Model Simplification for Deformable Object Manipulation

Shengyin Wang

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds
School of Computer Science
September 2024

Some of the results and work presented in this thesis have been published in the following papers:

Wang Shengyin, Leonetti Matteo and Dogar Mehmet. "Goal-Conditioned Model Simplification for Deformable Object Manipulation." 2nd Workshop on Representing and Manipulating Deformable Objects at ICRA. IEEE, 2022.

Wang Shengyin, Papallas Rafael, Leonetti Matteo and Dogar Mehmet. "Goal-Conditioned Action Space Reduction for Deformable Object Manipulation." 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023.

Adler Aviv, Ahmad Ayah, Qiu Yulei, **Wang Shengyin**, Agboh Wisdom C., Llontop Edith, Qiu Tianshuang, Ichnowski Jeffrey, Kollar Thomas, Cheng Richard, Dogar Mehmet and Goldberg Ken. "The Teenager's Problem: Efficient Garment Decluttering as Probabilistic Set Cover." Workshop on the Algorithmic Foundations of Robotics (WAFR). Springer, 2024.

The first two publications are primarily the work of the candidate. The third publication is a collaborative effort, where the candidate contributed by implementing and solving the set cover problem, as well as designing and testing the neural network-based predictor, as detailed in the appendix.

Acknowledgements

First and foremost, I would like to express my sincere and deep gratitude to my supervisors, Dr. Mehmet Dogar and Dr. Matteo Leonetti, for your guidance, support, and insightful discussions. I feel truly fortunate to have known you and to have been your PhD student; it has been a fun and rewarding journey. I have learned so much from both of you—from high-level skills like identifying research directions to practical skills like writing papers. You taught me to be rigorous, to ask the right questions, to present effectively, to communicate clearly, to understand others, to explore different ideas, and ultimately, to become a researcher. Prior to this, I did not have much experience with computer science and robotic manipulation, and your mentorship has been invaluable.

I would like to express my sincere gratitude to my examiners, Dr. Yanlong Huang and Dr. Jihong Zhu, for their time and effort in evaluating my work. Thank you for your insightful guidance and constructive feedback during my viva. Your invaluable input has helped me refine my research and gain a deeper perspective on key academic questions. I truly appreciate your support in strengthening this thesis.

I also want to thank Dr. Rafael Papallas. I learned a great deal from you about robotics and teaching, through our everyday discussions and during the robotics module. I would like to thank all the people in the Robot Manipulation Lab and Robot Learning Lab: Wissam Bejjani, Wisdom C. Agboh, Alexia Toumpa, Chengke Sun, Zisong Xu, David Russell, Xiao Wang, Dr. Linyan Han, Shaokang Wu, Qianyi Fu, Yijin Wang, Yulei Qiu, Longrui Chen, and Chen Liu. Thank you for all the insightful discussions, for your companionship, and for the friendships.

I would also like to thank everyone I have met in the School of Computing, the Bragg Building, the University of Leeds, and at various conferences—you have all been part of this journey. Additionally, I would like to thank everyone I met on the badminton court; you made this journey more enjoyable.

Finally, I would like to thank my family for your unconditional love and constant support throughout my whole life. Completing a PhD is no easy journey, but with your encouragement, we made it together.

Abstract

This thesis presents motion planning, control, and perception methods for deformable object manipulation, with a focus on physics-based approaches. Here, a physics model or engine is used to predict the effects of robotic motions, such as how a deformable object changes shape when manipulated by a robot gripper.

Deformable object manipulation poses three major challenges. First, the configuration space of the deformable object is extremely high-dimensional, which in turn makes the action space also large. Second, the dynamics of the deformable object is highly complex; a small motion may cause the shape to change drastically, making simulation computationally expensive. Third, compared to rigid objects, deformable objects experience not only environmental occlusion but also severe self-occlusion. These challenges make motion planning time-consuming and pose significant difficulties for both control and perception.

To alleviate the computational burden of motion planning, I first propose reducing the action space for the planner based on geometric model simplification at the goal. Specifically, two implementations are developed: piece-wise line fitting for 1-D linear objects and mesh simplification for 2-D surface objects. Simulation experiments validate that with the goal-conditioned action space, the planner finds more effective trajectories more efficiently. Next, I propose simplifying the dynamics model to enable faster trajectory rollouts during motion planning. This is achieved through both goal-informed model simplification and uninformed universal simplification. Experiments confirm a significant reduction in planning time, albeit with some sacrifice in optimality. Additionally, I introduce an iterative model simplification and motion planning framework, which progressively improves the simplified dynamics model during motion planning. Experiments verify the effectiveness of the framework, demonstrating improved trajectory quality compared to planning based on a single simplified model, while remaining more efficient than planning based on the original high-fidelity, time-consuming dynamics models.

To robustly execute the planned trajectory in the real world, I improve the Constrained Deformable Coherent Point Drift (CDCPD) method for deformable object tracking. This improvement explicitly accounts for occlusion, generates a thicker point cloud by combining the current and previous point clouds, and partially updates the positions of the tracking points. Experiments demonstrate that the improved tracking method can successfully differentiate between different layers and achieves much more precise tracking than the original method.

Based on the proposed methods above, a closed-loop experimental system is established in the real world, comprising a perception subsystem, a motion planning subsystem, and a robot control subsystem. A combination of position-based control and impedance control is implemented and demonstrated to be effective for manipulating deformable objects with a robot in real-world cloth folding tasks.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the past century, robots have stepped from a concept in science fiction to real entities in the physical world. With one of its aims as helping and liberating human beings from repetitive and heavy labor work, robots have been widely implemented and achieved great success in industries, such as automobile assembly, logistics sorting, etc. However, the environment in a highly automatic factory is well calibrated, as well as the objects that the robot needs to interact with. In many of these settings, the only task that a robot is required to do is to execute a pre-programmed trajectory in a repeating pattern.

Figure 1.1: A typical room in a household environment. [1]

Another incentive for developing robots, tracing back to the start of the era, is having a household robot managing routine matters such as cleaning and doing the laundry in everyday life. An exemplary working environment for the household robot is the bedroom, as shown in Figure 1.1. Unlike the well-calibrated industrial environment, the household environment is highly unstructured and constantly changing, which makes it impossible for the robot to

[1]Image generated using generative AI from: `https://www.canva.com`

behave as it does in the factory. Instead of performing measurements and calibration once, pre-programming a trajectory, and hoping it works for all scenarios, household robots need to constantly perceive their environment, update their knowledge of the surroundings, and automatically adjust their motion plans to achieve various tasks. Along this route, various research has been conducted on developing modeling, perception, motion planning, and control methods, and achieved significant progress for rigid object manipulation in the past decades. Specifically, for pick and place based rigid object manipulation, it is already a proven and applicable technology.

However, not all objects are rigid in our daily life. From Figure 1.1 we can see that most objects in the scene are deformable, which presents particular challenges to the methods established for rigid object manipulation. Thus, in this thesis, I dedicate to conducting research on the modeling, motion planning, perception, and control methods for deformable object manipulation.

1.1 Main Themes & Challenges

Deformable object manipulation has attracted extensive interest from the robotics community in the last decade, due to their ubiquitous existence and the wide range of potential applications in both domestic and industrial scenarios. Typical deformable objects can be classified into 1-D linear (e.g., rope), 2-D surface (e.g., cloth), and 3-D volumetric (e.g., sponge) categories according to their shape and physical properties. This thesis is particularly interested in modeling, motion planning, and related perception methods for 1-D linear and 2-D surface objects because they exhibit particular deformation properties. Compared to rigid objects which have low-dimensional state representations, known dynamics, and minimal occlusion, deformable object manipulation faces unique issues and challenges.

The first challenge is that the configuration space of the deformable object is infinite. For a rigid object in 3-D space, the pose of it can be easily specified by three position coordinates and three orientation angles. The shape of a rigid object does not change during manipulation. Whereas a deformable object lacks such a concise state representation; its shape is usually specified using a large number of points, making the dimensionality forbiddingly high, and it changes significantly during manipulation. This, in turn, makes the action space large as well, as the robot can pick anywhere on the deformable object and have distinct effects. The shape of a deformable object can be approximated using a certain number of points depending on the precision required. For example, to represent the shape of a piece of square cloth, as shown in Figure 1.2, even a coarse geometric model with 6×6 points gives us a configuration space of 108 degrees of freedom.

The second challenge is that the dynamics of the deformable object is highly complex. For a rigid object, it is moving as a whole body when being manipulated, and thus comparatively easy to predict how it moves according to Newton's second law of motion. Simulating the

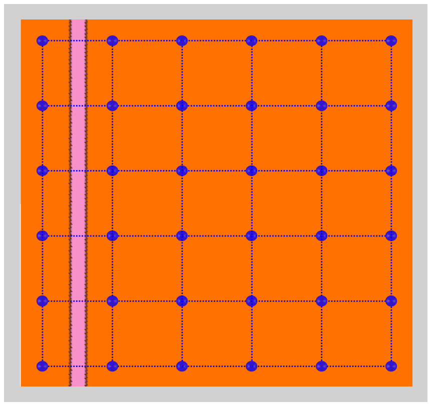


Figure 1.2: A piece of square cloth approximated by a 6×6 grid.

rigid object motion is also computationally cheap. In contrast, for deformable objects, it is much more difficult to model the dynamics. Even if we have a high-fidelity, fine-tuned model, the time cost of simulation is much higher. Besides, the deformable object behaves as an underactuated system. An action exerted on a certain part of the deformable object could cause the movements of other parts. A small motion may be able to change the shape of the deformable object drastically. Even with the same motion, if it is executed at different speeds, the outcome may be completely different.

These two challenges make it especially difficult and time-consuming for classical motion planners to plan trajectories given various goals. Typically, to achieve a specific goal, the standard approach is to first build a fine-grained model of the deformable object with thousands of particles. Then a large number of trajectories are sampled and rolled out on the built dynamics model by iteration. Next, based on the evaluation of the performance, the optimal actions are outputted and executed on the real robot. This process is time costly due to two main reasons: the action space is extremely large, a gripper can pick anywhere on the deformable object and cause distinct shape changes, which makes the sampling process inefficient; another is that the number of sampled trajectories is very large, and rolling out these trajectories on a fine model in simulation can take hours [7]. Therefore, in this thesis, I investigate alleviating this heavy burden by reducing the action space and simplifying the dynamics model adaptively to different tasks and goals.

For example, given the task of flattening a piece of cloth, picking the four corners is the most efficient way to finish the task, as shown in Figure 1.3(a). This is an easy task, and we human beings know this by instinct. The question is, how does the robot know where to pick, and what are the most efficient picking positions for other tasks? Furthermore, given the task of throwing a piece of cloth into a basket, as shown in Figure 1.3(b), instead of building a detailed model that exactly simulates how the cloth deforms, we only need to consider how the center of mass of the cloth moves in space. As long as it ends up inside the basket, the task is

(a) Robot flattening a cloth. (b) Robot throwing a towel.

Figure 1.3: Example scenarios where reduced action space and simplified dynamics model could help to plan more efficiently. The first image shows a robot flattening a piece of cloth, focusing on manipulating only the visible corners. The second image depicts a robot throwing a towel into the basket, a task where fine modeling of shape changes is unnecessary. A simplified model that ensures the towel ends up inside the basket is sufficient for successful execution. [2]

deemed a success. This is a trivial example of using a coarse model; how can the robot build a simplified model for various tasks or goals? What if the simplified model does not match with the original dynamics model? These are the key questions that I address in this thesis to establish an efficient planning framework for deformable object manipulation.

Another challenge is that, when tracking with a camera, deformable objects suffer more from occlusion. For a rigid object, as long as it is not fully occluded, it is relatively easy to deduce the pose of it by analyzing the exposed features. In contrast, for a deformable object, there are many possible shapes hidden behind the obstruction, and it is difficult to decide what shape it is given only one observation. Moreover, the deformable object often presents severe self-occlusion, while the rigid object does not have at all. These occlusion issues make it a challenging task to perceive and estimate the state of a deformable object, which is fundamental to deformable object manipulation.

Thus, in this thesis, I focus on developing methodologies to resolve these challenges for deformable object manipulation, with particular emphasis on resolving motion planning related problems.

