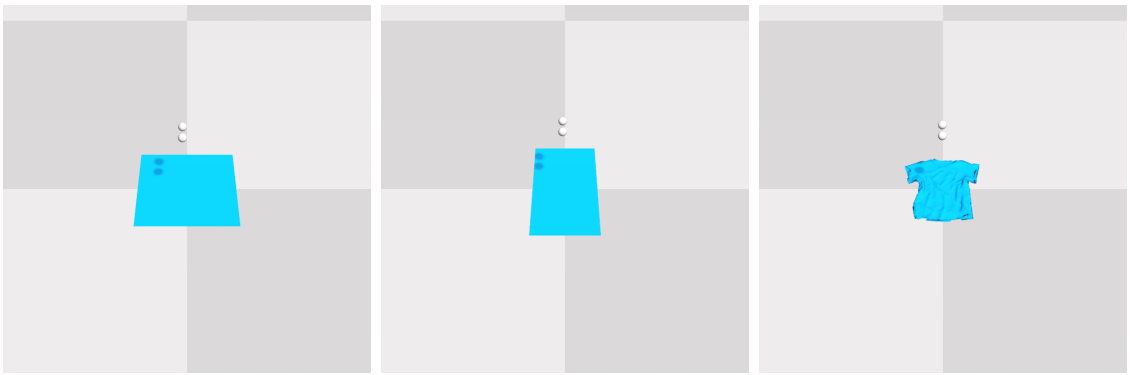


Figure B.3: Images of the square cloth, rectangular cloth, and t-shirt used in simulation.

type of square cloth as used in training; 2) Rectangular cloth. The length and width of the rectangular cloth is sampled from $[19, 21] \times [31, 34]$ cm. 3) T-shirt. Images of these three shapes of cloth in simulation are shown in Figure B.3.

We set the stiffness of the stretch, bend, and shear spring connections to $0.8, 1, 0.9$, respectively.

B.3.2 Real-world Setup

Our real robot experiments use a Franka Emika Panda robot arm with a standard panda gripper. We obtain RGBD from a side view Azure Kinect camera and crop the RGBD image into the size of [345, 425], which corresponds to a workspace of 0.4 x 0.5 meters. To obtain the cloth point cloud, we first use color thresholding to remove the table background and obtain the cloth segmentation mask and then back project each cloth pixel to 3d space using the depth information. We evaluate on three pieces of cloth: Two squared towels made of silk and cotton respectively and one shirt made of cotton. We use the covered area as described in Sec. B.1.4 as our reward function.

For each cloth, we evaluate 12 trajectories each with a maximum of 20 pick-and-place actions. For each trajectory, the robot stops if the normalized performance is higher than 0.95 or if the predicted rewards of all the sampled actions are smaller than the current reward. For each trajectory, we reset the cloth configuration using the following protocol: Each time, the arm picks a random point on the cloth, lifts it up to 0.4 meters above the table and drop it at a fixed point on the table. This procedure is done three times in the beginning of each trajectory.

B.3.3 Details on Robot Experiments

Running Time In average, it takes 12.7 seconds for VCD to plan each pick-and-place action (100 samples) on 4 RTX 2080Ti and 10.2 seconds for Franka to execute the action. With additional communication overhead, our current system takes around 40 seconds for computing and executing each pick-and-place action.

Normalized Coverage (NC) of Robot Experiments For the robot experiments, the main text reports the normalized improvement (NI). NC are reported here in Table B.2.

Material \ # of pick-and-place actions	5	10	20	Best
Cotton Square Cloth	0.690 ± 0.166	0.884 ± 0.293	0.959 ± 0.193	0.959 ± 0.080
Silk Square Cloth	0.744 ± 0.180	0.876 ± 0.314	0.964 ± 0.075	0.964 ± 0.054
Cotton T-Shirt	0.548 ± 0.114	0.601 ± 0.093	0.688 ± 0.068	0.773 ± 0.141

Table B.2: Normalized coverage (NC) of VCD in the real world.

B.4 Additional Visualizations

B.4.1 Simulation Experiments

Normalized Improvement (NI) and Normalized Coverage (NC) in Simulation NI and NC of our simulation experiments are reported in Table B.3 and Table B.4. With different metrics, our method consistently outperforms all baselines.

B.4.2 Visualization of Edge GNN

We compare predictions of our edge prediction model with the ground-truth edges used for training the edge model in Figure B.4. As shown, the edge GNN prediction reasonably matches the ground-truth, and thus well captures the cloth structure; it can also correctly disconnect the top layer from the bottom layer when the cloth is folded, e.g., the top left part of the first example and the bottom right part of the second example. Note our method uses only the point cloud as input and the color in this figure is only used for visualization. The edge GNN is trained on the same dataset as the dynamics GNN (described in Sec. B.1.2), and on the validation set, it achieves a prediction accuracy of 0.91.

Table B.3: Normalized Improvement (NI) of all methods in simulation, for varying numbers of allowed pick and place actions.

	Method \ # of pick-and-place actions	5	10	20	50
Square	VCD (Ours)	0.648 ± 0.174	0.770 ± 0.206	0.969 ± 0.260	0.989 ± 0.027
	VCD-shape-specific (Ours)	$\mathbf{0.765 \pm 0.252}$	$\mathbf{0.964 \pm 0.285}$	$\mathbf{0.991 \pm 0.038}$	$\mathbf{0.999 \pm 0.031}$
	VSF Hoque et al. (2020)	0.338 ± 0.107	0.573 ± 0.165	0.792 ± 0.116	0.960 ± 0.022
	CFM Yan et al. (2020)	0.034 ± 0.047	0.059 ± 0.060	0.096 ± 0.080	0.172 ± 0.147
	MVP Wu et al. (2020)	0.399 ± 0.210	0.435 ± 0.137	0.421 ± 0.361	0.307 ± 0.310
Rectangular	VCD (Ours)	0.557 ± 0.248	0.787 ± 0.193	0.969 ± 0.170	1.000 ± 0.029
	VCD-shape-specific (Ours)	$\mathbf{0.692 \pm 0.239}$	$\mathbf{0.891 \pm 0.247}$	$\mathbf{0.992 \pm 0.060}$	$\mathbf{1.000 \pm 0.025}$
	VSF Hoque et al. (2020)	0.298 ± 0.138	0.389 ± 0.186	0.576 ± 0.195	0.722 ± 0.336
T-shirt	VCD (Ours)	$\mathbf{0.559 \pm 0.206}$	$\mathbf{0.726 \pm 0.454}$	$\mathbf{0.956 \pm 0.595}$	$\mathbf{0.985 \pm 0.031}$
	VCD-shape-specific (Ours)	0.319 ± 0.260	0.166 ± 0.231	0.132 ± 0.329	-0.222 ± 0.600
	VSF Hoque et al. (2020)	0.046 ± 0.106	0.118 ± 0.062	-0.049 ± 0.244	-0.256 ± 0.425

Table B.4: Normalized coverage (NC) of all methods in simulation on the regular cloth, for varying numbers of allowed pick and place actions.

	Method \ # of pick-and-place actions	5	10	20	50
Square	VCD (Ours)	0.795 ± 0.100	0.864 ± 0.125	0.982 ± 0.143	0.994 ± 0.014
	VCD-shape-specific (Ours)	$\mathbf{0.853 \pm 0.143}$	$\mathbf{0.977 \pm 0.163}$	$\mathbf{0.994 \pm 0.024}$	$\mathbf{1.000 \pm 0.018}$
	VSF Hoque et al. (2020)	0.638 ± 0.071	0.777 ± 0.091	0.888 ± 0.085	0.979 ± 0.011
	CFM Yan et al. (2020)	0.445 ± 0.052	0.466 ± 0.044	0.494 ± 0.031	0.538 ± 0.044
	MVP Wu et al. (2020)	0.667 ± 0.121	0.667 ± 0.124	0.661 ± 0.194	0.609 ± 0.179
Rectangular	VCD (Ours)	0.765 ± 0.134	0.885 ± 0.092	0.984 ± 0.087	1.000 ± 0.013
	VCD-shape-specific (Ours)	$\mathbf{0.868 \pm 0.139}$	$\mathbf{0.942 \pm 0.130}$	$\mathbf{0.996 \pm 0.031}$	$\mathbf{1.000 \pm 0.014}$
	VSF Hoque et al. (2020)	0.644 ± 0.087	0.675 ± 0.113	0.771 ± 0.132	0.855 ± 0.155
T-shirt	VCD (Ours)	$\mathbf{0.832 \pm 0.084}$	$\mathbf{0.885 \pm 0.134}$	$\mathbf{0.985 \pm 0.184}$	$\mathbf{0.995 \pm 0.014}$
	VCD-shape-specific (Ours)	0.743 ± 0.076	0.707 ± 0.115	0.689 ± 0.103	0.573 ± 0.177
	VSF Hoque et al. (2020)	0.643 ± 0.090	0.686 ± 0.079	0.613 ± 0.088	0.558 ± 0.071

B.4.3 Visualizations of Planned Actions in Simulation

Figure B.5 shows three planned pick-and-place action sequences of VCD in simulation. As shown, VCD successfully plans actions that gradually smooths the cloth. We observe note that VCD favours picking edge / corner points and pulling outwards, which is an effective smoothing strategy, demonstrating the effectiveness of VCD for planning.

B.4.4 Visualizations of Open-loop Predictions in Simulation

In order to understand better what our model is learning, we visualize the prediction of our model compared to the simulator output in Figure B.6, B.7, B.8. Given a pick-and-place

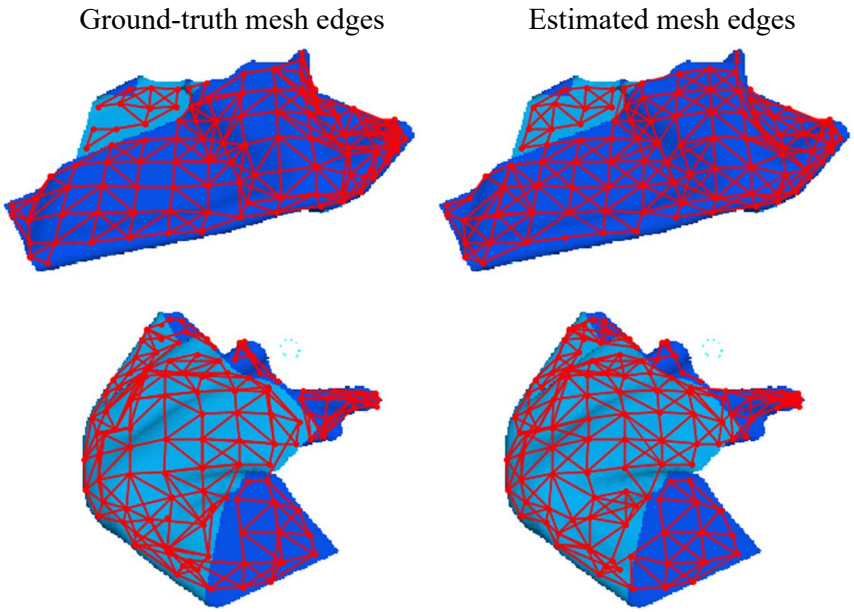


Figure B.4: The edge prediction result of our edge GNN. Red lines visualize the ground-truth (left) or inferred (right) mesh connections.

action decomposed into 75 low-level actions, the model is given the 5th point cloud in the trajectory with the past 4 historical velocities, and the dynamics model is used to generate the future predictions. As shown, even if the prediction horizon is as long as 70 steps, VCD is able to give relatively accurate predictions on all cloth shapes, indicating the effectiveness of incorporating the inductive bias of the cloth structure into the dynamics model.