1.2 Aims

Although facing a bunch of difficulties and challenges, I envision that dealing with deformable objects is an indispensable ability for a household robot. Thus, the ultimate goal of this thesis is to investigate various methodologies for deformable object manipulation, including motion planning, and related modeling, perception, and control methods, with a focus on 1-D linear and 2-D surface objects. I aim to overcome the motion planning challenges when manipulating de-

[2]Images generated using generative AI from: `https://www.canva.com`

formable objects and develop an efficient motion planning framework that can adaptively bring the deformable object from the initial configuration to various goal configurations. Specifically, I aim to find concise representations, extract efficient action spaces, build simplified dynamics models, and make better use of them to reduce the time cost for deformable object manipulation planning. Additionally, I also aim to alleviate the occlusion issues for the perception and build a closed-loop deformable object manipulation system with motion planning, shape tracking, and execution.

1.3 Contributions

The contributions of this thesis are as follows:

1. A general implementation of goal-conditioned geometric model simplification approach for 1-D linear deformable objects (e.g., ropes) that uses a piece-wise line fitted model, and for 2-D surface deformable objects (e.g., cloth) that uses a mesh simplified model (Chapter 3).
2. An action space reduction scheme integrated into a manipulation planning pipeline based on the simplified geometric model of the deformable objects (Chapter 3).
3. A systematic approach for building a simplified dynamics model given the original dynamics model and the simplified geometric model (Chapter 4).
4. A closed-loop experimental system consisting of perception, planning, and control in the real world for a Franka Panda robot folding deformable cloth (Chapter 4).
5. An iterative model simplification & motion planning framework that generates manipulation plans much faster and shows the advantage of balancing between the computational cost and trajectory optimality (Chapter 5).
6. Extensive studies of the proposed action space reduction, model simplification, and motion planning framework in simulation for a set of representative tasks on ropes and cloth using both sampling-based and optimization-based motion planning methods (Chapter 3, Chapter 4, and Chapter 5).
7. An extension of the proposed approaches to multi-step planning with intermediate goal configurations, where action space reduction and model simplification are performed on successive intermediate goals (Chapter 3, Chapter 4, and Chapter 5).

In addition to the contributions outlined above, which are primarily my own work, I also collaborated with other researchers on related problems in deformable object manipulation during the course of my PhD. Particularly, we worked on a project focusing on grasping and decluttering multiple deformable objects, referred to as the teenager's problem. In this collaboration, I contributed by implementing a set cover solver and designing a neural network-based predictor. Additionally, I conducted some preliminary work on learning-based bimanual deformable object manipulation. Both of these efforts are detailed in the Appendix.

1.4 Roadmap

In this chapter, I introduce the main themes and challenges, and describe the aims and contributions; In Chapter 2, I give a review of related works and an analysis of the research gap; In Chapter 3, I illustrate the research on geometric model simplification and goal-conditioned action space reduction for deformable object manipulation; In Chapter 4, I present the dynamics model simplification methods, the improved shape tracking method, and a closed-loop deformable object manipulation system for real robot experiments; In Chapter 5, I exhibit an iterative model simplification and motion planning framework that is able to balance the efficiency and optimality for deformable object manipulation. In Chapter 6, I summarize the work of this thesis and indicate potential future research directions.

Chapter 2

Related Work

This chapter serves two purposes: it provides a literature review of related work and a tutorial on relevant methodologies for deformable object manipulation.

First, in Section 2.1, I present a brief overview of deformable object manipulation and introduce developments in this field.

Modeling deformable objects is an important foundational technique that helps us understand and develop relevant planning and control methods. In Section 2.2, I provide an overview of several modeling methods that have been proposed and extensively utilized in deformable object manipulation.

Physics engines and simulators are widely used in robotics and have been a major driving force behind technological advancements in recent years. This is particularly true for the manipulation of deformable objects, as there is no concise and explicit physical model available. In Section 2.3, I give a brief introduction and comparison of simulators that support the simulation of deformable objects.

In contrast to rigid objects, the perception, state estimation, and tracking of deformable objects present significant challenges, yet they are indispensable components of robotic manipulation. In Section 2.4, I survey various methods that have been proposed for the perception of deformable objects.

Motion planning and control have been widely researched and extensively applied in various domains of robotic manipulation, especially for rigid objects. In Section 2.5, I provide a general introduction to the fundamental concepts of motion planning and present several representative motion planning algorithms, along with some related motion control methods. This is followed by more related work on planning-based deformable object manipulation in Section 2.6.

State representation and action space design are crucial in motion planning for deformable object manipulation, which is a primary focus of this thesis. In Section 2.7, I compare and analyze related methods for concise state representation and action space reduction.

Although high-fidelity models can more accurately simulate the shape changes of deformable objects in the real world, they are also relatively time-consuming. Approximate or simplified models remain popular among researchers due to their efficiency and are employed to address various robotic manipulation challenges. In Section 2.8, I provide a detailed overview of related work on model order reduction techniques.

2.1 Overview of Deformable Object Manipulation

Deformable object manipulation has gained extensive interest from both industry and academia in recent years due to the ubiquitous presence of deformable objects in our daily lives and their wide range of potential applications [3, 4, 8, 9]. However, compared to rigid objects, deformable objects present significant challenges for manipulation. Specifically, the state of a rigid object can be described using three position parameters and three orientation parameters, whereas deformable objects require high-dimensional parameters to accurately represent their state [10]. The dynamics of rigid objects are relatively simple, as their shape does not change and they often do not interact with the environment. In contrast, the dynamics of deformable objects are highly complex [11]; changes in one part can affect the entire object, leading to significant motion and deformation under external forces. Rigid objects rarely face issues of self-occlusion, whereas deformable objects frequently suffer from severe self-occlusion [12], making it challenging to accurately estimate their state and precisely manipulate them. These challenges complicate the planning, perception, and control of deformable objects. Therefore, in this thesis, I focus on solving the motion planning problem for robotic manipulation of deformable objects, aiming to make it more efficient. I also develop corresponding perception and control methods.

Deformable objects can be categorized in various ways based on different criteria. In the field of robotics, they are primarily classified into three types based on their shape and dimensional characteristics: 1-D (linear) [13, 14, 15], 2-D (surface) [16, 17, 18], and 3-D (volumetric) [19, 20, 21] deformable objects. 1-D deformable objects [13] are those in which one dimension is significantly larger than the other two. Examples include cables and ropes, which can stretch, contract, and bend under tensile or compressive forces. 2-D deformable objects [16] have two primary dimensions, with thickness—the third dimension—being negligible for manipulation purposes. Examples include cloth, thin shells, and paper. 3-D deformable objects [19] deform in three-dimensional space, undergoing compression, stretching, and bending in all three dimensions. Examples include rubber balls and human tissues, which deform under forces in various directions. While 3-D deformable objects deform in all dimensions, the manipulation of 1-D and 2-D deformable objects often presents unique challenges due to their complex deformation characteristics [9, 22]. Therefore, this thesis primarily focuses on the manipulation of 1-D and 2-D deformable objects.

Deformable object manipulation has wide applications, including packaging [23], cable routing [24], minimally invasive surgery [25], agricultural harvesting [26], food preparation [27], elderly care [28], and household management [2]. Chen et al. [23] researched the singulation of layers of bags for potential packaging applications, experimenting with bags of various shapes, materials, and sizes. Jin et al. [24] proposed a hierarchical planning and control framework using RGB-D images for cable routing with environmental constraints. Hu et al. [25] conducted research on human soft tissue manipulation with novel deformation modeling and shape control methods, achieving better precision in less time under occlusion. Ray et al. [29] proposed a spread-and-pick strategy for separating a certain amount of homogeneous fresh herbs. Zhu et al. [28] presented a bimanual robot system to perform dressing assistance tasks cooperatively, which has the potential to improve the quality of life for the elderly and disabled population. Additional work focusing on household management tasks such as rope tying [30], bed-making [31], cloth flattening [32], cloth folding [33], and cloth decluttering [2] has been researched, highlighting essential abilities that a household robot should possess. This thesis is primarily dedicated to advancing the development of domestic robots, which should possess the ability to deal with deformable objects such as ropes and cloth.

2.2 Deformable Object Modeling

Deformable object modeling plays a crucial role in robotic manipulation, providing an essential foundation for planning, perception, and control. Various modeling methods for deformable objects have been extensively studied across different fields. For example, in mechanics and material science, constitutive models—often solved using numerical techniques like the Finite Element Method (FEM) [34]—are commonly employed to create highly accurate representations of deformable objects, precisely simulating their deformation and motion under external forces. In contrast, in the computer graphics and computer vision communities, particle-based models such as Mass-Spring Systems (MSS) [35] and Position-Based Dynamics (PBD) [36] are widely used, offering visually realistic deformation effects with better computational efficiency. In robotics, the criteria for selecting an appropriate model include physical accuracy, visual plausibility, simplicity, intuitiveness, and computational complexity. Therefore, I will briefly introduce some of the widely accepted dynamics models for deformable objects, including constitutive models and particle-based models.

2.2.1 Constitutive Models

Constitutive models [37] are mathematical descriptions that define how materials respond to forces by relating stress to strain, which is essential for predicting material behavior under various conditions.

In this subsection, I briefly introduce the FEM, a numerical technique commonly used alongside constitutive models to simulate deformable objects.

Finite Element Method

The FEM [38, 39, 40] is a standard computational method for solving continuum mechanics problems. It aims to approximate the true physical behavior of a deformable object by dividing its body into smaller and simpler parts called finite elements. The shape and size of these elements can vary (e.g., triangles, quadrilaterals, tetrahedrons) depending on the complexity of the geometry and the type of analysis.

Within each element, the governing equations (such as equilibrium equations and constitutive relations) are formulated in a weak form and discretized using appropriate shape functions. This process converts partial differential equations into algebraic equations that are easier to solve numerically. The elements are then connected through nodes that form a mesh grid, which can be structured or unstructured. The deformation of the mesh is calculated through the displacements of these nodes, forming a displacement vector field $\boldsymbol{u}$.

To decrease computational complexity, dynamic effects are often neglected, and the deformation is calculated under the assumption of static equilibrium. This results in a system of algebraic equations for each element, relating nodal displacements and forces. The relationship for a finite element e with N_e nodes can be expressed as:

$$\boldsymbol{K}_e \boldsymbol{u}_e = \boldsymbol{f}_e \tag{2.1}$$

where $\boldsymbol{K}_e \in \mathbb{R}^{3N_e \times 3N_e}$ is the element stiffness matrix, $\boldsymbol{u}_e \in \mathbb{R}^{3\times N_e}$ is the vector of nodal displacements of element e, and $\boldsymbol{f}_e \in \mathbb{R}^{3\times N_e}$ is the vector of nodal forces of element e. For the entire mesh with N nodes, the global stiffness matrix $\boldsymbol{K}$ is assembled by appropriately combining the element stiffness matrices according to the mesh connectivity:

$$\boldsymbol{K} = \sum_e \boldsymbol{K}_e \tag{2.2}$$

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} \tag{2.3}$$

where $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{u}$ is the vector of unknown nodal displacements (or other quantities), and $\boldsymbol{f}$ is the vector of applied loads or forces. The matrix $\boldsymbol{K}$ depends on the nodal displacements $\boldsymbol{u} \in \mathbb{R}^{3\times N}$ and the constitutive material properties (e.g., Young's modulus E and Poisson's ratio v) to compute the nodal forces $\boldsymbol{f} \in \mathbb{R}^{3\times N}$ of the entire mesh. Therefore, this huge matrix $\boldsymbol{K}$ must be calculated at every time step t.

Boundary conditions and constraints (e.g., fixed supports, prescribed displacements, or loads) are applied to the global system by modifying the global stiffness matrix and the load vector to account for the specified conditions. This ensures that the solution satisfies both the equilibrium equations and the imposed constraints.

The global system of equations is typically large and sparse. It is solved using numerical methods such as direct solvers (e.g., Gaussian elimination) or iterative solvers (e.g., Conjugate Gradient method) to find the unknown nodal displacements. The results obtained from FEM simulations are often validated against experimental data or analytical solutions to ensure their reliability.