B.4.5 Visualization of Sampled Actions in The Real World

We show in Figure B.9 VCD's predicted score for each of the sampled action during smoothing of the cloth. Interestingly, though there is no explicit optimization for this, VCD favours picking corner or edge points and pulling outwards, which is a very natural and effective strategy for smoothing. This demonstrates the effectiveness of VCD for planning.

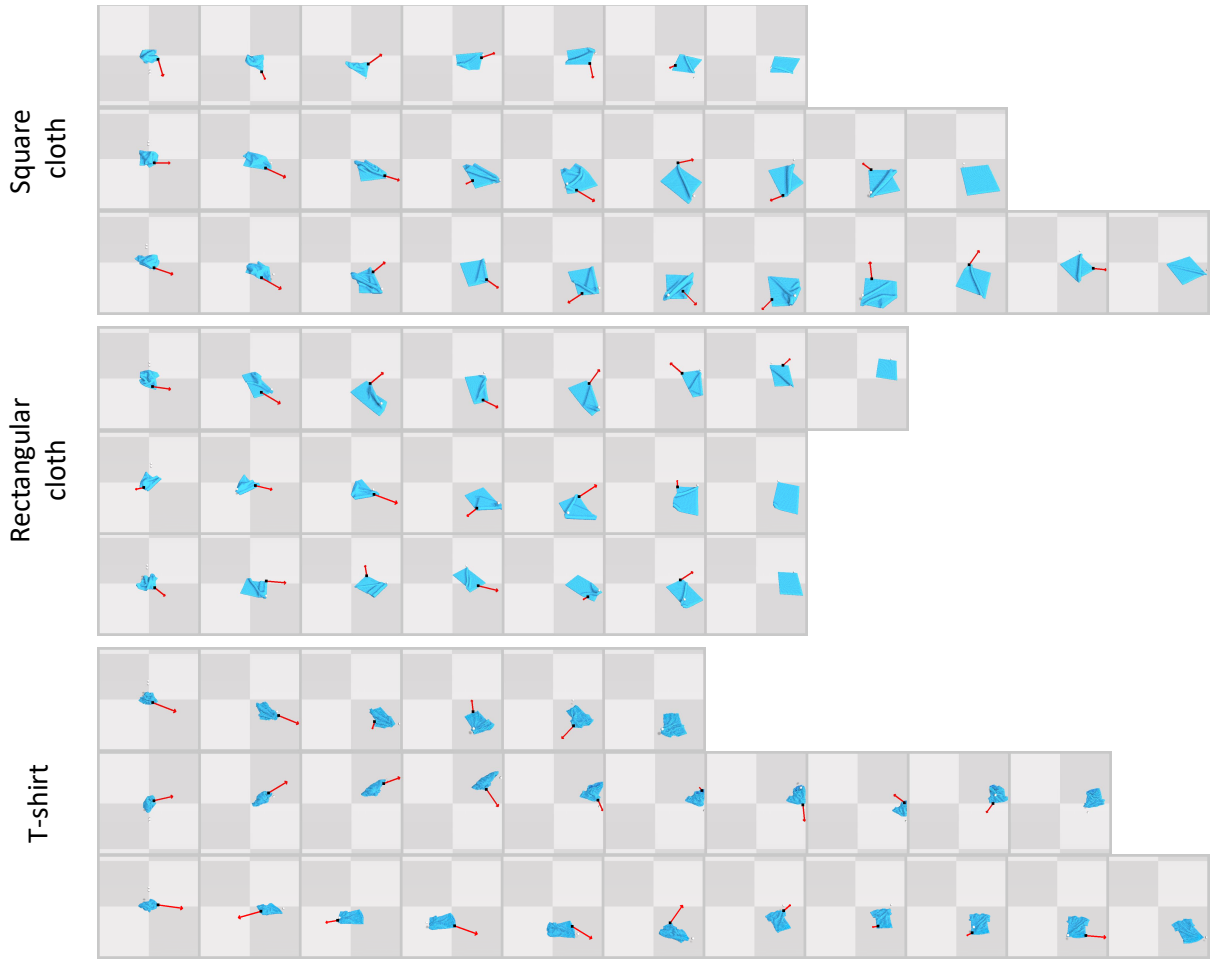


Figure B.5: Three example planned pick-and-place action sequences for square cloth, rectangular cloth, and t-shirt. All trajectories shown achieve a normalized improvement above 0.98.

B.5 Additional Experiments

B.5.1 Planning with VCD for Cloth Folding

We show that VCD can also be used for cloth folding. We assume an initially flattened cloth is given, which can be obtained via planning with VCD for smoothing. Given a goal configuration of a target folded cloth (e.g., a diagonal fold for square cloth), we use VCD with CEM to plan actions that fold the cloth into the target configuration. We explore the following three different goal specifications and cost functions for the CEM planning:

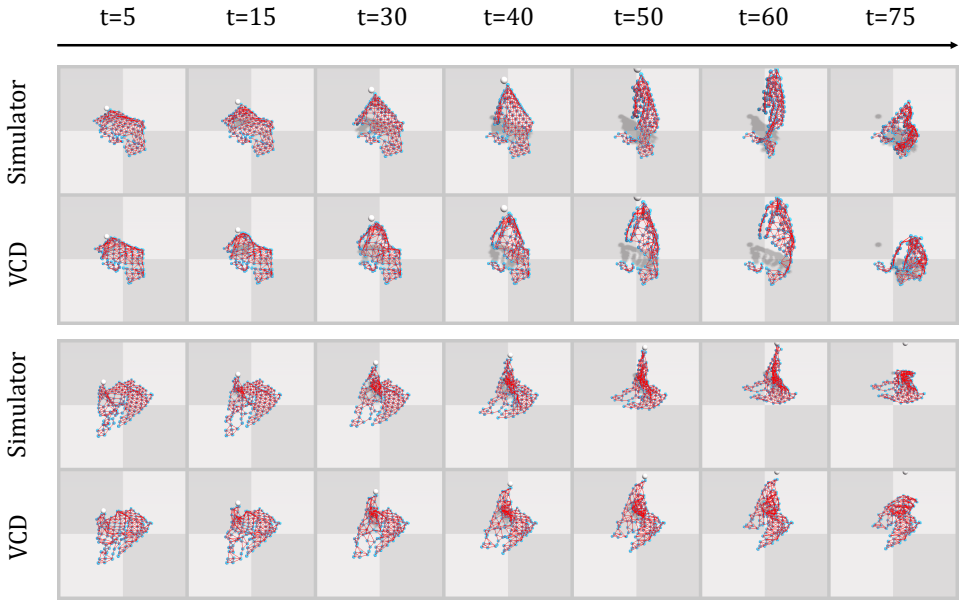


Figure B.6: Two open-loop predictions of VCD on square cloth. Blue points are observable particles/point cloud points and red lines are mesh edges. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by VCD, in which the mesh edges are inferred by the edge prediction GNN.

- A ground-truth cost function and goal specification that assumes access to the simulator cloth particles. The goal configuration of the cloth is specified as the goal locations of all particles. Given the voxelized point cloud of the initially flattened cloth, we first find a nearest neighbor mapping from each point in the point cloud to the simulator particles. The cost is then computed as the distance between the points in the achieved point cloud and their corresponding nearest-neighbor particles in the goal configuration.
- We use the point cloud of the cloth for goal specification and Chamfer distance as the cost. Specifically, the cost is the Chamfer distance between the achieved point cloud and the goal point cloud.
- We use the depth image of the cloth for goal specification and 2D IOU as the cost. Specifically, we compute the intersection over union between the segmented achieved depth map and the segmented goal depth map as the cost.

We evaluate VCD on three goals as shown in Figure B.10, B.11, B.12: (1) one-corner-in, which folds one corner of the square cloth towards the center; (2) diagonal, which folds

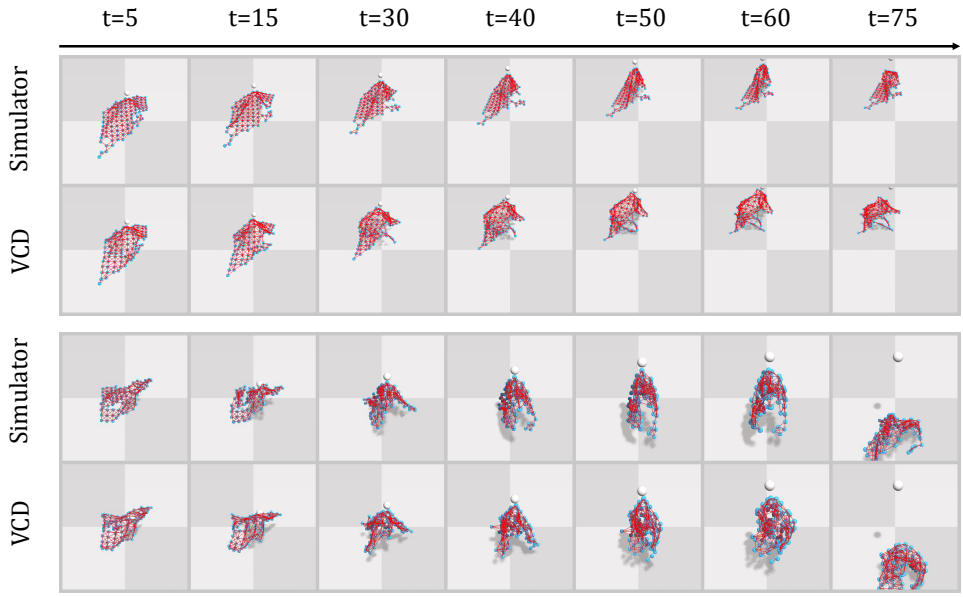


Figure B.7: Two open-loop predictions of VCD on rectangular cloth. Note VCD is only trained on square cloth. Blue points are particles/point cloud points and red lines are mesh edges. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by VCD, in which the mesh edges are inferred by the edge prediction GNN.

one corner of the square cloth towards the diagonal corner; (3) arbitrary, which folds one corner of the square cloth towards the middle point of the opposite edge. For evaluation, we report the average particle distance between the achieved cloth state and the goal cloth state. The numerical results are shown in Table B.5 and the qualitative results are shown in Figure B.10, B.11, B.12.

As the result shows, VCD can be applied for folding with the above three ways for goal specification. For the ground-truth goal specification and cost computation using simulator particles, VCD performs fairly well for folding (average particle error within 0.3 - 1.3 cm, also see Figure B.10, B.11, B.12 for qualitative results). With goal specification via point cloud and Chamfer distance as the cost, the performance of VCD is also reasonable (average particle error 0.3 - 2 cm, also see below for qualitative results), making it a practical choice to apply VCD for folding in the real world.

We also note that this VCD model is trained with random pick-and-place actions; the folding performance could be further improved if we add bias (such as corner grasping)

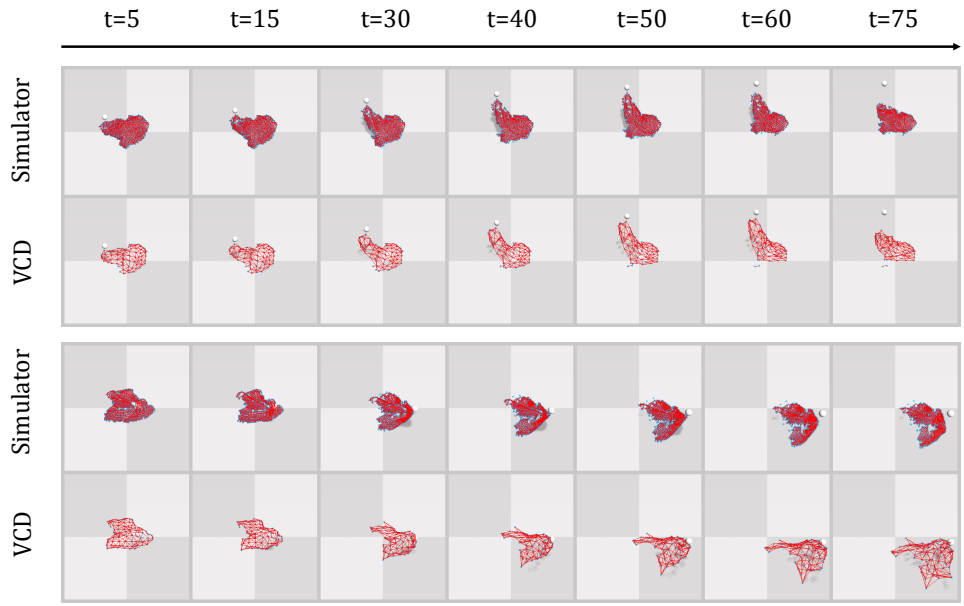


Figure B.8: Two open-loop predictions of VCD on t-shirt. Blue points are particles/point cloud points and red lines are mesh edges. Note VCD is only trained on square cloth. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by VCD, in which the mesh edges are inferred by the edge prediction GNN.

during data collection to train VCD with more folding motions.

	One-corner-in	Diagonal	Arbitrary
Ground-truth mapping	3.480	13.466	4.136
Chamfer distance	3.311	19.398	18.132
IOU	36.897	15.744	19.871

Table B.5: Average particle distance (mm) between final achieved cloth state and goal cloth state.

B.5.2 Robustness to Depth Sensor Noise

When deployed in the real world, VCD might suffer from the depth camera noise. To investigate this, we manually add different levels of noise (Gaussian noise with different levels of variance) to the depth map in the simulation and test VCD's planning performance (with a maximal number of 10 pick-and-place actions). The result is shown in Figure B.13.

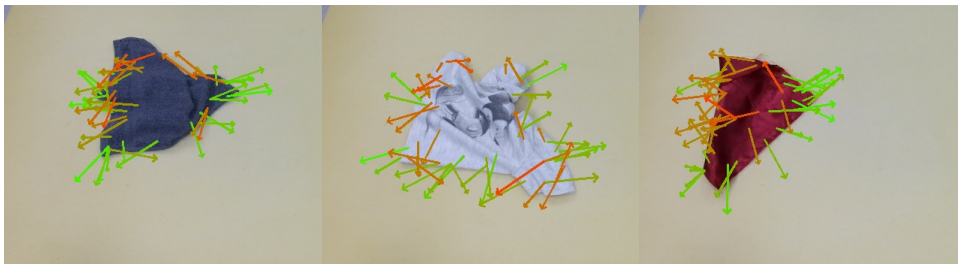


Figure B.9: Examples of 50 sampled actions used for planning. Each arrow goes from the 2D projection of the pick location to that of the place location. The actions with the higher predicted reward are shown in greener color and the actions with the lower predicted reward are shown in redder colors.

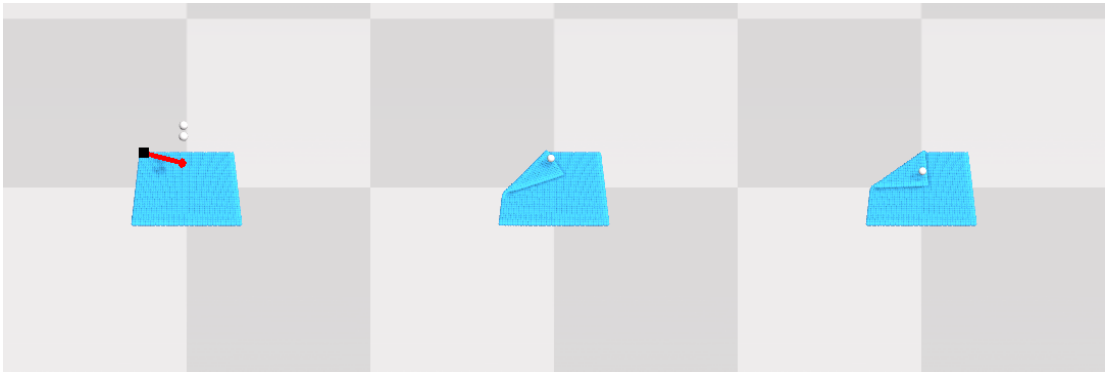
The dashed vertical line is the noise level of Azure Kinect depth camera that we use in the real world, as measured by Michal et al. Tölgyessy et al. (2021). As shown, VCD is quite robust within the noise range of the Azure Kinect depth sensor.

B.5.3 Comparison to Oracle using the FleX Cloth Model

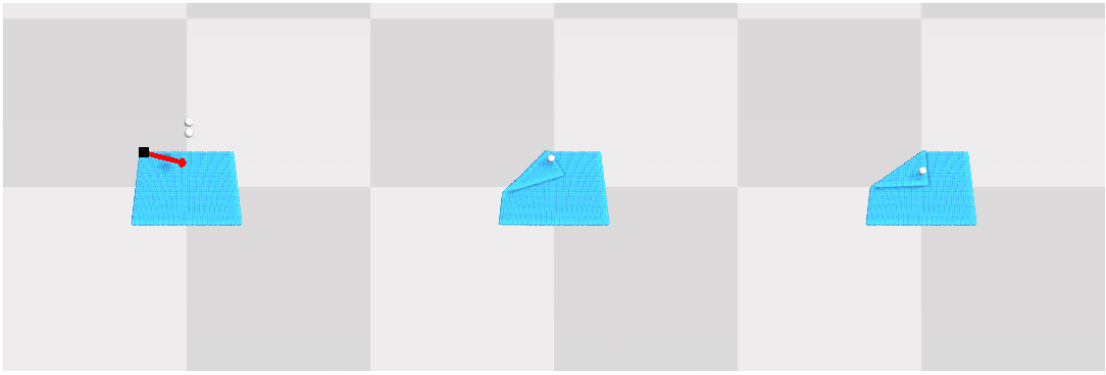
How good can the system be if we know the full cloth dynamics? To answer this question, for our simulation experiments (shown in Figure B.14), we additionally show the performance of an oracle that uses the FleX cloth model for planning in Figure B.14. Here, oracle uses the same planning method as VCD and achieves perfect results in different clothes. This shows that better performance can be achieved if the full cloth model and dynamics can be better estimated, which we leave for future work.

B.5.4 Ablations on architectural choices

For our edge and dynamics GNNs, we adopt the model architecture from GNS Sanchez-Gonzalez et al. (2020), as described in Appendix B.1.1. In Sanchez-Gonzalez, et al Sanchez-Gonzalez et al. (2020), a comprehensive analysis on architectural design decisions for the GNS model was investigated. We modify the GNS architecture by adding a global model in each GN block of the processor, which has the potential to speed up the propagation of information across the graph. The global model has been widely used in previous works in graph neural networks Battaglia et al. (2018); Wang et al. (2020); Hamrick et al. (2018). Figure B.15 (left) shows that using a global model in the dynamics model yield better



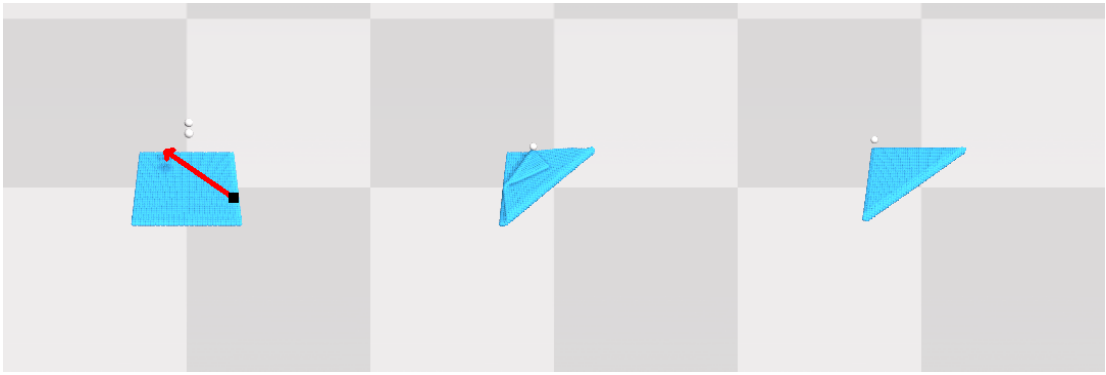
(a) Cost: groundtruth mapping



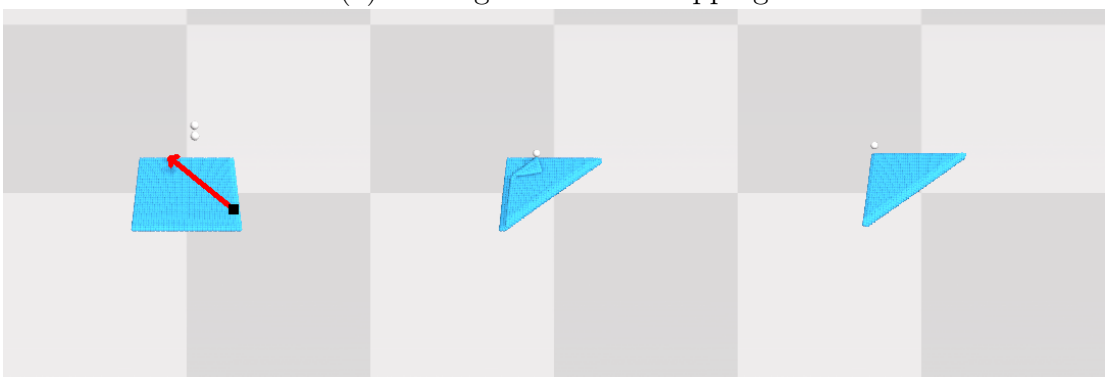
(b) Cost: Chamfer distance

(c) Cost: IOU

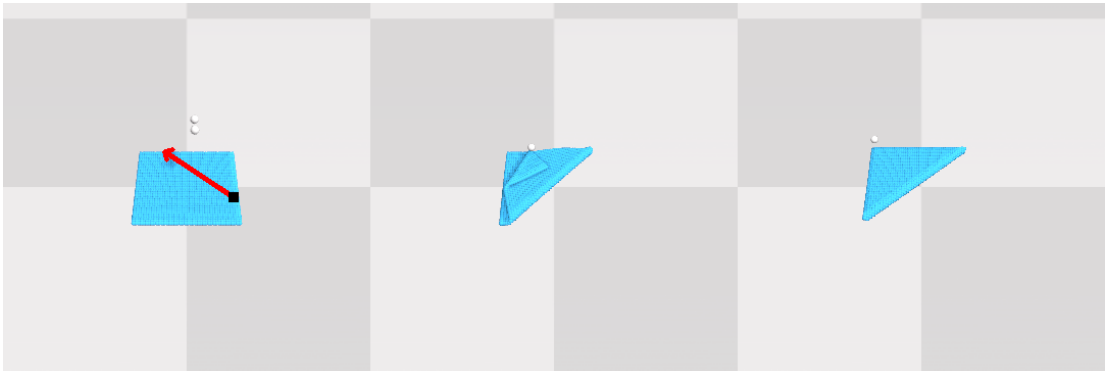
Figure B.10: VCD for folding, one-corner-in goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.