FEM has the advantage of producing physically realistic simulations and accurately modeling complex deformations [41], making it suitable for applications where precision is a top priority. Model parameters have clear physical interpretations [42], aiding in understanding and tuning simulations. However, FEM can be computationally intensive, especially for large-scale problems with fine meshes, due to the size of the global stiffness matrix and the computational cost of solving the system of equations. This makes achieving real-time performance with FEM challenging.

2.2.2 Particle-Based Models

Instead of simulating deformable objects as continuum models, another approach is to represent the objects as discrete particles interconnected by constraints. This method is intuitive, reflecting the fundamental nature of our world, which is composed of atoms and basic particles. Depending on the type of connections and how interactions are computed between the particles, particle-based models include MSS and PBD.

Mass Spring System

An MSS is composed of a network of mass points (particles) and springs [35, 43]. It is a force-based modeling method, and for each particle in the system, the motion is governed by Newton's second law:

$$m_i \ddot{\boldsymbol{p}}_i = \boldsymbol{f}_i \tag{2.4}$$

where m_i denotes the mass of particle i, $\boldsymbol{p}_i$ is its position, and $\boldsymbol{f}_i$ represents the total force acting on it.

The forces that affect the motion of the particles include external forces $\boldsymbol{f}_i^{ext}$ such as gravity, and internal forces such as the spring force $\boldsymbol{f}_i^s$, and the damping force $\boldsymbol{f}_i^d$:

$$\boldsymbol{f}_i^s = k_s(|(\boldsymbol{p}_j - \boldsymbol{p}_i)| - l_{ij})\frac{(\boldsymbol{p}_j - \boldsymbol{p}_i)}{|(\boldsymbol{p}_j - \boldsymbol{p}_i)|} \tag{2.5}$$

$$\boldsymbol{f}_i^d = k_d(\dot{\boldsymbol{p}}_j - \dot{\boldsymbol{p}}_i) \tag{2.6}$$

Here, k_s is the stiffness coefficient, k_d is the damping coefficient, l_{ij} is the rest length of the spring between particles i and j, and the indices i and j represent connected particles.

Then the Equation 2.4 can be rewritten as:

$$m_i \ddot{\boldsymbol{p}}_i = \boldsymbol{f}_i^{ext} + \boldsymbol{f}_i^s + \boldsymbol{f}_i^d \tag{2.7}$$

For the entire mass-spring system, the equation in matrix form becomes:

$$\boldsymbol{M}\ddot{\boldsymbol{p}} + \boldsymbol{D}\dot{\boldsymbol{p}} + \boldsymbol{K}\Delta\boldsymbol{p} = \boldsymbol{f}_{ext} \tag{2.8}$$

where $\boldsymbol{M} \in \mathbb{R}^{3N\times 3N}$ is the mass matrix; $\boldsymbol{D} \in \mathbb{R}^{3N\times 3N}$ is the damping matrix; $\boldsymbol{K} \in \mathbb{R}^{3N\times 3N}$ is the stiffness matrix; N is the number of particles; and $\ddot{\boldsymbol{p}}$, $\dot{\boldsymbol{p}}$, and $\Delta\boldsymbol{p}$ are the acceleration, velocity, and displacement vectors of all particles, respectively. Besides, $\boldsymbol{M}$ is determined by the stiffness coefficient k_s, and $\boldsymbol{D}$ is determined by the damping coefficient k_d, both of which are tunable to adjust the deformability of the object.

Based on the motion models for the mass spring system constructed above, various numerical integration methods can be employed to perform the integration [44, 45, 46], including the explicit Euler method, Runge-Kutta method, backward Euler method, etc.

MSS is widely used as the physics-based model for various deformable objects because it is intuitive, simple to implement, and computationally more efficient than the constitutive model. The disadvantage is that it is mostly used to simulate small deformation and simple elastic effects. A high-fidelity model is time-consuming and not guaranteed to converge to the correct behavior. Additionally, the coefficients affect the performance greatly and are difficult to tune to achieve the desired dynamical behavior.

Position Based Dynamics

Unlike MSS, which relies on forces and the integration of equations of motion, PBD [36] directly computes particle positions by applying geometric constraints. This offers better controllability and avoids the overshooting issues common in explicit integration schemes, making it increasingly popular for simulating deformable objects [47, 48, 49, 50].

In PBD, an object is also represented by particles, but there are no springs connecting them. When the particle system is disturbed by external forces, it tends to adopt a configuration that does not preserve the original shape, referred to as the disturbed configuration:

$$\bar{\boldsymbol{p}}_i = \boldsymbol{p}_i^{t-1} + \bar{\boldsymbol{v}}_i^t \Delta t \tag{2.9}$$

where $\bar{\boldsymbol{v}}_i = \dot{\boldsymbol{p}}_i^{t-1} + \boldsymbol{f}^{ext}\frac{\Delta t}{m_i}$, $\boldsymbol{p}_i^{t-1}$ is the previous position, $\dot{\boldsymbol{p}}_i^{t-1}$ is the previous velocity, $\boldsymbol{f}^{ext}$ is the external force (e.g., gravity), m_i is the mass of particle i, and Δt is the time step.

To preserve the original shape of the object, an optimal linear transformation between the initial shape $\boldsymbol{p}^0$ and the intermediate deformed shape $\bar{\boldsymbol{p}}$ is calculated as follows:

$$\boldsymbol{A} = (\sum_i m_i \boldsymbol{r}_i \boldsymbol{q}_i)(\sum_i m_i \boldsymbol{q}_i \boldsymbol{q}_i)^{-1} \tag{2.10}$$

where $\boldsymbol{r}_i = \bar{\boldsymbol{p}}_i - \bar{\boldsymbol{c}}$, $\boldsymbol{q}_i = \boldsymbol{p}_i^0 - \boldsymbol{c}_0$, with $\boldsymbol{c} = \frac{1}{\sum_i^N m_i}\sum_i^N m_i \boldsymbol{p}_i$ being the center of mass of the deformed configuration, and $\boldsymbol{c}_0$ being the center of mass of the initial configuration. Furthermore, the linear transformation matrix can be divided into rotational and symmetric parts:

$$\boldsymbol{A} = \boldsymbol{R}\boldsymbol{S} \tag{2.11}$$

where $\boldsymbol{R}$ represents rigid behavior and $\boldsymbol{S}$ represents deformable behavior.

Thus, the goal(or corrected) position of the particle is given by:

$$\boldsymbol{p}_i = ((1-\beta)\boldsymbol{R} + \beta\boldsymbol{A})\boldsymbol{q}_i + \boldsymbol{t} \tag{2.12}$$

where $\boldsymbol{t} = \boldsymbol{c}$ is the translation of the object, $\beta \in [0, 1]$ is the control parameter determining the degree of deformation introduced by matrix $\boldsymbol{S}$, and as β approaches 1, the degree of deformability increases.

Finally, the new position and velocity for the next time step are updated:

$$\boldsymbol{p}_i^t = \bar{\boldsymbol{p}}_i + \alpha(\boldsymbol{p}_i - \bar{\boldsymbol{p}}_i) \tag{2.13}$$

$$\boldsymbol{v}_i^t = (\boldsymbol{p}_i^t - \boldsymbol{p}_i^{t-1})/\Delta t \tag{2.14}$$

where α affects the stiffness of the model and determines the speed of convergence of the intermediate position to the goal position.

PBD is more efficient and offers more stable simulation for both deformable objects and rigid objects than constitutive models and MSS-based models. The underlying particle system in PBD allows for better scalability for parallel computation. Additionally, PBD can simulate grippers, deformable objects, and their interactions within a unified model, which MSS cannot. Moreover, PBD produces more visually realistic deformations compared to MSS models.

However, PBD is physically less accurate than constitutive models because it is geometrically motivated rather than based on physical laws. The parameters in PBD do not have direct physical meanings, requiring extensive tuning to achieve satisfactory results. Although PBD is more efficient than FEM and MSS, the computational cost for a high-fidelity model with thousands of particles remains prohibitively expensive for motion planners.

2.2.3 Summary of Different Modeling Methods

To summarize, the modeling methods discussed above each have their respective merits and shortcomings, as outlined in Table 2.1. FEM-based methods excel in modeling the behavior of deformable objects with high precision. Their parameters, such as Young's modulus, have interpretable meanings in the real world, allowing for tuning through material experiments. However, FEM is computationally slow and struggles with objects that undergo large deformations during manipulation. Consequently, most robotics simulators do not support FEM.

MSS, on the other hand, are comparatively easy to implement and much faster to simulate than FEM. However, the parameters of the masses and springs lack physical interpretability, making them difficult to tune. While MSS works well for small deformations, it may not accurately capture the behavior of deformable objects undergoing large deformations. Low-fidelity MSS models fail to provide accurate dynamics, whereas high-fidelity MSS models do not necessarily offer better precision and often fail to meet the efficiency required for fast trajectory rollouts in motion planners.

PBD improves upon MSS by being easier to parallelize and producing more stable simulations than explicit integration methods used in MSS-based dynamics. PBD supports modeling both rigid and deformable objects, making it suitable for studying interactions between these types of objects. It also generates visually realistic effects that closely resemble real-world behavior. However, the parameters in PBD lack physical meaning and require extensive tuning to achieve satisfactory results. Although PBD is more efficient than FEM and MSS, the computational cost for a high-fidelity model with thousands of particles remains prohibitively expensive for motion planners.

Given these considerations, the methods proposed in this thesis are based on particle-based modeling approaches, including MSS and PBD. It is important to note, however, that the ideas and methods presented in this thesis can also be extended to FEM-based modeling approaches.

Table 2.1: Overview of deformable objects modeling approaches [3]

	Advantages	Limitations	Modeling applications
FEM	Physical fidelity & interpretability	Complex & expensive to compute (nonlinear)	Rods & cables
			Fabrics
		Not well integrated to robotics simulator well yet	Food
			Tissues
MSS	Fast	Inaccurate for large deformation	Ropes
			Fabrics
	Simple to implement	Lacking physical interpretability	Sponges
			Rubber spheres
PBD	Fast and stable	Visually fidelity only	Paper
			Fabrics
	Supports modelling of various objects	Lacking physical interpretability	Cushion
			Liquids

2.3 Deformable Object Simulators

Physics simulators have significantly advanced the field of robotics and continue to play a fundamental role across various areas. By definition, a simulator is an application that includes several core functionalities: 1) a physics engine for modeling realistic physical behavior, 2) collision detection and friction models, 3) a graphical user interface (GUI), 4) the capability to import scenes and meshes, and 5) an API for specific programming languages (e.g., C++/Python).

According to this definition, there are plenty of simulators capable of producing kinematic and dynamic behaviors, such as MuJoCo [51] and PyBullet [52]. However, while most simulators are designed for rigid bodies, those that support deformable object manipulation are scarce due to the inherent complexity of simulating such objects.

Table 2.2 lists commonly used robotics simulators that support deformable object simulation. The table includes information about the underlying physics engine, the dynamics modeling method, whether robot integration is supported, the grasping type for deformable objects (P for point-based grasp, L for line-based grasp), and whether the simulator supports CPU or GPU computation.

MuJoCo (Multi-Joint dynamics with Contact) [51] is a high-performance physics engine renowned for simulating complex physical systems in robotics. It uses MSS as the underlying modeling method for deformable objects and can simulate interactions between rigid robots and deformable objects. While it excels at handling rigid body dynamics and contact-rich interactions, its capability to simulate deformable objects is limited compared to specialized soft-body simulators.

PyBullet [52] is an open-source physics simulator widely used for robotics, machine learning, and reinforcement learning, known for its ease of use and integration with Python. While PyBullet includes basic support for deformable object simulation (e.g., cloth and soft body physics), it is less advanced in this area, offering less visual realism and limited support for robot-deformable object interaction. An example of a Franka Panda robot interacting with a piece of cloth is shown in Figure 2.1.

SOFA (Simulation Open Framework Architecture) [53] is an open-source framework specifically designed for interactive physics simulation based on FEM, with an emphasis on medical and robotic applications. Its modular architecture allows users to customize and combine different models and solvers. SOFA is widely recognized for its realistic simulation of soft tissue dynamics, making it a popular choice in the field of robotic surgery.