(a) Cost: groundtruth mapping

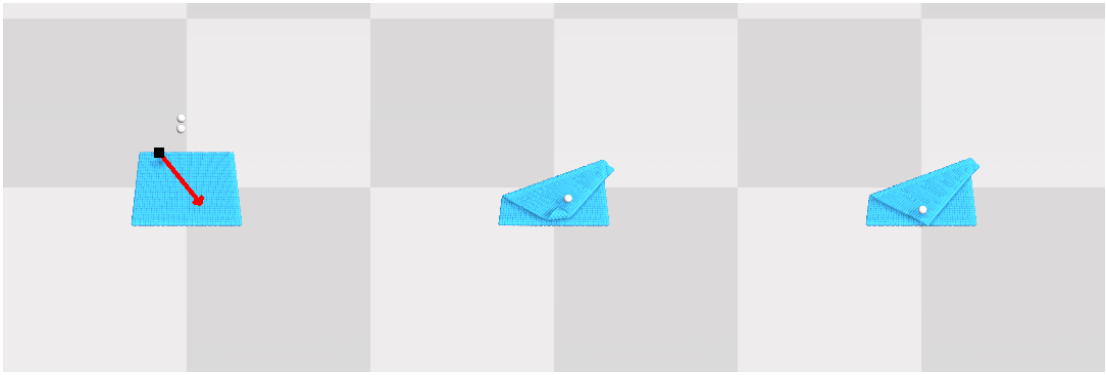


(b) Cost: Chamfer distance

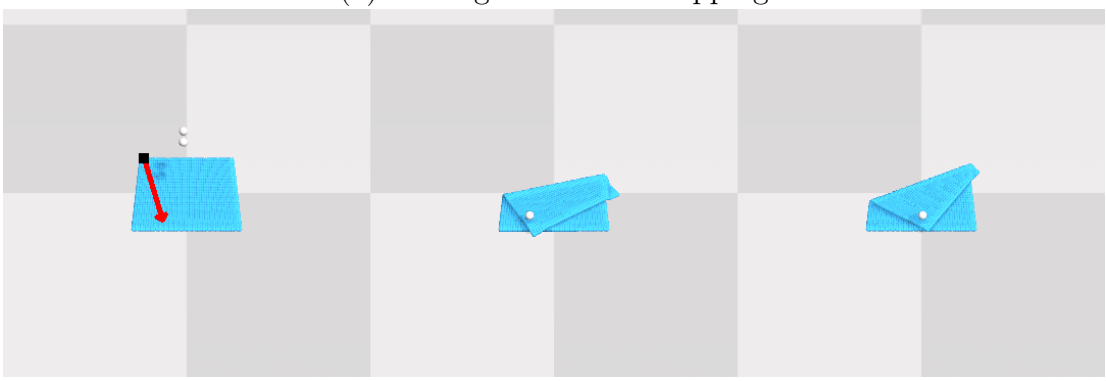


(c) Cost: IOU

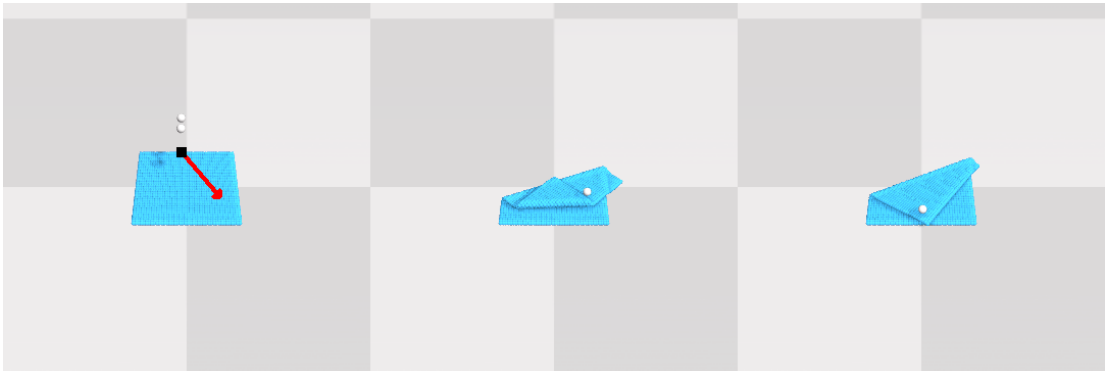
Figure B.11: VCD for folding, diagonal goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.



(a) Cost: groundtruth mapping



(b) Cost: Chamfer distance



(c) Cost: IOU

Figure B.12: VCD for folding, arbitrary goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.

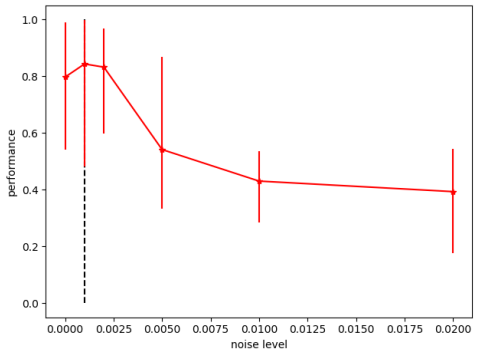


Figure B.13: Normalized Improvement of VCD under different levels of depth sensor noise, with a maximal number of 10 pick-and-place actions for smoothing. The vertical dashed line represents the typical level of Azure Kinect noise, which is the depth sensor that we use for the real-world experiment. The error bars show the 25% and 75% percentile.

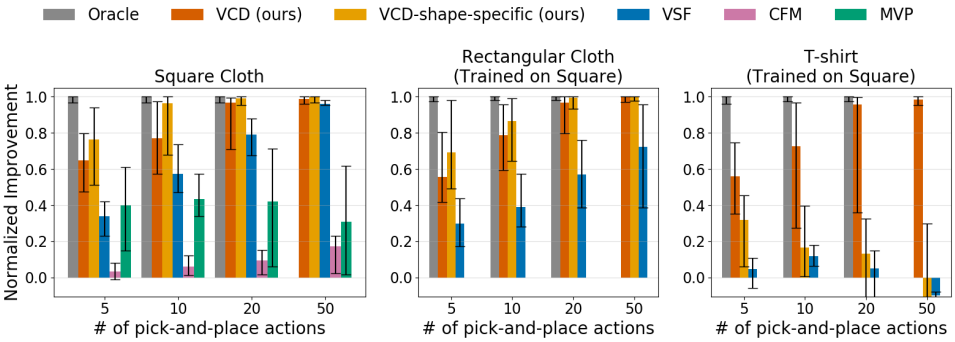


Figure B.14: Normalized improvement on square cloth (left), rectangular cloth (middle), and t-shirt (right) for varying number of pick-and-place actions. The height of the bars show the median while the error bars show the 25 and 75 percentile.

planning performance than without it.

We also evaluate the sensitivity of our dynamics model to the number of message passing steps (L). As shown in the right figure of Figure B.15, our dynamics model is robust to a broad range of values for the number of message passing steps. We speculate that, when the number of message passing is too small, the effect of action cannot propagate to the particles that are distant from the picked point. With too many message passing steps, the model is prone to overfitting. Nonetheless, Figure B.15 (right) shows that there is a broad of values for the number of message passing steps that lead to similar performance; thus, our model is fairly robust to this parameter.

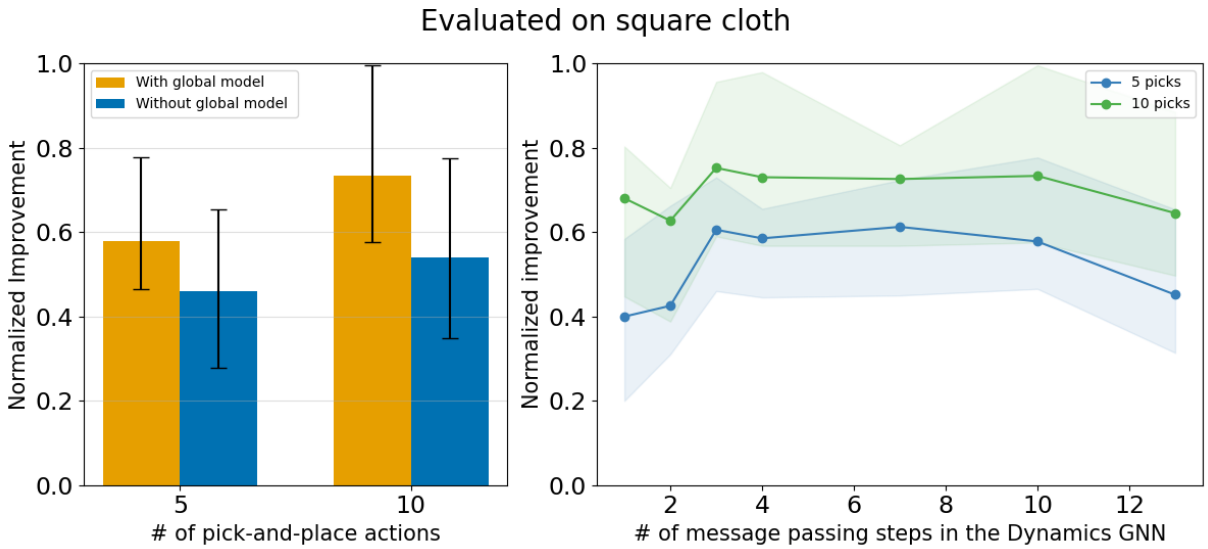


Figure B.15: We evaluate the effects of a global model and the number of message passing steps in the dynamics GNN on the square cloth. The left figure shows that the usage of a global model is helpful to the planning performance. The right figure shows that our model is generally robust to the number of message passing steps as long as the number lies within the range of [3, 10].

Appendix C

Hyper-parameteres for DiffSkill

A list of the hyperparameters used can be found in Table C.1.

Model parameter	Value
dimension of latent space	16
MLP hidden node number	1024
Training parameters	**Value**
learning rate	0.001
batch size	128
optimizer	Adam
beta1	0.9
beta2	0.999
weight decay	0
hindsight relabeling ratio $p_h ind$	0.5
Feasibility predictor loss weighting λ_{fea}	1
Reward predictor loss weighting λ_r	1
VAE loss weighting λ_{vae}	1
Trajectory optimizer	
learning rate	0.02
number of iteration	1
episode length T	50
action sequence initialization	all initialized to zero
Planning over continuous variables using Adam	
number of iteration	1500
learning rate	0.5
number of initial seeds	400
log likelihood constraint M	-10 * H

Table C.1: Summary of all hyper-parameters.

Appendix D

Details on Planning with Spatial-Temporal Abstraction

Contents

D.1 Implementation Details

D.1.1 Entity Encoding

To train our point cloud variational autoencoder Yang et al. (2019), we normalize the point cloud of each entity P_i to be centered at the origin, i.e. $\bar{P}_i = P_i - t_i$, where t_i is the mean of all points in P_i. We then encode each centered $\bar{P}_i$ into a latent encoding: $z_i = \phi(\bar{P}_i)$. Our latent representation $u_i = [z_i, t_i]$ consists of the encoding of the point cloud's shape z_i and the position of the center of the point cloud t_i. We model the point cloud's position t_i explicitly such that the learned latent embedding z_i can focus on shape variation alone and the model that plans over u_i can still reason over point clouds at different spatial locations. During training, we record the 3D bounding box of all training data $t_{min}, t_{max} \in \mathbb{R}^3$, and we sample from $[t_{min}, t_{max}]$ during planning. We denote this combined distribution of $u = [z, t]$ as P_u.

D.1.2 Details on Training Set Feasibility Predictor

Hard Negative Samples Suppose skill k takes N_k latent vectors from observation $\hat{U}^o$ and M_k latent vectors from goal $\hat{U}^g$ as input. To generate random pairs of observations and goals as negative samples for the feasibility predictor, we can sample each latent point cloud representation u_i by sampling the shape z_i from the VAE prior p_z and sampling the position t_i from the distribution of positions in the training dataset. Such random negative samples are used similarly in DiffSkill Lin et al. (2022a). However, as the combined dimension of the set representation becomes larger compared to a flat representation, we need a way to generate harder negative samples. To do so, for a positive pair of set representation $(\{u_i^o\}, \{u_j^g\})$, we randomly replace one of the entities u_i^o or u_j^g with a random sample in the latent space and use it as a negative sample. Our ablation results show that this way of

generating hard negative samples is crucial for training our set feasibility predictor.