Table 2.2: Robotics simulators and their ability to handle deformable objects [4, 5, 6]

Simulators	Physics engine	Dynamics model	Robot integration	Grasp	CPU/GPU
Mujoco	Mujoco	MSS	Yes	P/L	CPU
PyBullet	Bullet	MSS	Yes	P/L	CPU
SOFA	SOFA	FEM	No	N/A	CPU
SoftGym	Flex	PBD	No	P	GPU

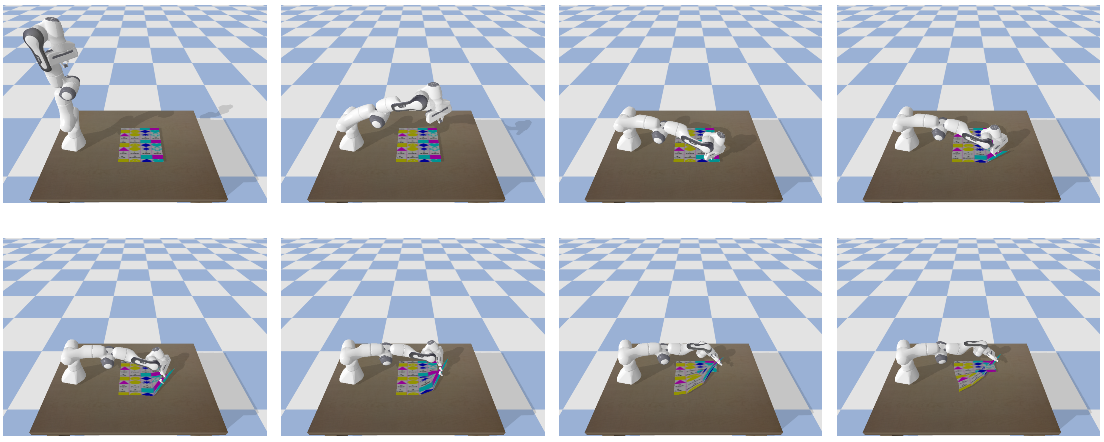


Figure 2.1: Simulation of a Franka Panda robot folding cloth in PyBullet. The gripper does not physically grasp the cloth but manipulates it by anchoring to specific particles on the cloth.

SoftGym [54] is a recently developed simulator based on Nvidia's open-source physics engine Flex. It includes a set of benchmark deformation tasks covering rope, cloth, and liquids, with exceptional simulation fidelity. Nonetheless, it is currently lacking the ability to add robot models into the simulation, which requires extra permission from Nvidia.

Given considerations of visual realism, efficiency, and computational performance, in this thesis, I primarily use SoftGym to test the proposed ideas by solving the provided benchmark tasks. I also use it as the underlying simulator and physics engine for rolling out trajectories within the planner.

2.4 Deformable Object Perception: State Estimation & Tracking

State estimation and tracking of deformable objects are critical for downstream tasks such as planning and control. Unlike rigid objects, which can typically be described using three Cartesian coordinates and three orientation parameters, deformable objects require significantly more complex state representations. Their state is determined by the positions of numerous points in space, often requiring a high-dimensional or even infinite-dimensional representation to accurately describe their shape and pose. Additionally, the estimation process is highly susceptible to occlusions, including self-occlusion, further complicating the accurate assessment of the object's full configuration.

To address these challenges, researchers in computer vision, graphics, and robotics have developed a variety of methods for state estimation and tracking of deformable objects [9]. Some approaches involve attaching markers [55, 56, 57] to the object and tracking their positions to infer the overall state. However, this method may alter the object's dynamic properties. Other techniques rely on visual pattern recognition [58] for state estimation, which can be effective for specific object types with distinctive visual patterns. Moreover, template matching methods [59] have been proposed to match sensor-extracted features with predefined templates, though these techniques often lack generalizability.

For markerless tracking and estimation, some researchers frame the problem as a non-rigid registration process using point clouds as input [60]. The challenge in these cases is to find a nonlinear warping function that maps one surface, image, or point cloud onto another. Tang et al. [61] developed a real-time state estimator for deformable linear objects using coherent point drift (CPD), which demonstrated robustness under occlusions. They further improved the estimator with physics simulation [62], incorporating a Gaussian mixture model for registration, CPD for maintaining topology under occlusion, and a dynamics simulation to refine the estimation and ensure feasibility.

Schulman et al. [63] explored the use of a physics engine to track the shape of deformable objects like ropes and cloth. They proposed a modified expectation-maximization algorithm that updates the posterior estimation based on sequential point cloud inputs and the physical properties of the object and environment. However, the need for accurate object and environment properties makes this approach less suitable for novel environments with unknown objects, and the reliance on physics simulation can hinder real-time tracking.

Alternatively, Chi and Berenson [64] proposed the Constrained Deformable Coherent Point Drift (CDCPD) framework, which eliminates the need for physics simulation and is robust to occlusions. CDCPD builds upon CPD by using locally linear embedding and constrained optimization to maintain consistency under occlusion. It can also detect tracking failures during manipulation and automatically recover to the most relevant previous state. Wang et al. [65] further enhanced this framework by incorporating a motion model and convex geometric constraints to prevent self-intersection and penetration into obstacles.

Learning-based methods have also been explored for estimating and tracking deformable objects. Chi and Song [66] proposed GarmentNets, which estimates the shape of garments even under severe self-occlusion. Their approach first maps the perceived point cloud to a canonical state, completes the garment state in this space, and then warps it back to the observation space as a complete 3D mesh. However, this method only estimates state from a single image, without considering sequence correspondence, and requires the robot to pick up the garment before estimation. Xue et al. [67] extended GarmentNets to GarmentTracking, an end-to-end system that continuously updates the garment's state using a sequence of point clouds. They developed a VR-based recording system to collect and label data in simulation, though this approach demands specific equipment, limiting its practical deployment.

In this thesis, considering the challenges of using markers or visual patterns and the difficulty of collecting large datasets for training an end-to-end system, I establish a tracking system based on CDCPD. Several improvements are incorporated to enhance its ability to accurately track the shape changes of deformable objects during manipulation, meeting the real-time requirements for robotic tasks.

2.5 Motion Planning & Control

Motion planning and control are fundamental components in robotics, enabling robots to interact with their environment and accomplish tasks effectively. Motion planning involves generating a path or trajectory that guides the robot toward a specific goal while adhering to constraints such as obstacle avoidance. Control methods ensure that the robot accurately follows this planned path by adjusting its movements in real time, compensating for any disturbances or uncertainties.

In this section, I introduce the basic concepts of motion planning and present several representative algorithms that will be utilized in the following chapters. Additionally, I provide a brief overview of the control methods employed in real-world robotic experiments.

2.5.1 Basics of Motion Planning

Traditional motion planning primarily focuses on finding collision-free paths within the state space, often applied to low-dimensional navigation tasks where strict avoidance of environmental contact is required. In contrast, robotic manipulation planning involves high-dimensional problems that encompass not only geometric considerations but also physical interactions with objects. To comprehensively understand this field, it is essential to explore key concepts such as state space, action space, geometric-based motion planning, and physics-based motion planning.

State Space

In motion planning [68, 69], the state space represents all possible configurations of a system. For a robotic system, this typically includes parameters such as position, orientation, joint angles, and velocities. In manipulation planning, the state space can also encompass the position, orientation, velocity, and shape of the object being manipulated. This creates a high-dimensional space where each point represents a specific state of the system at a given time.

The complexity of the state space increases with the degrees of freedom of the robot and the manipulated object. For example, a robotic arm with six degrees of freedom has a six-dimensional state space. When multiple objects or objects with complex shapes are considered, the dimensionality rises further, making the motion planning problem computationally challenging.

Action Space

The action space, or control space, consists of all possible control commands or actions that a robot can execute, determining how the robot moves within its environment. The specific nature of the action space depends on the robot type; for example, the action space of a mobile robot might include linear and angular velocities, while for a robotic arm, it could involve end-effector poses or joint velocities.

Traditionally, motion planning is performed in the state space [70, 71], where actions are inferred from transitions between start and goal states. However, some planners operate directly in the action space [72, 73]. This approach has several advantages: the action space typically has a lower dimensionality than the state space, making it more suitable for high-dimensional or complexly constrained tasks. Moreover, planning in the action space ensures that actions correspond directly to executable commands, thus avoiding infeasible state transitions. Given the complexity of deformable object manipulation, this thesis adopts planning in the action space and evaluates trajectories using established dynamics models to achieve feasible and efficient plans.

Geometric and Physics-based Motion Planning

Motion planning problems can be broadly categorized into geometric-based [74, 75, 76] and physics-based approaches [77, 78, 79, 80].

Geometric-based motion planning aims to find a collision-free path for the robot to move from an initial state to a goal state without considering the dynamics of the robot or interactions with the environment. This approach is well-established and suitable for scenarios where dynamic interactions are minimal. In the real-robot experiments conducted for this thesis, I use geometric-based motion planning to guide the robot between states when interactions with deformable objects are not a concern. Specifically, the motion planning module from MoveIt [81] is employed to accomplish these tasks.

Physics-based motion planning, on the other hand, involves interactions with objects or the environment, often accounting for the dynamics of both the robot and the manipulated objects. This approach is essential when the robot must interact physically with the environment, such as manipulating deformable objects where the object's dynamics significantly influence the outcome. Consequently, most of the efforts in this thesis are dedicated to addressing the challenges associated with physics-based motion planning in the context of deformable object manipulation.

Planners

A planner is an algorithm or software that generates a feasible path or trajectory for a robot or autonomous system to follow from an initial state to a goal state [74, 75, 82, 83]. The planner takes into account the environment in which the robot resides, as well as constraints and dynamics, ensuring that the generated path avoids obstacles, adheres to physical limitations, and satisfies the task requirements. There are various types of planners, such as optimization-based planners (e.g., Cross-Entropy Method, CEM [84]) and sampling-based planners (e.g., Rapidly-exploring Random Tree, RRT [75]), each suited to different problem complexities and environments, which I will introduce in the following sections.

2.5.2 Sampling-based Motion Planning

Sampling-based methods [69, 85] are widely used for various motion planning problems due to their effectiveness in handling challenging planning scenarios, particularly those involving high-dimensional state spaces. The core idea of these planners is to build a graph or tree that approximates the connectivity of the state space, where nodes represent sampled states and edges denote valid transitions between these states. The process typically begins by randomly sampling states within the state space. Each sampled state is evaluated for feasibility, which involves checking whether it can be connected to existing states in the graph or tree through valid, collision-free paths. Alternatively, some planners sample feasible actions first and then expand the state accordingly, which helps maintain the constraints of the states and ensures valid transitions. New nodes and edges are added based on these connections, allowing the graph or tree to expand as the algorithm explores the state space. This exploration continues iteratively until a path from the initial state to the goal state is discovered or a predefined stopping condition is reached.

Two of the most common sampling-based planning algorithms are the Probabilistic Roadmap Method (PRM) [86, 87] and the RRT [75], which has inspired many other planning algorithms, including RRT* [88], RRT-Connect [89], and others.

PRM [74] operates through two main phases: a learning phase and a query phase. During the learning phase, the state space is sampled over time, retaining valid samples while discarding those that violate constraints. This phase generates a probabilistic roadmap, which is sometimes referred to as a forest, analogous to the trees in RRT. In the query phase, specific start and goal states are defined and connected to the existing roadmap. Multiple queries can be made simultaneously; thus, PRM is also referred to as a multi-query planner. This design allows PRM to efficiently solve different planning problems within the same environment. The primary computational cost in PRM is spent on sampling and constructing the roadmap, while solving queries is comparatively faster.

RRT [75, 69] represents another category of sampling-based planners, particularly suited for single-query problems. Unlike PRM, which requires a learning phase to pre-sample the state space and construct a roadmap, RRT operates by incrementally building a tree from the start state toward the goal state, or vice versa. During each iteration, RRT selects a random new state in the state space and attempts to connect it to the nearest vertex in the tree. If the new state is within the free space and a collision check confirms a clear path, the edge is added to the tree. Even after an initial path is found, RRT continues to explore until either a predefined time limit is reached or a fixed number of iterations is completed. In this thesis, I experiment with RRT within the planning framework to demonstrate the applicability of our methods to sampling-based motion planners.