Noise on Latent Vectors During training of the feasibility predictor, for each of the input latent vector $u = [z, t]$, where $z \in \mathbb{R}^{D_z}$ is the latent encoding of the shape and $t \in R^3$ is the 3D position of the point cloud, we add a Gaussian noise to each part, i.e. $\hat{z} = z + \sigma_z \epsilon$ and $\hat{t} = t + \sigma_t \epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The amount of noise determines the smoothness of the feasibility landscape. Without any noise, planning with gradient descent with the feasibility function becomes much harder.

D.1.3 Details on the Attention Structure for Planning with Set Representation

Given the initial latent set observation U^{obs} with N_o components, and the skill sequences $k_1, \ldots, k_H$, in this section we describe how to generate the attention structure. We denote the latent set representation at step h as U^h, $h = 1 \ldots H$ and define $U^0 = U^{obs}$. As skill k_h takes in N_{k_h} components as input and M_{k_h} components as output, by calculation we know that U^h has $N_o + \sum_{i=1}^{h} M_{k_i} - N_{k_i}$ components. From now on, we denote $|U^h| = N_h$ and $U^h = \{u_{h,1}, \ldots, u_{h,N_h}\}$. As the skill k_h only applies to a subset of the input U^{h-1}, we now formally define the attention structure at step h to be I^h, which consists of a list of indices, each of length N_{k_h}, such that I^h selects a subset from U^{h-1} to be the input to the feasibility predictor, i.e.

$$\hat{U}^{h-1} = U^{h-1}_{I^h} \subseteq U^{h-1}.$$

However, enumerating all I^h is infeasible, as there are $C_{N_h}^{N_{k_h}}$ combinations for each step. Fortunately, we do not have to enumerate all different structures. The insight here is that, for each attention structure $I_1, \ldots I_H$, we will perform a low-level optimization. In this low-level optimization, we will first initialize all the latent vectors to be optimized from P_u and then perform gradient descent on them. As many of the attention structures yield topologically equivalent tree structures (An example of such tree is illustrated in Figure 2c of the main paper), and each latent vector in the tree is sampled independently from the same distribution P_u, these topologically equivalent tree structures result in the same optimization process. As such, we do not need to exhaust all of such attention structures.

Instead of enumerating each topologically different structure and then sample multiple initializations for the low-level optimization, we randomly sample sequences $(I_1, \ldots I_H)$

and perform low-level gradient-descent optimization on all the samples. In this way, with enough samples, we will be able to cover all attention structures.

Now, we can sequentially build up the subgoals latent set representation during planning. Specifically, assuming that we have constructed the previous latent set representation U^{h-1}, we will now describe the procedure for constructing U^h, as well as the predicted feasibility for the current skill k_h, i.e. $f_{k_h}(\hat{U}^{o,h}, \hat{U}^{g,h})$, where $\hat{U}^{o,h}, \hat{U}^{g,h}$ are the subset of U^{h-1} and U^h attended by the feasibility predictor. First, we generate I_h by randomly choose the index of a subset from U_{h-1}. U^h are composed of two parts: The first part are the latent vectors generated by applying the skill k_h. For this part, we will create a set of new vectors $u_{h,0}, \dots u_{h,M_{h_k}}$. This part of the latent vectors will be attended by the feasibility predictor as $\hat{U}^{g,h} = \{u_{h,0}, \dots u_{h,M_{h_k}}\}$. The second part of U^h comes from the previous latent set vectors that are not modified by the skill, i.e. $U^{h-1} \setminus U^{h-1}_{I_h}$, and U^h is the addition of both parts, i.e.

$$U^h = \hat{U}^{g,h} \cup (U^{h-1} \setminus U^{h-1}_{I_h})$$

In this way, we can sequentially build up U^h from U^{h-1}, and U^0 is simply U^{obs}. At the same time, we have determined our attention structure and the feasibility prediction. Our objective can thus be written as

$$\arg\min_{\mathbf{k},\mathbf{U}} C(\mathbf{k}, \mathbf{U}) = \prod_{h=1}^{H} f_{k_h}(\hat{U}^{o,h}, \hat{U}^{g,h}) \exp(-R(U^H, U^g)), \tag{D.1}$$

where $\mathbf{U}$ is the set union of all latent vectors to be optimized.

D.1.4 Network Architectures

Set Feasibility Predictor We use a Multi-Layer Perceptron (MLP) with ReLU activations for our feasibility predictor. We apply max-pooling the transformed latent vectors of $\hat{U}^o$ and $\hat{U}^g$ to achieve permutation invariance. Below is our architecture:

Set Reward Predictor We use a 3-layer MLP with a hidden dimension of 1024 and ReLU activations.

Set Policy The set point cloud policy for the kth skill π_k takes in an observed point cloud P^{obs}, a goal point cloud P^{goal}, and a tool point cloud P_k^{tool} and outputs an action at each timestep to control the tool directly. The tool point cloud P^{tool} is obtained by sampling points on the mesh surface of the tool and then transforming these points to the

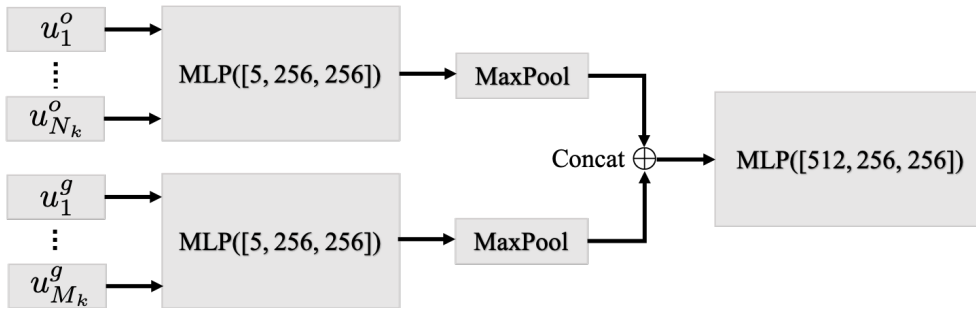


Figure D.1: Architecture for the set feasibility predictor

same camera frame as the P^{obs} and P^{goal} , assuming the pose of the tool is known from the robot state. Instead of taking latent vectors as input, the policy functions directly in point cloud space, which allows it to handle times when spatial abstraction is ambiguous. For instance, during cutting and merging, the number of dough components gradually increase or decrease, during which the latent set representation is not changing smoothly while the point clouds change smoothly during the process. We later show the advantage of using point clouds directly as the policy input. We concatenate each point's (x, y, z) coordinates with a one-hot encoding to indicate whether the point belongs to the observation, tool, or goal, and we input the points into a PointNet++ Qi et al. (2017) encoder followed by an MLP which outputs the action. We use a point cloud for the tool to allow the PointNet encoder to reason about the interaction between the tool and the dough in the same space. We use PyTorch Geometric's Fey and Lenssen (2019) implementation of PointNet++ and with the following list of modules in our encoder.

```
SAModule(0.5,0.05,MLP([3+3,64,64,128]))
SAModule(0.25,0.1,MLP([128+3,128,128,256]))
GlobalSAModule(MLP([256+3,256,128,512,1024]))
```

The MLP following the encoder consists of hidden dimensions $[1024, 512, 256]$ and ReLU activations.

Parameter	LiftSpread	GatherMove	CutRearrange	CRS + CRS-Twice
Yield stress	200	200	150	150
Ground friction	1.5	1.5	0.5	0.5
Young's modulus (E)	5e3	5e3	5e3	5e3
Poisson's ratio (ν)	0.15	0.15	0.15	0.15

Table D.1: Parameters for simulation dough

D.2 Details on Simulation Experiments

D.2.1 Hyperparameters for Simulation Dough

We use PlasticineLab Huang et al. (2020) for evaluating our simulation experiments. We provide the hyperparameters that are relevant to the properties of the dough in simulation to enhance the replicability of our results. See Table D.1 for details.

D.2.2 Hyperparameters for DBSCAN

To cluster a point cloud, we use Scikit-learn's Pedregosa et al. (2011) implementation of DBSCAN Ester et al. (1996) with `eps=0.03, min_samples=6, min_points=10` for all of our environments. Further, we assign each noise point identified by DBSCAN to its closest cluster.

D.2.3 Hyperparameters for PASTA

Table D.2 shows the hyperparameters used for PASTA in our simulation tasks. Planning for CRS-Twice requires a large amount of samples. Therefore, we modify the planner to improve sample efficiency. See Sec. D.2.4 for details.

D.2.4 Receding Horizon Planning for CRS-Twice

As the planning horizon increases, the number of possible skill sequences as well as the number of possible attention structures increase exponentially. The task of CRS-Twice has a planning horizon of 6 and is a much more difficult task to solve. As such, for this task we specify the skill sequences and use Receding Horizon Planning (RHP). Starting from the

Training parameters	LiftSpread	GatherMove	CutRearrange	CRS	CRS-Twice
Point Cloud VAE					
learning rate	2e-3	2e-3	2e-3	2e-3	2e-3
latent dimension	2	2	2	2	2
Feasibility					
learning rate	1e-4	1e-3	1e-4	1e-4	1e-4
batch size	256	256	256	256	256
noise on shape encoding σ_z	0	0	0	0.02	0.02
noise on position σ_t	0.01	0.01	0.005	0.01	0.01
Reward					
learning rate	1e-4	1e-3	1e-4	1e-4	1e-4
batch size	256	256	256	256	256
Policy					
learning rate	1e-4	1e-3	1e-4	1e-4	1e-4
batch size	10	10	10	10	10
noise on point cloud	0.005	0.005	0.005	0.005	0.005
Planning parameters	**LiftSpread**	**GatherMove**	**CutRearrange**	**CRS**	**CRS-Twice**
learning rate	0.01	0.01	0.01	0.01	0.01
number of iterations	200	100	100	200	300
number of samples	5000	5000	5000	50000	500000

Table D.2: Summary of hyperparameters used in PASTA. For CRS-Twice, we use the same model as CRS but modify the planner to have better sample efficiency.

first time step, we follow the procedure in Algorithm 2 but only optimize for H_{RHP} steps into the future and compare the achieved subgoal with the final target to compute the planning loss. After optimization, we take the first subgoal from the plan and discard the rest of the plan. We then repeat this process until we reach the overall planning horizon H. In our experiments, we use $H_{RHP} = 3$. While we can perform model predictive control and execute the first step before planning for the second step, we find this open-loop planning and execution to be sufficient for the task.

D.3 Details on Real World Experiments

D.3.1 Heuristic Policies

Transferring the learned policies from simulation to the real world can be more difficult than transferring the planner itself, as the policies are affected more by the sim2real gap, such as the difference in friction and properties of dough in the real world. To sidestep this challenge, for our real world experiments, we design three heuristic policies: cut, push, and roll to execute the generated plans in the real world.

Just like our learned policies in simulation, each heuristic policy takes in the current observation and the generated subgoal in point clouds and outputs a sequence of desired end effector positions used for impedance control. In addition, each policy takes in the attention mask provided by the planner indicating the components of interest. The same DBSCAN procedure is used for this. For the cut policy, it first calculates the cutting point by computing the length ratio of the resulting components. Then it cuts the dough and separates it such that the center of mass of each resulting component matches the one in the subgoal. For the push policy, given a component and a goal component, the policy pushes the dough in the direction that connects two components' center of mass. For the roll policy, it first moves the roller down to make contact with the dough. Then, based on the goal component's length, the policy calculates the distance it needs to move the the roller back and forth when making contact with the dough.

D.3.2 Procedure for Resetting the Dough

To compare different methods with the same initial and target configurations, we first use a 3D-printed mold to fit the dough to a same initial shape. We then overlay the desired initial location on the image captured the top-down camera and place the dough at the corresponding location in the workspace to ensure different methods start from the same initial location.

D.3.3 Procedure for the Human Baseline

Following the same procedure in section D.3.2, we first reset the dough to the initial configuration. Then, we overlay the goal point cloud on the image captured the top-down

camera. The overlay image is shown on a screen and presented to the human in real time when the human is completing the task.

D.3.4 Procedure for Making the Real Dough

Material	Quantity(g)	Baker's percentage(%)
Flour	300	-
Water	180	60
Yeast	3	1

Table D.3: All purpose dough recipe

We follow the recipe shown in Table D.3 to make the real dough. First, we take 300 grams of flour, 3 grams of yeast and 180 grams of water into a basin. Then, we mix the ingredients and knead the dough for few minutes. Next, we use a food warp to seal the dough in the basin and put them in the refrigerator to let the dough rest for 4-5 hours. Finally, we take out the dough from the refrigerator and reheat it with a microwave for 30-60 seconds to soften it.

D.4 Additional Experiments

D.4.1 Ablation Studies

Ablations on feasibility predictor. Following the discussions in Sec D.1.2, we train a feasibility predictor without adding any noise to show that adding noise helps with the optimization landscape during planning. We call this ablation *No Smoothing Feasibility*. As shown in Table D.4, this variant only achieves half of the success rate of PASTA, suggesting the importance of noise during training.

Ablation Method	Performance / Success
No Smoothing Feasibility	0.744 / 40%
Shared Encoder Policy	-0.304 / 0%
Tool Concat Policy	0.516 / 60%
Set without Filtering	0.360 / 20%
PASTA (Ours)	**0.837 / 80%**

Table D.4: Additional ablation results from CutRearrange.

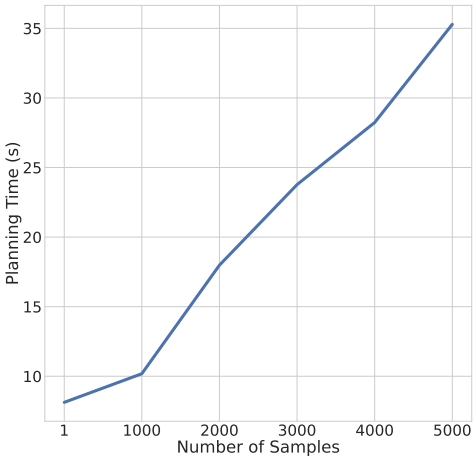
Ablations on policy. We consider two ablation methods for our set policy. First, we consider a *Shared Encoder Policy* that takes in the latent vectors from the encoder and uses a max pooling layer followed by an MLP to produce the action. The architecture is very similar to our Set Feasibility Predictor. Our results in Table D.4 shows that this architecture has zero success in our task. We hypothesize that this is because the entity encoding can be unstable during the skill execution. For example, during cutting, the dough slowly transitions from one piece to two pieces, making the input to the policy unstable.

Second, we compare with a *Tool Concat Policy* that takes in the observation and goal point cloud of the dough, passes them through a PointNet++ Qi et al. (2017) encoder to produce a feature, and then concatenates the tool state to the feature. The concatenated feature is passed through a final MLP to output the action. In comparison, the set policy in PASTA takes the point cloud of the tool and concatenate it with the dough in the point cloud space before passing it to the PointNet. We hypothesize that this way allows PointNet to reasons more easily about the spatial relationships between the tool point cloud and the dough point cloud. Results in Table D.4 highlight the advantage of using a point cloud to represent the tool.

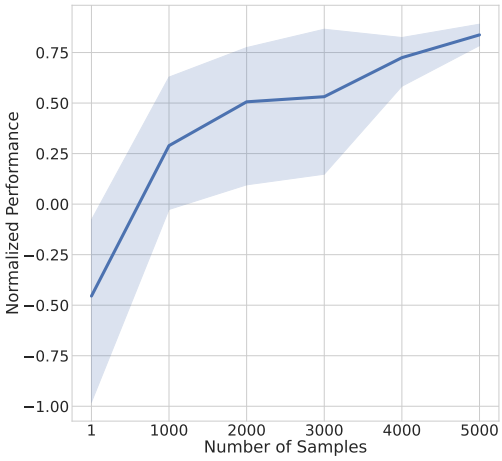
Ablation on set representation. We consider a variant of PASTA *Set without Filtering*, which uses the same set representation as PASTA, but does not filter entities that are approximately the same both during training and testing. This filtering is only possible with a set representation and we want to show the advantage of this filtering. For this ablation, during training, the feasibility predictor takes in all the entities in the scene in set representation, and the policy takes in the concatenation of point clouds from each entity. During planning, we do not enumerate attention structures but instead optimize for all the entities. As shown in Table D.4, without filtering, this ablation performs significantly worse than PASTA, showing that filtering is an important advantage enabled by our set representation.

D.4.2 Visualization of the Latent Space

We visualize the latent space of PASTA in CutRearrange in Figure D.3 and visualize the latent space of Flat 3D baseline in Figure D.4 for comparison. Since we use a latent dimension of 2 for all of our environments, we can visualize the original latent space without



(a) Planning time v.s. number of samples



(b) Planning performance v.s. number of samples

Figure D.2: Planning time and performance with varying number of samples in CutRearrange.We show the mean and standard deviation of performance over 5 runs.

applying any dimensionality reduction techniques. PASTA only encodes the shape of each entity and thus can better model the variations in shapes. On the other hand, Flat 3D couples the shape variation with the relative position of two entities. This makes a flat representation difficult to generalize compositionally to scenes with different numbers of entities or scenes with entities that have novel relative spatial locations to each other.

D.4.3 Runtime of PASTA

We implement the planning in the latent set representation in an efficient way, which can plan with multiple different structure in parallel on a GPU. To demonstrate the efficiency of PASTA, we vary the number of samples used for planning and record the planning time and final performance. We conduct the experiments in CutRearrange. Figure D.2a shows that the planning time scales approximately linearly with the number of samples, and Figure D.2b shows the planning performance versus the number of samples. As the result suggests, PASTA can achieve its optimal performance with a very short amount of planning time ($35s$). Finally, we summarize the planning time for all of our tasks in simulation in Table D.5.

	LiftSpread	GatherMove	CutRearrange	CRS	CRS-Twice*
Planning time (seconds)	58	35	35	307	7810

Table D.5: Summary of planning time of PASTA in all of the simulation tasks. CRS-Twice uses Receding Horizon Planning, which results in an increase in planning time.

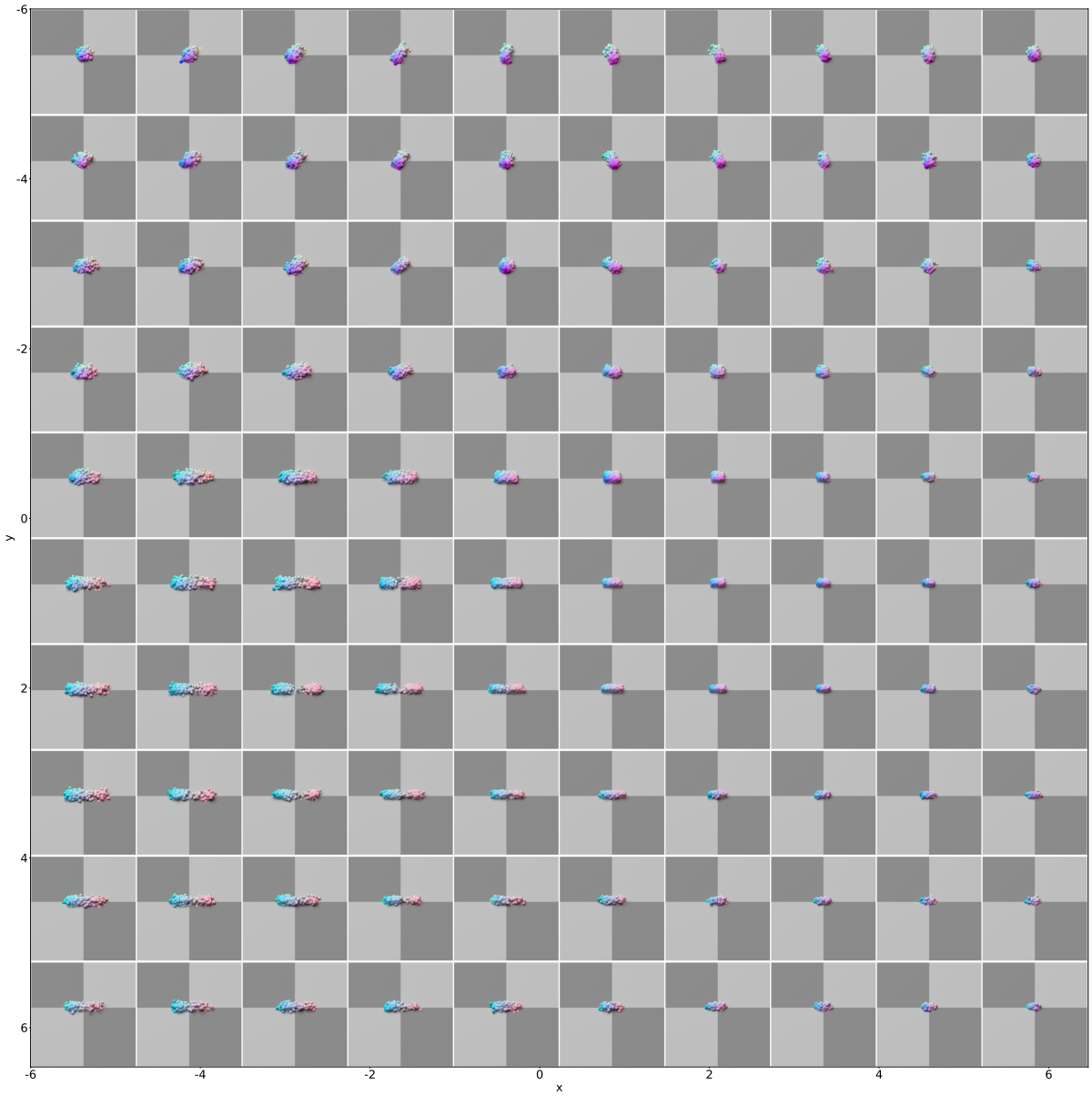


Figure D.3: Latent space of PASTA in CutRearrange. We sample coordinates on a grid from the 2D latent space encoding the shapes and then decoding each latent vector into a point cloud. We then rearrange the decoded point cloud into the grid based on the corresponding coordiantes in the latent space.