Although sampling-based methods are effective for navigating complex and high-dimensional spaces and widely used in robotics and autonomous systems, they often produce suboptimal solutions and only provide probabilistic completeness. This means that while the algorithm will find a solution if one exists given enough time, it does not guarantee optimality. Additionally, the trajectories generated by these methods are often noisy and may require smoothing through post-processing. In contrast, optimization-based planners address these issues by offering more refined solutions, as discussed next.

2.5.3 Optimization-based Motion Planning

Optimization-based motion planning [90] is another category of motion planning methods that is widely used for its ability to handle complex constraints and produce smooth, feasible trajectories. Unlike sampling-based methods that explore state space to find feasible paths, optimization-based planning methods aim to minimize a cost function—such as achieving a goal state, reducing path length, or conserving energy—while satisfying constraints like collision avoidance and dynamic feasibility. This makes them particularly effective in environments where the quality of the path and adherence to physical constraints are critical.

A general formulation of trajectory optimization includes several components: the cost function, state and control boundaries, obstacle avoidance functions, dynamics transition functions, and other constraints. For instance, if a trajectory $\tau \in \mathcal{T}$ is defined as the action sequence applied to the system, a cost function $\mathcal{F} : \mathcal{T} \to \mathbb{R}^+$ can be used to evaluate the trajectory, mapping it to a real value that represents the cost associated with that trajectory. The optimization problem then seeks to find a trajectory that minimizes this cost function while adhering to the given constraints, which can be formally expressed as:

$$\begin{aligned} \tau^* &= \arg\min_{\tau \in \mathcal{T}} \mathcal{F}(\tau) \\ s.t. \quad & \mathcal{C}(\tau) \leq 0 \end{aligned} \tag{2.15}$$

where $\mathcal{C}(\tau)$ represents the various constraints mentioned above.

Then various optimization methods can be employed to solve the trajectory optimization problem as defined in Equation 2.15. These methods are generally categorized into two main groups: gradient-based methods such as Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [91] and iterative Linear Quadratic Regulator (iLQR) [92], and gradient-free methods such as Stochastic Trajectory Optimization for Motion Planning (STOMP) [93] and CEM [94].

Gradient-based methods utilize the gradient of the cost function with respect to the trajectory to iteratively improve the solution. They are effective when the cost function and constraints are differentiable.

CHOMP [91] is an iterative algorithm that refines an initial trajectory using functional gradient techniques. It improves the trajectory by locally optimizing a cost function that penalizes deviations from the desired path and proximity to obstacles. Through a simple variational strategy, CHOMP can iteratively update the trajectory while balancing smoothness with obstacle avoidance. This makes CHOMP versatile for various motion planning scenarios, especially in high-dimensional problems. However, the gradient-based nature of CHOMP introduces a limitation: if the initial trajectory is poorly chosen, the algorithm may converge to a local minimum rather than the global optimum. In such cases, CHOMP may either fail to find a viable solution or settle for a sub-optimal trajectory, leading to potential planning failures.

iLQR [92] solves the complex motion planning problem by iteratively linearizing the system around a nominal trajectory and applying a modified Linear Quadratic Regulator(LQR) method. Its workflow involves initializing a nominal trajectory, linearizing the dynamics, computing optimal control inputs, updating the trajectory, and repeating the process until convergence. Although iLQR is known for its speed and efficiency, it can be sensitive to initial conditions, may converge to local minima, and relies on the assumption of smooth system dynamics, which may not always be valid.

Gradient-free methods do not require the computation of gradients and are suitable for optimization problems where the cost function or constraints are non-differentiable or the gradients are difficult to obtain.

STOMP [93] utilizes a stochastic approach to iteratively refine trajectories by sampling and perturbing candidate solutions and evaluating their performance. In each iteration, STOMP generates a set of random trajectories, each perturbed by noise around the current best trajectory, which is not necessarily feasible at first. These perturbed trajectories are evaluated by rolling out the dynamics model to determine their cost. The trajectory with the lowest cost is then perturbed again to explore further, and the process continues until a satisfactory trajectory is found. In the whole process, this method does not rely on any explicit gradient information. Instead, it explores the solution space through randomness, which helps to avoid local minima and potentially yields better optimization results. Although the cost function used in STOMP is similar to that in CHOMP, the stochastic nature of STOMP allows it to more effectively navigate complex constraint spaces and achieve better solutions by overcoming these limitations.

CEM [94] is similar to STOMP, which iteratively refines a probability distribution to identify optimal solutions by sampling and evaluating candidate solutions. The main process of CEM involves generating candidate trajectories, assessing their performance, selecting the best candidates, and updating the distribution accordingly. This iterative refinement allows CEM to effectively converge towards the global minimum, making it highly flexible for various motion planning problems.

In this thesis, due to the fact that the manipulation problem involves discrete constraints, I use CEM as a representative optimization-based planner in the proposed framework.

2.5.4 Position-based Control and Impedance Control

After obtaining a plan from the motion planner, executing the plan requires controllers to ensure the robot follows the planned path accordingly. In this thesis, two main types of controllers are utilized: the position-based controller and the impedance controller.

The difference between position-based control and impedance control for robot arms lies in how each method handles the interaction between the robot and its environment, as well as how they respond to external forces.

Position-based control [95] focuses on accurately following a predefined trajectory or reaching a specific position in space. It adjusts the joint angles or end-effector position of the robot arm to match the desired setpoints, usually using feedback loops to correct errors. This approach assumes that the environment is mostly static or that the robot can move without significant interaction forces. The robot arm will try to reach the desired position regardless of external disturbances, which can sometimes result in stiff or rigid behavior. Position-based control is commonly used in tasks requiring high precision where environmental interactions are minimal, such as assembly tasks in controlled environments. I use a position-based controller for controlling the gripper as it approaches the deformable objects before making contact. This method is chosen because, in this phase, there is no physical interaction with the object; the main focus is on accurately following a predefined path and avoiding obstacles.

Impedance control [96] aims to regulate the dynamic relationship between forces and movements, creating a compliant response to external forces. Unlike position-based control, which focuses solely on position, impedance control models the robot-environment interaction as a mechanical impedance, managing forces, velocities, and displacements simultaneously. The controller adjusts the motion of the robot based on sensed forces, allowing it to respond naturally and compliantly to disturbances. The robot behaves like a spring-damper system, which can "give way" under force or maintain a soft interaction, making it ideal for contact tasks such as pushing, pulling, or handling delicate objects. In this thesis, I use the impedance controller to manage the interaction of the gripper with the deformable object during the grasping and releasing phases. This ensures gentle and adaptive handling of the object, accommodating both the deformability of the object and interactions with the environment.

2.6 Motion Planning for Deformable Object Manipulation

Plenty of work has been conducted on deformable object manipulation using classical planning methods. These approaches often rely on fine-tuned simulators or simplified local approximations, primarily due to the absence of universal and accurate analytical models for non-rigid objects.

Moll et al. [97] present a technique for planning paths of flexible wires that minimizes energy while satisfying endpoint constraints. They introduce a novel parameterization to represent low-energy configurations and propose three methods for minimizing energy within the self-motion manifold of the curves. They also develop a local planner to facilitate smooth minimal energy deformations, which can be integrated into general path planning algorithms.

Ramirez-Alpizar et al. [98] introduce an energy-based function for assembling a ring-shaped elastic component onto a cylindrical part using a robot with dual-arm manipulators. They integrate both an energy function and an obstacle avoidance function into the optimization goal and employ the CHOMP method to solve the trajectory optimization problem. Experiments with a PR2 robot show that their proposed motion planner is able to find a feasible path while keeping the deformation of the elastic object as small as possible.

In [99], Li et al. propose an offline trajectory optimization method for deformable cloth based on predictive simulators. To prevent generating wrinkles, gaps, or completely failing the task, they introduce a novel quadratic objective function that measures the discrepancy between the predicted and predefined goal states. The trajectory is parameterized using a Bézier curve, and optimization is performed with the Levenberg-Marquardt algorithm. With the help of a key point detection system, they demonstrate that the proposed method can successfully achieve accurate manipulations of deformable shirts.

Hamajima et al. [100] present a planning strategy for robots to effectively handle and unfold clothes, focusing on the rehandling subtasks necessary for this process. They outline three key rehandling subtasks: grasping the hemline, addressing unfound hemlines, and managing shadow regions that may obscure detection. They detail the algorithms used for detecting hemlines and the subsequent unfolding tasks, emphasizing the importance of grasping points and the classification of clothing states.

McConachie et al. [101] present a novel deformable object manipulation method by interleaving prediction, planning, and control without relying on high-fidelity modeling or simulation. They combine the advantages of planning and control while mitigating their drawbacks: planning generates a coarse global path to an intermediate point where the controller can finish the task locally without getting stuck in local minima. A new deadlock prediction algorithm, based on a virtual elastic band (VEB) propagation model, determines when to switch between planning and control, and this model also supports global path planning. The local controller builds on prior work with Jacobian-based deformable models [102, 103]. Extensive simulations demonstrate that the proposed framework can successfully manipulate deformable ropes or cloth through narrow passages or avoid obstacles without being stretched, and the computation time is much lower than using high-fidelity simulators.

Although planning methods are capable of finding global trajectories, they usually rely on simulators with manually defined grasping positions and finely tuned models. The former does not generalize well to tasks involving various deformable objects, while the latter is time-consuming due to the complexity and the large number of rollouts required during the search process. This makes planning-based methods generally unsuitable for robots performing time-critical deformable manipulation tasks, especially in unstructured environments with complex and unknown deformable object properties. Therefore, some researchers propose decreasing the dimensionality of the state representation, reducing the potential action space, and using approximate or simplified dynamics models, as detailed in the following sections.

2.7 Reduced Representation & Action for Deformable Object Manipulation

The idea of identifying important points or features on deformable objects has been studied before. One way of approaching this problem is to determine particular features for a specific task. For example, Qiu et al. [32] and Sun et al. [104] propose to detect and eliminate wrinkles, where the task is to flatten a deformable object. Other approaches include using manually input key points [105], or contours [106]. Recently, learning-based methods for deformable object manipulation have been widely used [107, 108, 109, 110, 111, 112]. Some of these approaches aim to simplify deformable object representations and identify key features on them. For example, Yan et al. [17] propose extracting a compact representation of the deformable object directly from raw sensor inputs for dynamics learning to facilitate faster planning. Lips et al. [108] learn key points from RGB images directly for specific cloth categories using synthetic data and manipulate deformable objects with scripted motion primitives. Zhou et al. [109] introduce a latent representation for soft object manipulation with semantic correlations, while Arnold et al. [110] estimate mesh representations from voxel inputs, and perform planning in mesh format internally. Ma et al. [111] approximate a deformable object as a sparse set of interacting key points and learn a graph neural network that captures the geometry abstractly. Additionally, [112] proposes learning 3D features using a combined PointNet encoder and neural radiance field (NeRF) for various deformable objects.

In this thesis, I also exploit the idea of identifying important features, and use it to reduce the action space. However, the approach proposed in this thesis differs from the above in that our method 1) can adapt to different tasks (as opposed to identifying features for one specific task); 2) identifies the key particles autonomously; and 3) takes a model-based (as opposed to learning-based) *and* goal-conditioned approach.