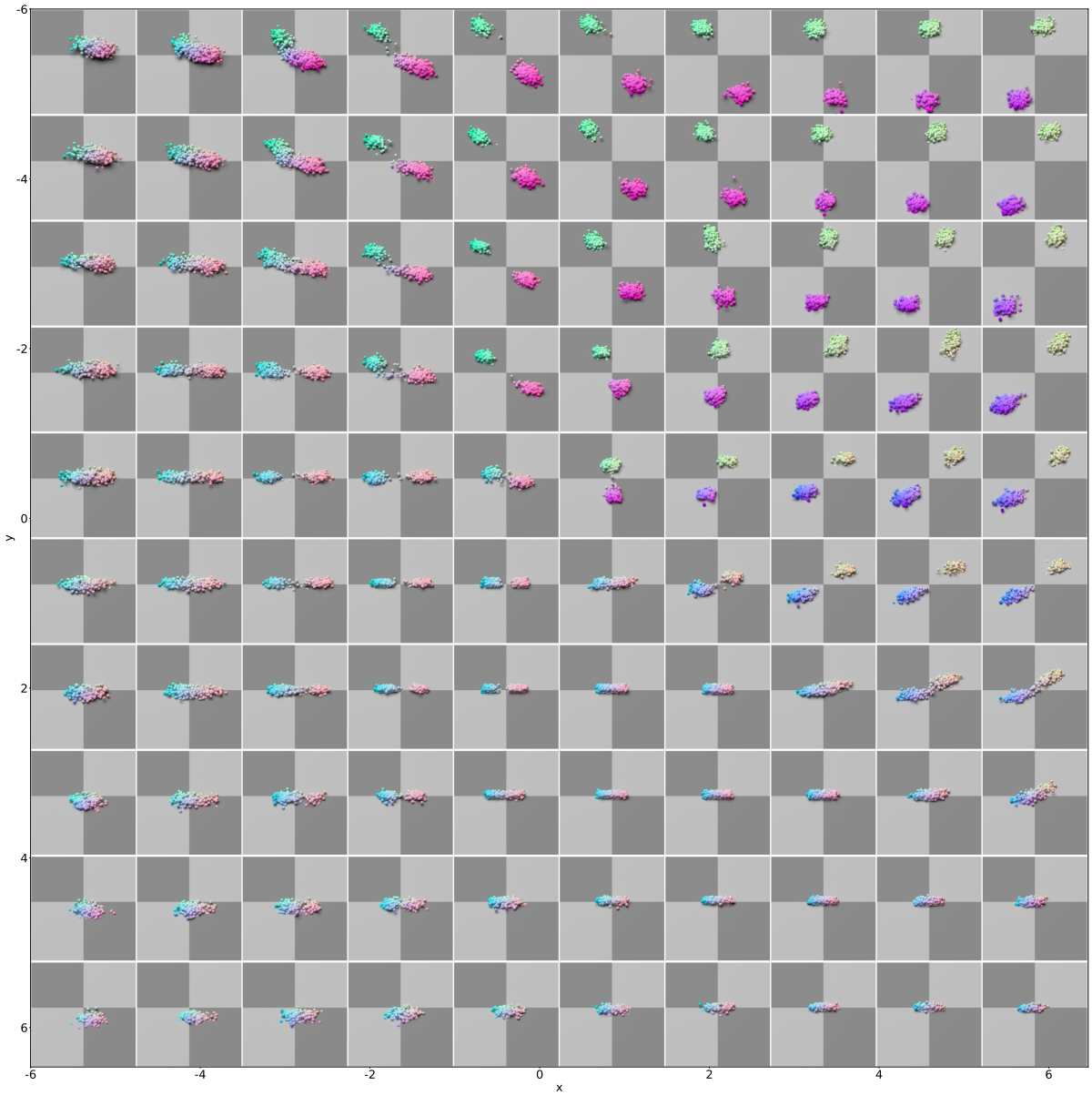


Figure D.4: Latent space of Flat 3D in CutRearrange. We sample coordinates on a grid from the 2D latent space encoding the shapes and then decoding each latent vector into a point cloud. We then rearrange the decoded point cloud into the grid based on the corresponding coordiantes in the latent space.

Bibliography

P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016. 2.1, 2.2

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017. 4.3.3

O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 2.1

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 4.2

P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016. 4.2

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 3.3.2, B.1.1, B.5.4

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013. 2.2

M. Bollini, S. Tellex, T. Thompson, N. Roy, and D. Rus. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics*, pages 481–495. Springer, 2013. 2.2, 4.1

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 2.1

M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016. 4.2

D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020. 3.3.6

T. L. Chen, M. Ciocarlie, S. Cousins, P. M. Grice, K. Hawkins, K. Hsiao, C. C. Kemp, C.-H. King, D. A. Lazewatsky, A. E. Leeper, et al. Robots for humanity: using assistive robotics to empower people with disabilities. *IEEE Robotics & Automation Magazine*, 20 (1):30–39, 2013. 2.2

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. A.2.4

E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019. 3.3.3

M. Cusumano-Towner, A. Singh, S. Miller, J. F. O'Brien, and P. Abbeel. Bringing clothing into desired configurations with limited perception. In *ICRA*, pages 3893–3900. IEEE, 2011. 3.2

M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34, 2021. 5.1

F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31:7178–7189, 2018. 4.2

J. Degrave, M. Hermans, J. Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:6, 2019. 4.2

C. Devin, P. Abbeel, T. Darrell, and S. Levine. Deep object-centric representations for generalizable robot learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7111–7118. IEEE, 2018. 5.2

T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000. 4.2

D. Driess, J.-S. Ha, and M. Toussaint. Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. *arXiv preprint arXiv:2006.05398*, 2020. 4.2

D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint. Learning multi-object dynamics with compositional neural radiance fields. In *International Symposium of Robotics Research*, 2022. 5.1, 5.2

T. Du, K. Wu, P. Ma, S. Wah, A. Spielberg, D. Rus, and W. Matusik. Diffpd: Differentiable projective dynamics. *arXiv preprint arXiv:2101.05917*, 2021. 4.2

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016. 2.1, 2.2

F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-

based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018. 2.2, 3.2, 5.2, 6.1

Z. Erickson, H. M. Clever, G. Turk, C. K. Liu, and C. C. Kemp. Deep haptic model predictive control for robot-assisted dressing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018. 2.2

N. Essahbi, B. C. Bouzgarrou, and G. Gogu. Soft material modeling for robotic manipulation. In *Applied Mechanics and Materials*, volume 162, pages 184–193. Trans Tech Publ, 2012. 2.2

M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996. 5.3.1, D.2.2

L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning (CoRL)*, pages 767–782, 2018. 2.1, 2.2

K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *The International Journal of Robotics Research*, 39(2-3):202–216, 2020a. 4.2

K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. In *Conference on Robot Learning*, pages 42–52. PMLR, 2020b. 5.2

M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. D.1.4

N. Figueroa, A. L. P. Ureche, and A. Billard. Learning complex sequential tasks from demonstration: A pizza dough rolling case study. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 611–612. Ieee, 2016. 5.2

R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971. 5.2

P. R. Florence, L. Manuelli, and R. Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2018. 5.1

C. Florensa, J. Degrave, N. Heess, J. T. Springenberg, and M. Riedmiller. Self-supervised learning of image embedding for continuous control. *Workshop at NIPS 2018*, 2019. 2.5.1

S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018. 4.4.3

C. Gan, J. Schwartz, S. Alter, M. Schrimpf, J. Traer, J. De Freitas, J. Kubilius, A. Bhandwaldar, N. Haber, M. Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. *NeurIPS Dataset and Benchmark Track*, 2021. 2.2

A. Ganapathi, P. Sundaresan, B. Thananjeyan, A. Balakrishna, D. Seita, J. Grannen, M. Hwang, R. Hoque, J. E. Gonzalez, N. Jamali, K. Yamane, S. Iba, and K. Goldberg. Learning dense visual correspondences in simulation to smooth and fold real fabrics. *arXiv preprint arXiv:2003.12698*, 2020. 4.2

A. Ganapathi, P. Sundaresan, B. Thananjeyan, A. Balakrishna, D. Seita, J. Grannen, M. Hwang, R. Hoque, J. Gonzalez, N. Jamali, K. Yamane, S. Iba, and K. Goldberg. Learning Dense Visual Correspondences in Simulation to Smooth and Fold Real Fabrics. In *ICRA*, 2021. 3.2

C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020a. 5.2

C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684. IEEE, 2020b. 4.2

C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021. 4.2

N. Gautham, Z. Kai, E. Ben, L. Xingyu, and H. David. Transparent liquid segmentation for robotic pouring. In *International Conference on Robotics and Automation (ICRA)*, 2022. 5.2

M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros. Add: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020. 4.1, 4.1, 4.2

F. Gravot, S. Cambon, and R. Alami. asymov: a planner that deals with intricate symbolic and geometric problems. In *Robotics Research. The Eleventh International Symposium*, pages 100–110. Springer, 2005. 4.2

H. Ha and S. Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. *arXiv preprint arXiv:2105.03655*, 2021. 2.6, 4.2, 6, 6.1

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. 2.5.1.2, 5.5.1, 5.1

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018. A.2.4

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019. 2.5.1.3, 2.5.2.2, 5.2

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *International Conference on Learning Representation (ICLR)*, 2020. 6.1

J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. McKee, J. B. Tenenbaum, and P. W. Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018. B.5.4

E. Heiden, M. Macklin, Y. S. Narang, D. Fox, A. Garg, and F. Ramos. DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.067. 4.1, 4.1, 4.4.1, 5.2

N. Heravi, A. Wahid, C. Lynch, P. Florence, T. Armstrong, J. Tompson, P. Sermanet, J. Bohg, and D. Dwibedi. Visuomotor control in multi-object scenes using object-aware representations. *arXiv preprint arXiv:2205.06333*, 2022. 5.2

R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg. VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation. In *Robotics: Science and Systems (RSS)*, 2020. URL https://arxiv.org/abs/2003.09044. 3.1, 3.2, 3.4.1, B.3, B.4

R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg. VisuoSpatial Foresight for Physical Sequential Fabric Manipulation. *Autonomous Robots (AURO) journal*, 2021. 5.2

Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):150, 2018. 4.4.1

Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019a. 4.2, 4.4.1

Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE, 2019b. 4.1, 4.1, 4.2

Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In *International Conference on Learning Representations*, 2020. 5.5.1, 5.1, D.2.1

Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. *International Conference on Learning Representations (ICLR)*, 2021. 4.1, 4.2, 4.3.2, 4.4.1

Z. Huang, X. Lin, and D. Held. Mesh-based dynamics model with occlusion reasoning for

cloth manipulation. *Robotics: Science and Systems (RSS)*, 2022. 3.5, 5.2

J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26): eaau5872, 2019. 2.1

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016. 3.3.6, B.1.3

S. James, Z. Ma, D. Rovick Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020. 2.2

P. Jiménez and C. Torras. Perception of cloth in assistive robotic manipulation tasks. In *Natural Computing*, pages 1–23. Springer, 2020. 3.1, 3.2

T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019. 2.1

L. P. Kaelbling and T. Lozano-Pérez. Hierarchical planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. 4.2

L. P. Kaelbling and T. Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013. 4.2

H. J. Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960. 4.3.2

F. F. Khalil and P. Payeur. Dexterous robotic manipulation of deformable objects with multi-sensory feedback-a review. In *Robot Manipulators Trends and Development*. IntechOpen, 2010. 2.2

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014. 4.3.2, A.6, B.1.2

Y. Kita and N. Kita. A model-driven method of estimating the state of clothes for manipulating it. In *Sixth IEEE Workshop on Applications of Computer Vision, 2002.(WACV 2002). Proceedings.*, pages 63–69. IEEE, 2002. 3.2

Y. Kita, F. Saito, and N. Kita. A deformable model driven visual method for handling clothes. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3889–3895. IEEE, 2004. 3.2

Y. Kita, T. Ueshiba, E. S. Neo, and N. Kita. Clothes state recognition using 3d observed data. In *2009 IEEE International Conference on Robotics and Automation*, pages 1220–1225. IEEE, 2009. 3.2

I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv*, 2020. 2.5.1.3

M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. arXiv:2004.14990, 2020a. 2.5.1.3

M. Laskin, A. Srinivas, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119*, 2020b. arXiv:2004.04136. 2.5.1.3

J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Sci Robot*, 5(47), Oct. 2020a. (document), 3.3.6, B.1

R. Lee, D. Ward, A. Cosgun, V. Dasagi, P. Corke, and J. Leitner. Learning arbitrary-goal fabric folding with one hour of real robot experience. *Conference on Robot Learning (CoRL)*, 2020b. 3.2

Y. Lee, E. S. Hu, Z. Yang, A. Yin, and J. J. Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019. URL https://clvrai.com/furniture. 2.2

Y. Lee, E. S. Hu, and J. J. Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6343–6349. IEEE, 2021. 5.4

A. Levy, G. Konidaris, R. Platt, and K. Saenko. Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*, 2017. 4.2

Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019a. 2.1, 3.1, 3.2, 4.2, 5.1, 5.2

Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211. IEEE, 2019b. 2.5.1.2, 4.2

Y. Li, S. Li, V. Sitzmann, P. Agrawal, and A. Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022. 4.2, 5.1, 5.2

J. Liang, M. Lin, and V. Koltun. Differentiable cloth simulation for inverse problems. *NeurIPS*, 2019. 4.2

J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer. Search-based task planning with learned skill effect models for lifelong robotic manipulation. In *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022. 5.2

X. Lin, H. Baweja, G. Kantor, and D. Held. Adaptive auxiliary task weighting for reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 4772–4783. Curran Associates, Inc., 2019a. URL http://papers.nips.cc/paper/

8724-adaptive-auxiliary-task-weighting-for-reinforcement-learning.pdf. 6.1

X. Lin, H. S. Baweja, and D. Held. Reinforcement learning without ground-truth state. *arXiv preprint arXiv:1905.07866*, 2019b. 2.5.1, 4.3.3, 6.1

X. Lin, P. Guo, C. Florensa, and D. Held. Adaptive variance for changing sparse-reward environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3210–3216. IEEE, 2019c. 6.1

X. Lin, Y. Wang, J. Olkin, and D. Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 2020. 3.3.3, 3.3.5, 3.4.1, 4.2, B.3.1

X. Lin, Y. Wang, Z. Huang, and D. Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, 2021. 4.2, 5.1, 5.2

X. Lin, Z. Huang, Y. Li, J. B. Tenenbaum, D. Held, and C. Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *ICLR*, 2022a. 5.1, 5.2, 5.4, 5.4.1, 5.5.1, 5.1, D.1.2

X. Lin, C. Qi, Y. Zhang, Y. Li, Z. Huang, K. Fragkiadaki, C. Gan, and D. Held. Planning with spatial-temporal abstraction from point clouds for deformable object manipulation. *preprint*, 2022b. 6

F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020. 5.3.1

X. Ma, D. Hsu, and W. S. Lee. Learning latent graph dynamics for deformable object manipulation. *IEEE International Conference on Robotics and Automation (ICRA)*, 2022. 2.6, 5.2

M. Macklin and M. Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013. 2.3

M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):1–12, 2014. 2.3, 3.3.5

J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. 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. 2.1, 2.2, 3.2, 4.1, 4.2, 5.2

L. Manuelli, W. Gao, P. Florence, and R. Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019. 5.1

I. Mariolis, G. Peleka, A. Kargakos, and S. Malassiotis. Pose and category recognition of highly deformable objects using deep learning. In *2015 International conference on advanced robotics (ICAR)*, pages 655–662. IEEE, 2015. 3.2

J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018. 2.1, 2.2, 3.1, 3.2

C. Matl and R. Bajcsy. Deformable elasto-plastic object shaping using an elastic hand and model-based reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3955–3962. IEEE, 2021. 5.2

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. *Technical Report CVC TR-98-003/DCS TR-1165*, 1998. 5.2

S. Miller, M. Fritz, T. Darrell, and P. Abbeel. Parametrized shape models for clothing. In *ICRA*, pages 4861–4868. IEEE, 2011. 2.1, 3.2

S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31 (2):249–267, 2012. 2.1

P. Mitrano, D. McConachie, and D. Berenson. Learning where to trust unreliable models in an unstructured world for deformable object manipulation. *Science Robotics*, 6(54): eabd8170, 2021. 5.2

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 2.1

D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. Yamins. Flexible neural representation for physics prediction. *arXiv preprint arXiv:1806.08047*, 2018. 4.2

M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. 2.3, 3.3.5

A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2146–2153. IEEE, 2017. 4.2

A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems*, pages 9191–9200, 2018. 2.5.1

R. Narain, A. Samii, and J. F. O'brien. Adaptive anisotropic remeshing for cloth simulation. *ACM transactions on graphics (TOG)*, 31(6):1–10, 2012. 3.3.3

S. Nasiriany, H. Liu, and Y. Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022. 5.1

F. Osawa, H. Seki, and Y. Kamiya. Unfolding of massive laundry and classification types

by dual manipulator. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 11(5):457–463, 2007. 3.2

I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepezvari, S. Singh, et al. Behaviour suite for reinforcement learning. *International Conference on Leanring Representation (ICLR)*, 2019. 2.2

D. Park, Y. Hoshi, H. P. Mahajan, H. K. Kim, Z. Erickson, W. A. Rogers, and C. C. Kemp. Active robot-assisted feeding with a general-purpose mobile manipulator: Design, evaluation, and lessons learned. *Robotics and Autonomous Systems*, 124:103344, 2020. 4.1

D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2050–2053, 2018. 2.2

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. D.2.2

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. 3.1, 4.2, 5.2

C. Qi, X. Lin, and D. Held. Learning closed-loop dough manipulation using a differentiable reset module. *Robotics and Automation Letters (RA-L) with IROS 2022 presentation*, 2022. 4.5

C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. D.1.4, D.4.1

Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin. Scalable differentiable physics for learning and control. *arXiv preprint arXiv:2007.02168*, 2020. 4.2

Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin. Efficient differentiable simulation of articulated bodies. In *ICML*, 2021. 4.2

Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese. Keto: Learning keypoint representations for tool manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7278–7285. IEEE, 2020. 4.2

S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato. An obstacle-based rapidly-exploring random tree. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 895–900. IEEE, 2006. 2.2

R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization.

Methodology and computing in applied probability, 1(2):127–190, 1999. 2.5.1.1

M. Saha and P. Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007. 2.2

J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, and Y. Mezouar. Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37(7):688–716, 2018. 2.2

A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR, 2018. 4.2

A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020. 3.1, 3.2, 3.3.2, 4.2, B.1.1, B.5.4

C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2629–2636. IEEE, 2017. 5.2

C. Schenck and D. Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pages 317–335. PMLR, 2018. 4.2

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019. 2.1

D. Seita, A. Ganapathi, R. Hoque, M. Hwang, E. Cen, A. K. Tanwani, A. Balakrishna, B. Thananjeyan, J. Ichnowski, N. Jamali, et al. Deep imitation learning of sequential fabric smoothing policies. *arXiv preprint arXiv:1910.04854*, 2019. 2.2

D. Seita, A. Ganapathi, R. Hoque, M. Hwang, E. Cen, A. K. Tanwani, A. Balakrishna, B. Thananjeyan, J. Ichnowski, N. Jamali, K. Yamane, S. Iba, J. Canny, and K. Goldberg. Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. URL https://arxiv.org/abs/1910.04854. 3.2

T. Séjourné, J. Feydy, F.-X. Vialard, A. Trouvé, and G. Peyré. Sinkhorn divergences for unbalanced optimal transport. *arXiv preprint arXiv:1910.12958*, 2019. 4.3.2, 5.5.1

P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain. Time-contrastive networks: Self-supervised learning from video. In *ICRA*, pages 1134–1141. IEEE, 2018. 2.1, 2.2

H. Shi, H. Xu, Z. Huang, Y. Li, and J. Wu. Robocraft: Learning to see, simulate, and shape elasto-plastic objects with graph networks. *Robotics: Science and Systems (RSS)*, 2022. 5.2

A. Simeonov, Y. Du, B. Kim, F. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez.

A long horizon planning framework for manipulating rigid pointcloud objects. In *4th Conference on Robot Learning (CoRL 2020)*. Massachusetts Institute of Technology, 2020. 5.2

A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal planning networks: Learning generalizable representations for visuomotor control. *Proceedings of Machine Learning Research*, 80:4732–4741, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/srinivas18b.html. 2.5.1

S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014. 4.2

J. Stria, D. Prusa, and V. Hlavac. Polygonal models for clothing. In *Conference Towards Autonomous Robotic Systems*, pages 173–184. Springer, 2014a. 3.2

J. Stria, D. Prusa, V. Hlavac, L. Wagner, V. Petrik, P. Krsek, and V. Smutny. Garment perception and its folding using a dual-arm robot. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 61–67. IEEE, 2014b. 3.2

H. Suh and R. Tedrake. The surprising effectiveness of linear models for visual foresight in object pile manipulation. *arXiv preprint arXiv:2002.09093*, 2020. 4.2

L. Sun, G. Aragon-Camarasa, P. Cockshott, S. Rogers, and J. P. Siebert. A heuristic-based approach for flattening wrinkled clothes. In *Conference Towards Autonomous Robotic Systems*, pages 148–160. Springer, 2013. 3.2

P. Sundaresan, J. Grannen, B. Thananjeyan, A. Balakrishna, M. Laskey, K. Stone, J. E. Gonzalez, and K. Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. *arXiv preprint arXiv:2003.01835*, 2020. 4.2, 5.2

Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 2.1

R. Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu. 4.2

M. Tölgyessy, M. Dekan, L. Chovanec, and P. Hubinskỳ. Evaluation of the azure kinect and its comparison to kinect v1 and kinect v2. *Sensors*, 21(2):413, 2021. B.5.2

M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. 4.2, 5.2

M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. *Robotics: Science and Systems*

Foundation, 2018. 4.1, 4.2, 5.2

D. Triantafyllou and N. A. Aspragathos. A vision system for the unfolding of highly non-rigid objects on a table by one manipulator. In *International Conference on Intelligent Robotics and Applications*, pages 509–519. Springer, 2011. 2.2, 3.2

E. Ugur and J. Piater. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2627–2633, 2015. doi: 10.1109/ICRA.2015.7139553. 5.2

B. Ummenhofer, L. Prantl, N. Thuerey, and V. Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2019. 4.2

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. 2.1

V. Viswanath, K. Shivakumar, J. Kerr, B. Thananjeyan, E. Novoseller, J. Ichnowski, A. Escontrela, M. Laskey, J. Gonzalez, and K. Goldberg. Autonomously untangling long cables. *Robotics: Science and Systems (RSS)*, 2022. 6

Z. Wang, W. Wei, G. Cong, X.-L. Li, X.-L. Mao, and M. Qiu. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 169–178, 2020. B.5.4

Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021. 5.2

A. Warrington, J. W. Lavington, A. Scibior, M. Schmidt, and F. Wood. Robust asymmetric learning in pomdps. *arXiv preprint arXiv:2012.15566*, 2020. 3.3.6

T. Weng, S. M. Bajracharya, Y. Wang, and D. Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *5th Annual Conference on Robot Learning*, 2021. 4.2, 5.2

K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021. 4.2

D. B. West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996. 2.4.2

B. Willimon, S. Birchfield, and I. Walker. Model for unfolding laundry using interactive perception. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4871–4876. IEEE, 2011. 2.2, 3.2

W. Wu, Yilin adn Yan, T. Kurutach, L. Pinto, and P. Abbeel. Learning to manipulate deformable objects without demonstrations. *Robotics Science and Systems (RSS)*, 2020. 2.2, 2.5.1.3, 2.5.2.1, 3.1, 3.2, 3.4.1, 4.2, A.2.5, B.3, B.4

F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, L. Yi, A. X. Chang, L. J. Guibas, and H. Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 2.2

A. Xie, F. Ebert, S. Levine, and C. Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv preprint arXiv:1904.05538*, 2019. 4.2

Z. Xu, C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song. Dextairity: Deformable manipulation can be a breeze. *Robotics: Science and Systems (RSS)*, 2022. 2.6, 6

M. Yan, Y. Zhu, N. Jin, and J. Bohg. Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE Robotics and Automation Letters*, pages 1–1, 2020. ISSN 2377-3774. doi: 10.1109/LRA.2020.2969931. 2.2

W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto. Learning predictive representations for deformable objects using contrastive estimation. *Conference on Robot Learning (CoRL)*, 2020. 3.1, 3.2, 3.4.1, 4.2, B.3, B.4

G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. *ICCV*, 2019. 5.3.1, D.1.1

T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019. 2.2

W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *Conference on Robot Learning*, pages 148–157. PMLR, 2022. 5.2