2.8 Dynamics Model Approximation for Deformable Object Manipulation

The idea of using coarse/approximate dynamics models in deformable object manipulation has attracted wide attention from researchers due to the fact that high-fidelity models are scarce, difficult to tune, and expensive to simulate [113]. Thus various simple models are proposed for specific tasks [102, 114, 115, 116]. Ruan et al. [102] develop a directional diminishing rigidity model for rope and cloth dragging without relying on simulating expensive mass-spring models. McConachie et al. [114] employ a virtual elastic band model to approximate true dynamics and train a classifier to decide when to trust the simplified model for rope manipulation in clutter. Additionally, McConachie et al. [115] introduces a multi-armed bandit method to adaptively select from multiple simplified models, avoiding the need for a high-fidelity deformable model in applications like rope winding and table covering. Power et al. [116] utilize simple models for cost-efficient data collection, e.g. using a pendulum model to approximate a tethered rope, thereby enhancing learning efficiency. Zhou et al. [117] focus on identifying, modeling, and manipulating structures of interest rather than the entire object for bimanual bag manipulation, which substantially reduces computational load compared to full dynamics modeling. Learning-based methods have also been explored to train a neural network as the underlying dynamics model for motion planning [107, 118, 112]. For instance, [118] demonstrate a visual dynamics model trained on domain-randomized RGBD images for fabric folding tasks, while Li et al. [112] use a recurrent state-space model (RSSM) for latent dynamics modeling. Other approaches involve using graph neural networks to learn deformable dynamics based on key points or mesh representations [111, 119, 12]. Mitrano et al. [120] propose adapting a learned dynamics model to novel domains by focusing on the regions where source and dynamics are similar. Lee et al. [121] propose simulating cloth more efficiently using a miniature model with similar physical properties and an upscaling deep neural network. However, only limited results are presented for hanged cloth. Interests in model reduction extend to related areas in robotics, such as the framework from [122] for optimizing reduced order models and trajectories for bipedal locomotion, and the application of model order reduction techniques to solve optimal control problems for cable-driven soft robots, significantly improving computational efficiency [123].

Instead of manually constructing simplified models or training neural network models offline for specific tasks, I propose to simplify the dynamics model based on various goals and tasks provided, allowing it to operate online without the need for offline training. Furthermore, the method proposed in this thesis can progressively refine the simplified model in a consistent way. These enable the method in this thesis to accept arbitrary goals of the object and to quickly perform model simplification conditioned on this goal, without requiring goal-specific training.

2.9 Summary

This thesis focuses on deformable object manipulation from both a model-based and motion planning perspective. Thus, a bunch of work in deformable object modeling, simulation, perception, planning, and control [4] are reviewed in this chapter. From this review, it becomes evident that simplifying the problem is crucial for efficient motion planning. Consequently, this thesis explores the ideas of reducing the action space and simplifying the dynamics model to address the challenges associated with deformable object manipulation.

Firstly, this thesis is related to existing works on reducing the state representation or extracting key features of the deformable object [32]. Unlike other works, I solve it in a configuration-informed way, taking into account various goals that differ from task to task. This allows for simplifying the representation by focusing on the most relevant elements for each specific goal. Besides, I use this simplified representation to reduce the action space, with an aim to help motion planners search for more effective plans with better efficiency. This reduction in action space enhances the planner to find optimal solutions, making the motion planning process more efficient.

Many prior works find that using approximate models or simplified dynamics [107] is a practical approach when dealing with the complexities of deformable object manipulation, due to the high-dimensional state and intricate dynamics involved. In this thesis, I adopt a similar strategy but with a distinct approach: instead of uniformly simplifying the dynamics model or tailoring it to a specific task, I address the problem in a goal-conditioned manner. This method allows for automatic simplification of the dynamics model based on the specific goals of various tasks, eliminating the need for task-specific training.

There are also approaches that utilize multiple models [115] for a single task, where simplified models are pre-built, but the relationships between these models are often not fully leveraged. In contrast, this thesis adopts a configuration-informed approach and simplifies the dynamics model sequentially. This method ensures that each simplified model aligns with the previous ones while progressively incorporating more details. This consistent and gradual refinement of the model allows for a more coherent representation of the dynamics, enhancing the overall effectiveness of the planning process.

Chapter 3

Goal-Conditioned Action Space Reduction for Deformable Object Manipulation

3.1 Introduction

Consider a scenario where a robot is required to manipulate a piece of cloth into a specific goal configuration, like the object folded diagonally in Figure 3.1. A general way to achieve this goal is building a model of the object in simulation, and optimizing a trajectory within a planning framework. Deformable objects are usually modeled as mass spring system, with each mass point (which is referred to as a *particle* in this thesis) also representing a possible picking point for the robot. High-fidelity models of deformable objects often contain thousands of such particles, making planning extremely computationally expensive.

To alleviate this computational burden associated with high-dimensional models of deformable objects, one possible way is to confine the pickable points on the object, which consequently results in a truncated search space where the motion planner needs to explore. For example, controlling the shape of a rope by manipulating only its two ends [47], folding or unfolding a piece of cloth by focusing on its visible corners [124, 125, 105], or flattening fabric by removing wrinkles [32]. These methods are shown to perform well, however, they are effective only for specific tasks and do not generalize well to a wide range of goals.

To address this limitation, in this chapter, I propose a more flexible approach by simplifying the geometric model and reducing the action space in a goal-conditioned manner. This approach aims to generalize the action space reduction for various tasks, providing a more versatile solution for deformable object manipulation. This is done by identifying a small number of key particles sufficient as picking points to reach the goal state. These key particles are determined

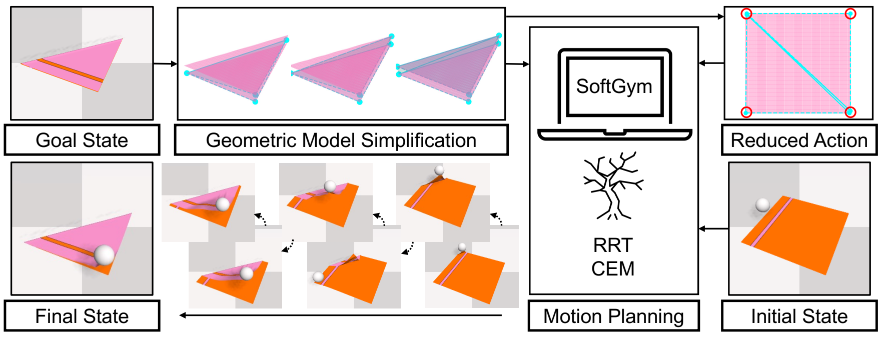


Figure 3.1: Overview of the proposed action space reduction for deformable object manipulation planning. Given the task of folding a piece of cloth diagonally, our approach simplifies the geometric model at the goal configuration to extract the key particles as denoted by red circles, which represent the pickable points in the reduced action space. After that, a motion planner (CEM or RRT) is invoked to search for a folding plan in the SoftGym simulator.

through a geometric model simplification process, which gives a minimal geometric model that still enables a good approximation of the original model at the goal configuration. For instance, as shown in Figure 3.1, to fold a piece of cloth in half diagonally, a minimal geometric model consisting of four triangles and six particles provides a good approximation of the goal shape. This minimal model is then used to identify the corresponding key particles on the original, high-dimensional model. These key particles serve as the picking points in the reduced action space, which, in this case, are the four corners of the cloth. Planning based on the resulting small set of key picking points in the action space is then significantly faster, as opposed to planning using the original action space with several thousands of picking points. This approach allows for faster and more efficient planning while maintaining effective control over the deformable object. In summary, the methods proposed in this chapter differs from other research that also explores simplifying deformable object models and identifying important features on them in three ways: our method 1) can adapt to different tasks (as opposed to identifying features for one specific task); 2) identifies the key particles autonomously; and 3) takes a model-based (as opposed to learning-based) *and* goal-conditioned approach. These enable the proposed method to accept arbitrary goal configurations of the object and to quickly perform simplification of the geometric model and reduction of the action space conditioned on this goal, without requiring goal-specific training.

To validate the proposed action space reduction approach for deformable object manipulation, a set of experiments are conducted on some representative tasks involving ropes and cloths, both in simulation and in the real-world. First, geometric model simplification methods, including line fitting for 1-D ropes and mesh simplification for 2-D cloth, are tested to demonstrate their effectiveness in approximating the original geometric models. Next, the performance of motion planning using the reduced action space with autonomously extracted key picking points is compared against those using the original action space and other baseline action spaces. At last, real-world experiments of a Franka Panda robot performing cloth diagonal folding and side folding tasks are conducted, to demonstrate the applicability of the planned trajectory in a practical setting.

In particular, key contributions of this chapter include:

- A general action space reduction scheme integrated into a manipulation planning pipeline based on goal-conditioned geometric model simplification for deformable objects.
- An implementation of this general approach for 1-D linear deformable objects (e.g., ropes) that uses a piece-wise line fitted model, and for 2-D surface deformable objects (e.g., cloth) that uses a mesh simplified model.
- An extension of this approach to multi-step planning with intermediate goal configurations, where geometric model simplification and action space reduction are performed on successive intermediate goals.

The rest of this chapter is organized as follows: First in Section 3.2, I provide a general definition of the motion planning problem for deformable object manipulation, which establishes the foundation for this chapter and subsequent chapters in the thesis. In Section 3.3, I present the method for action space reduction, beginning with the general formulation of geometric model simplification, and then detailing 1-D line fitting and 2-D mesh simplification. Then I explain how these techniques are used to reduce the action space for specific tasks and how this reduced action space is incorporated into a planning framework, including extensions to handle long-horizon tasks with multiple intermediate goals. In Section 3.4, I describe a set of representative tasks used to validate our methods, present the results of geometric model simplification, and compare the action space reduction scheme for deformable object manipulation planning with various baseline approaches. In Section 3.5, I demonstrate the effectiveness of the planned trajectory with a Franka Panda robot performing cloth folding tasks in the real world.

3.2 Problem Formulation of Motion Planning for Deformable Object Manipulation

The main problem that I consider in this chapter as well as the rest chapters in this thesis is the motion planning for manipulating a deformable object into a given goal. While manipulation happens in 3-D space, I focus on objects that can be represented by 1-D lines (like ropes) or 2-D surfaces (like cloths). For example, straightening a crumpled rope (Figure 3.6(a)) or folding a piece of cloth diagonally into half (Figure 3.6(c)).

In this section, I define the state space, action space, state transition function, and objective function of the manipulation problem. I distinguish between the geometric model of a deformable object and its state. The geometric model represents the connections between the particles in the mass spring model and is topological, therefore is not affected by manipulation.

The state of the model, on the other hand, represents the position of all particles, and is affected by manipulation actions.

For both 1-D linear and 2-D surface deformable objects, the geometric model can be represented as an undirected and weighted graph $\mathcal{G} = (V, E, L)$, where $V = \{1, 2, ..., N\}$ denotes a set of particles indexed from 1 to $N = |V|$, $E \subseteq \{\langle i, j\rangle \mid i, j \in V \, and \, i \neq j\}$ denotes the edges, and $L \in \mathbb{R}^{|E|}$ represents the edge weights, corresponding to the resting length of the edges.

The state of the deformable object at each time step during manipulation can be represented using the positions of all particles, denoted as $\xi^t := \{p_i \mid \forall i \in V\}$, where $p_i = (x_i, y_i, z_i)$ represents the position of the i^{th} particle.

As for action space $\mathcal{A}$, I assume the robot is equipped with one picker, which can pick any given particle in the object, that is, any $i \in V$, and move it by a certain distance along a direction in 3D space. I also add a $None$ action, corresponding to not holding any picking point. The action at time t can be represented as:

$$a^t = \begin{cases} \langle i, \delta x, \delta y, \delta z\rangle \\ None \end{cases} \tag{3.1}$$

It is worth noting that the action space can be easily extended to multiple grippers.

Moreover, I constrain the movement of the picker at each step to avoid moving the deformable object drastically: $|\delta x| \leq \Delta x$, $|\delta y| \leq \Delta y$, $|\delta z| \leq \Delta z$; Δx, Δy and Δz are motion limits along each axis in Cartesian space. This ensures that the object's motion remains quasi-static, minimizing inertial effects and preserving stability.

The dynamics model of deformable objects considered in this thesis is a mass spring system, which determines how the object moves and deforms given the actions. I define it as $\mathcal{D} = (\mathcal{G}, \mathcal{M}, \mathcal{R})$, of which $\mathcal{G}$ denotes the geometric model as described above, $\mathcal{M}$ denotes the dynamics properties of all particles such as mass and radius, and $\mathcal{R}$ denotes various dynamics relations between different particles such as the springs and constraints of collision. For the implementation of the dynamics model, I use the one provided by SoftGym [54], which is built on top of Flex from Nvidia [126, 127]. Within the simulator, original dynamics models of ropes and cloths are constructed by a sequence of particles or a grid of particles respectively. Neighboring particles are connected by stretching-constrained springs, one step away particles are constrained by bending springs, and shearing springs are added for diagonally connected particles. Meanwhile, self-collision of all particles is also considered, details of which can be found in [126].

Based on the dynamics model $\mathcal{D}$, a state transition function can be defined accordingly:

$$\xi^{t+1} = f_{\mathcal{D}}(\xi^t, a^t). \tag{3.2}$$

which outputs the new state of the object given the current state ξ^t and action a^t.

I formulate the manipulation problem as a finite-horizon motion planning problem, whose solution is a trajectory, i.e., a sequence of T actions, $\tau = \langle a^0, a^1, ..., a^{T-1} \rangle$, that minimizes the distance between the final state of the particles ξ^T after executing τ and their goal state ξ^G:

$$\begin{aligned} \min_{\tau} \quad & \|\xi^G - \xi^T\| \\ s.t. \quad & \xi^{t+1} = f_{\mathcal{D}}(\xi^t, a^t),\ \forall t \in [0, T-1]\,, \end{aligned} \tag{3.3}$$

where I assume ξ^0 is given as the initial state, and the state transition function $f_{\mathcal{D}}$ guarantees the feasibility of states along the trajectory. Other model constraints are considered by the SoftGym simulator, such as stretching and bending limits, environmental constraints like working space restrictions, and robot kinematic constraints.

In this planning framework, the size of the action space is $|\mathcal{A}| = (N \times \mathbb{R}^3 + 1)$. The number of graspable particles affects linearly the size of the action space, which has, in turn, an exponential effect on the search space through the branching factor. Therefore, the number of picking points considered for planning has a significant impact on planning efficiency. Furthermore, the complexity of the dynamics model used in Equation 3.3 also affects the computational greatly due to the massive amounts of trajectory rollouts required by motion planners.

In the next section, I present a methodology to solve this problem by simplifying the geometric model and reducing the action space.

3.3 Goal-Conditioned Action Space Reduction

As defined in Equation 3.1, the action space consists of a picking index and a 3-D vector in Cartesian space, which is infinitely large for a deformable object. For example, a gripper can pick anywhere on a piece of cloth for manipulation. However, not all parts of the cloth are equally important for achieving a specific goal, and picking some parts may even hinder progress. While one can manually confine the pickable positions on the deformable object, such as the corners, this approach may work for some tasks but not others. Instead, I propose to extract specific particles from the deformable object as the potential picking points in the action space by simplifying the original geometric model $\mathcal{G}_O$ to $\hat{\mathcal{G}}_S = (\hat{V}_S, \hat{E}_S, \hat{L}_S)$ at the goal configuration. In this way, the particles that are most relevant to achieving the goal are automatically selected as picking points in the action space, the number of which is $\hat{N}_S = |\hat{V}_S|$ and thus much smaller than the original one. The process of getting such reduced action space is described in this section, based on geometric model simplification methods, which differ for objects that can be approximated with 1-D models and objects that can be approximated with 2-D models.

In Section 3.3.1, I summarize the overall approach. In Section 3.3.2, I introduce the function to simplify the geometric model. In Section 3.3.3, I focus on 1-D models, and in section Section 3.3.4, on 2-D models. Lastly, in Section 3.3.5, I describe how this planner can be applied to multi-step planning with intermediate goal configurations.

3.3.1 Action Space Reduction

The overall algorithm for action space reduction is illustrated in Algorithm 1 which defines the `Reduce_Action_Space` function to extract specific particles from the deformable object as the potential picking points in the action space based on the simplified geometric model at the goal configuration. The inputs of the method are the original geometric model $\mathcal{G}_O$ and goal configuration of the original model ξ_G^O, while the output is the extracted key particle set $V_O^{\mathcal{A}} \subseteq V_O$ in the action space, which is related yet different from the set of particles $\hat{V}_S$ of the simplified model.

To extract these key particles, I first simplify the original geometric model at the goal configuration, and get a simplified geometric model $\hat{\mathcal{G}}_S$ as well as the corresponding state $\hat{\xi}_S$ (Line 1) using the `Simplify_Geometry` function defined in Section 3.3.2. Then I calculate the distances between the vertices of both models to identify the key particles on the original model that correspond to the particles of the simplified geometric model $\hat{V}_S$ (Line 2):

$$\hat{V}_O \leftarrow \bigcup_{\hat{v}_s \in \hat{V}_S} \operatorname*{arg\,min}_{v_o \in V_O} \|\hat{\xi}_S(\hat{v}_s) - \xi_O(v_o)\| \tag{3.4}$$

where $\hat{V}_S$ denotes the vertices of the simplified geometric model $\hat{\mathcal{G}}_S(\hat{V}_S, \hat{E}_S, \hat{L}_S)$; V_O denotes the vertices of the original geometric model $\mathcal{G}_O(V_O, E_O, L_O)$; $\hat{\xi}_S$ denotes the state of the simplified geometric model, and $\hat{\xi}_S(\hat{v}_s)$ denotes the position of vertex $\hat{v}_s$; ξ_O denotes the state of the original geometric model, and $\xi_O(v_o)$ denotes the position of vertex v_o.

The set of particles $\hat{V}_O$ is then filtered based on their geodesic distances, in which geodesically close particles are combined to produce a final set of particles $V_O^{\mathcal{A}}$ (Line 3).

I then use $V_O^{\mathcal{A}}$ as the picking points in the reduced action space $\mathcal{A}_S$, which are subsequently used for motion planning. Compared to using all particles in the original model as picking points in the action space, the search space is significantly reduced by confining the picking points to the reduced key particles, where $|V_O^{\mathcal{A}}| \ll |V_O|$.

An example of this process is illustrated in Figure 3.2, where the original geometric model at the goal state is shown in Figure 3.2(a), while a simplified model with two triangles, obtained using Algorithm 2, is depicted in Figure 3.2(b). Four particles on the original geometric model are then identified as key picking points which are marked by red circles in Figure 3.2(c).

Algorithm 1: Action Space Reduction (`Reduce_Action_Space`)

```
    Input: $\mathcal{G}_O$, $\xi_O$
    Output: $V_O^{\mathcal{A}}$
1   $\hat{\mathcal{G}}_S, \hat{\xi}_S \leftarrow$ Simplify_Geometry($\mathcal{G}_O, \xi_O$);
2   $\hat{V}_O \leftarrow$ Extract($\mathcal{G}_O, \xi_O, \hat{\mathcal{G}}_S, \hat{\xi}_S$);
3   $V_O^{\mathcal{A}} \leftarrow$ Filter($\hat{V}_O$)
```

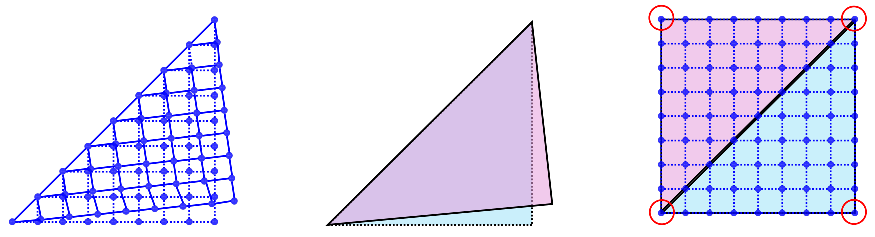

(a) Original geometric model (b) Simplified geometric model (c) Reduced action space

Figure 3.2: Geometric model simplification and action space reduction for cloth diagonal folding. The original geometric model at the folded state is shown in (a); I simplify it using the proposed Algorithm 2 and get a simplified geometric model in (b); by extracting the nearest particle on the original model, four particles on the original model are identified as picking points in the reduced action space, as shown in (c).

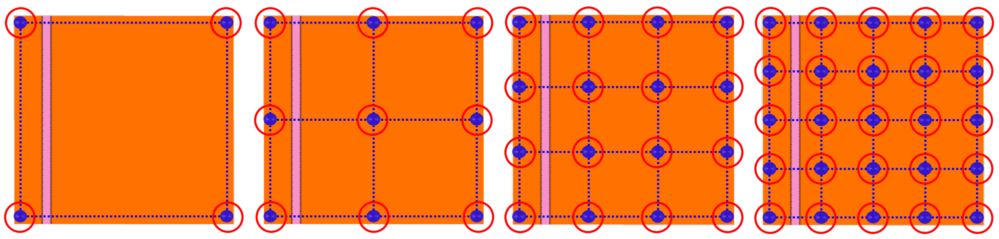

(a) Grid (2×2) (b) Grid (3×3) (c) Grid (4×4) (d) Grid (5×5)

Figure 3.3: Grid-based action space reduction. A grid of varying resolutions is overlaid onto the underlying cloth model, which is composed of thousands of particles. The particles intersected by the grid on the original model (blue spheres) are extracted as potential picking points, marked by red circles.

3.3.2 Geometric Model Simplification

The aim of simplifying the geometric model is to create an approximation of the original model while maintaining simplicity. A grid model of a certain resolution can only approximate certain shapes; for example, a grid geometry of resolution 2×2 cannot accurately represent the state of a side-folded cloth, as shown in Figure 3.3(a). I adopt a goal-conditioned approach to simplify the geometric model so that it aligns well with the original model at the goal state.

The basic process of the configuration informed geometric model simplification method, `Simplify_Geometry`, is illustrated in Algorithm 2, where the inputs include the original geometric model $\mathcal{G}_O$ and the given state of the original model ξ_O, which can either be the goal state or the actually achieved final state, and the outputs are the simplified geometric model $\hat{\mathcal{G}}_S$ and the corresponding state of the simplified geometric model $\hat{\xi}_S$. Firstly, the algorithm tries to simplify the original geometric model to its simplest form, in which the number of target simplification elements is 2 (Line 1). Based on the original geometric model, the given configuration of the model, and the target number of the model elements, the simplification function (`Simplify`) is invoked to reduce the original geometric model $\mathcal{G}_O$ to $\hat{\mathcal{G}}_S$, with corresponding simplified goal configuration $\hat{\xi}^S$ (Line 3). The distance between the given state of the original geometric model and the simplified model is calculated as the *error*, indicating how good a fit the simplified geometric model is to the original model, at the given state (Line 4). Lastly, if the *error* falls below a user-defined threshold or if the approximation fails to improve for a certain number of iterations—indicating convergence—the simplification process terminates, the simplification process is terminated (Line 6); otherwise, the process repeats with a more complex simplified geometric model (Line 5) by incrementing N_S. Overall, the `Simplify_Geometry` method adaptively finds a value of N_S that is sufficient to represent the given state ξ_O.

The method I use to simplify the geometric model differs for objects that can be approximated with 1-D models and objects that can be approximated with 2-D models, as detailed below.

3.3.3 Piece-Wise Line Fitting for 1-D Linear Models

For 1-D linear models, I adopt a piece-wise line fitting method to implement the `Simplify` function (Algorithm 2, Line 3). Given the number of key particles N_S of the target simplification, a Quadratic Programming (QP) problem is defined and solved to find the optimal position of the N_S particles. The cost function of the QP problem to be minimized is the "distance" between the fitted piece-wise lines and the original shape of the object. To find such a distance value, I sample a large number of, N_E, points on the two models. For example in Figure 3.4, the pink dots show the N_E points sampled on the original model, while the blue dots show the N_E points sampled on the simplified model. I find the mean distance between corresponding particles (grey dot-dash line in Figure 3.4):

$$distance(\xi_O, \hat{\xi}_S) = \sum_{i=1}^{N_E} \frac{\|q_O^i - q_S^i\|}{N_E} \tag{3.5}$$

where q_O^i represents the position of the i^{th} sampled point on the original model, and q_S^i represents the position of the i^{th} point sampled on the simplified model.

After QP minimization is complete, I use the same distance formulation above (Equation 3.5) to implement the `Error` function in Algorithm 2 (Line 4) to compute the final distance between the two models.

An example piece-wise line fitting process for a rope of random shape is shown in Figure 3.4. The first picture shows a simplified model with two particles and one line segment; the fit to the original model is quite poor. As the number of particles in the simplified model, N_S, is increased, we get better fits. Depending on the threshold set (Algorithm 2, Line 6), the simplification process can terminate, for example, at the ten-particle model ($N_S = 10$) in Figure 3.4.

Algorithm 2: Geometric Model Simplification (`Simplify_Geometry`)

```
    Input: $\mathcal{G}_O$, $\xi_O$, threshold
    Output: $\hat{\mathcal{G}}_S$, $\hat{\xi}_S$
1 $N_S = 2$;
2 do
3     $\hat{\mathcal{G}}_S, \hat{\xi}_S \leftarrow$ Simplify($\mathcal{G}_O, \xi_O, N_S$);
4     $error \leftarrow$ Error($\hat{\xi}_S$, $\xi_O$);
5     $N_S \leftarrow N_S + 1$;
6 while $error > threshold$ and not convergent;
```

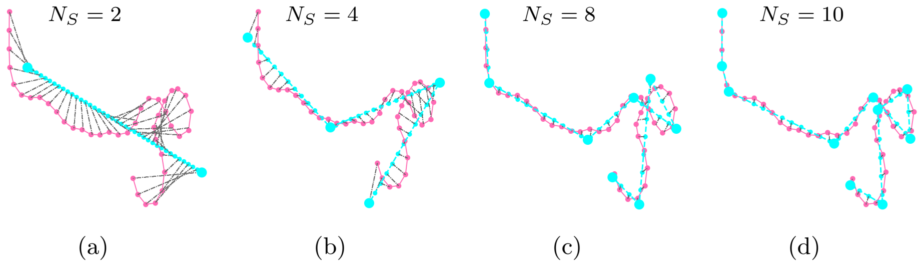


Figure 3.4: Piece-wise line fitted simple model (blue) for a given goal state of the original model (pink), for different N_S values. Blue dots represent sampled points on the simplified model, and grey dashed lines connect the corresponding points of both models.

3.3.4 Mesh Simplification for 2-D Surface Models

For objects that can be approximated by surfaces, I use the Quadric Edge Collapse Decimation (QECD) method as the model simplification function, `Simplify`, of Algorithm 2, which can simplify the model towards a given number of elements N_S. The basic element for mesh simplification is a triangle, which is composed of particles that can be shared between different triangles. QECD is a surface simplification algorithm based on the quadric error metrics proposed by Garland and Heckbert [128]. During simplification, pairs of vertices (particles) are contracted to one iteratively, until the target number of triangles, N_S, is achieved. We use the QECD implementation in the mesh processing library Meshlab [129].

I start from simplifying the original mesh to a model with $N_S = 2$ triangles (four particles, two of which are shared), and gradually increase the number of triangles until the error is below the threshold, as in Algorithm 2.

To implement the `Error` function and find the distance between the simplified mesh and the original one (Algorithm 2, Line 4), I use the Hausdorff distance [130], which is a widely used similarity metric for mesh and image comparison. The Hausdorff distance is defined as the maximum distance of a set to the nearest point in the other set, and in our case, is found by:

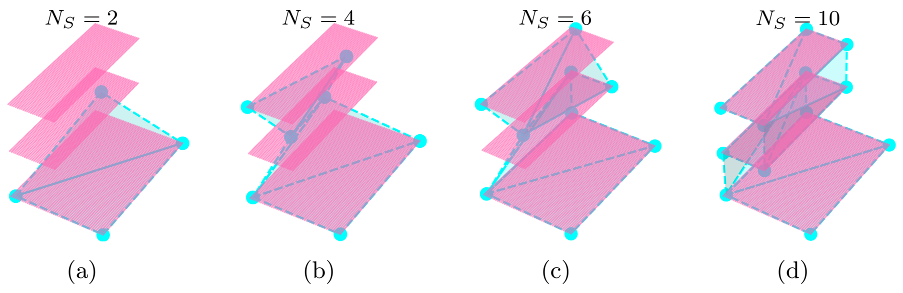


Figure 3.5: Reduced mesh of the simple model (blue) for a reflectively folded state of the original model (pink), for different N_S values. The vertical dimension in these figures are scaled up for visualization purposes, to make the three layers of the folded cloth visually separate.

$$h(\hat{\xi}_S, \xi_O) = \max_{q_S \in \hat{\xi}_S} \{ \min_{q_O \in \xi_O} \|q_S - q_O\| \} \tag{3.6}$$

As shown in Fig. 3.5, a piece of reflectively-folded cloth (i.e., when the cloth is first folded in half, and then the top half is folded a quarter back onto itself) is initially approximated by a simple mesh with two triangles, which only covers the bottom face. This is improved by adding more triangles to the simplified model. In the last picture, a simplified model with ten triangles overlaps with the original model, giving us a satisfying approximation.

3.3.5 Planning with Multiple Intermediate Goals

Some complex manipulation tasks may require picking up the object at points that cannot be identified from the final configuration only. For example, consider the task of tying a knot, where the rope should be straightened from a crumpled state before crossing the rope over itself (similar to the rope-folding goal shown in Figure 3.6(b)), and then getting one end of the rope through the resulting loop, and so on. Such plans, with intermediate goals, are often illustrated in knot or origami books. In the model-simplification-based action space reduction and motion planning framework, a sequence of goals can be processed and achieved one by one, which only requires repeating the process of extracting key particles and motion planning for each subgoal sequentially. However, in this thesis, I do not consider the problem of computing the higher-level plan and producing the intermediate goals.

3.4 Simulation Experiments

To validate the effectiveness of the proposed geometric mdoel simplification and action space reduction methods, extensive experiments are conducted in simulation on a range of representative deformable object manipulation tasks for ropes and cloths. All simulation experiments are performed in SoftGym [54] on a workstation equipped with Intel(R) Core(TM) i9-11900 CPU @2.50GHz, 32 GB of RAM, and Nvidia GeForce GTX 4090 GPU. Section 3.4.1 details the specific tasks for rope and cloth manipulation. Section 3.4.2 provides a brief introduction of the planning methods used for testing. Section 3.4.3 presents the results of the geometric model simplification, demonstrating that the proposed methods yield a reasonable approximation of the original model at the goal configuration, leading to a reduced set of graspable points. Section 3.4.4 gives both quantitative and qualitative results of the action space reduction for motion planning.

3.4.1 Tasks

Seven representative tasks are considered in this section, including two for ropes, four for square cloths and one for a single layer t-shirt. Among these tasks, Rope Straightening and Cloth Side Folding are borrowed directly from SoftGym, while the remaining tasks are created based on SoftGym.

Rope Straightening

Manipulating a rope from a randomized initial state to a straightened goal state (Figure 3.6(a)). Performance (i.e., cost) is measured by the error between the distance of the two endpoints and the original length of the rope.

Rope Folding

Manipulating a rope into a crossed triangle (Figure 3.6(b)), from an initially straightened state. Performance (i.e., cost) is measured on the bipartite matching distance between the final and goal positions of all particles.

Cloth Diagonal Folding

Folding a flattened cloth into half diagonally (Figure 3.6(c)). Performance (i.e., cost) is measured based on the distance between corresponding particles of the two triangular halves of the cloth, with an additional penalty for any drift of the bottom-side particles from the cloth's initial position, as similarly described in [54]. To compute the overall cost, we used a weight of 1.0 for the averaged distance between corresponding particles and a weight of 1.2 for the averaged drift penalty from the initial position. For all the other tasks below, we used these same weights.

Cloth Side Folding

Folding the flattened cloth on the table into half sideways (Fig. 3.6(d)). The cost function is similar to that of Cloth Diagonal Folding, comprising the distance between corresponding particles of the two rectangular halves of the cloth and a penalty for any displacement of the cloth from its initial position [54].

Cloth Reflective Folding

Reflective folding is borrowed from origami folding where two consecutive folds are in opposite directions [131]. This task follows the goal state of the previous task (i.e., the side folding goal), with the final goal being to fold a quarter of the cloth back, as shown in Fig. 3.6(e). The cost is evaluated based on the distance between corresponding particles across the three folded layers, along with a penalty for dragging the cloth away from its initial position.

Cloth Underneath Folding

Underneath folding (e.g., Figure 3.6(f)) refers to folding a quarter of the cloth under the rest of it, contrary to placing it on the top. This task builds upon the final state of the previous task (i.e., Cloth Reflective Folding) and showcases multi-step planning with intermediate goals. The cost function considers the distance between the underneath folded part and the ground, the alignment with corresponding particles on the top layer, and a penalty for any displacement from the initial pose.

Single Layer T-shirt Side Folding

This task involves manipulating a single layer t-shirt, which has an irregular shape compared to the previous tasks. The goal is to fold the left-hand half of the flattened t-shirt on top of the right-hand half (Figure 3.6(g)). The cost function follows the same structure as the Cloth Side Folding task.

In the above tasks, *Cloth Side Folding*, followed by *Cloth Reflective Folding*, followed by *Cloth Underneath Folding* can be interpreted as a sequence of sub-goals that combine into a long-horizon task. While these sub-goals are currently provided by us, the development of high-level sub-goal planners is a promising research direction.

3.4.2 Planning Algorithms

The planning problem defined in Equation 3.3, can be solved by various off the shelf trajectory optimization and motion planning methods. For the experiments in this chapter, I use CEM [94, 84] and RRT [132] as representative examples. CEM is a well established, population based optimization algorithm, which has been applied to address plenty of manipulation problems including deformable objects. To find a solution, CEM repeatedly samples a set of trajectories from a multi-variate Gaussian distribution, calculates the cost of each trajectory, identifies the elite individuals whose cost is below a given threshold, and uses these elite individuals to refit the Gaussian distribution for the next iteration. RRT, on the other hand, is a popular sampling based motion planning algorithm widely used in robotics to efficiently navigate complex spaces. It works by incrementally building a space filling tree that randomly expands towards unexplored areas. This approach is particularly effective in high dimensional spaces and challenging environments, where traditional motion planning methods may struggle.

3.4.3 Geometric Model Simplification Results

Geometric model simplification serves as the foundation for action space reduction. While the proposed method is configuration-informed and adaptable to various states, I am particularly interested in simplifying the geometric model from the goal configuration, as it captures the most relevant features for representing and achieving the desired goal.

As the parameters of our method, we set the error threshold in Alg. 2 to a small value (0.001). Moreover, if the error in Alg. 2 does not improve (i.e., converges) for a certain number of steps (5 for 1-D linear objects, and 10 for 2-D surface objects), Alg. 2 also stops. These values can also be observed in the second column of Fig. 3.6

The results of geometric model simplification given the goal configurations of different tasks are shown in Figure 3.6. For the Rope Straightening task, as expected, only the two ends are enough to represent and reach the goal, as shown in Figure 3.6(a). For the Rope Folding task, as depicted in Figure 3.6(b), a simplified model of four particles and three line segments is acquired, identifying four corners on the original model as key particles.

As for cloth manipulation tasks, the proposed method finds much simpler geometric models than the original model for each task, which are illustrated in Figure 3.6(c-f). In Figure 3.6(c), the Hausdorff distance curve flattens at four triangles, which provides a fine approximation of the original mesh model for the diagonal folding task. Corresponding key particles are extracted and marked with red circles in the figure. Similarly, for the rest of the cloth folding tasks, a set of simplified geometric models with six, ten, and ten triangles are acquired to approximate the original models respectively.

For the t-shirt which has a more complex shape than cloth, the proposed method finds a simplified model consisting of eighteen triangles, which fits well to the original geometric model both at the goal state and the flat state.

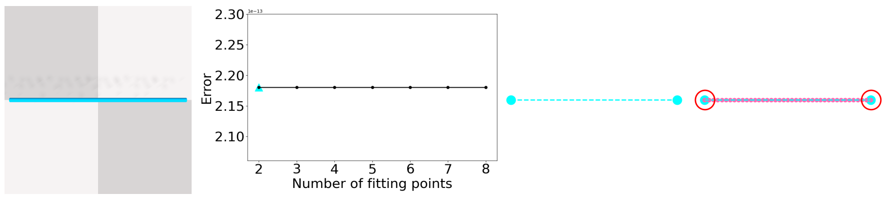
The average geometric model simplification time costs for line-fitting and mesh simplification are 0.125 s and 0.730 s respectively, which are negligible compared to the time required for motion planning.

3.4.4 Action Space Reduction for Motion Planning

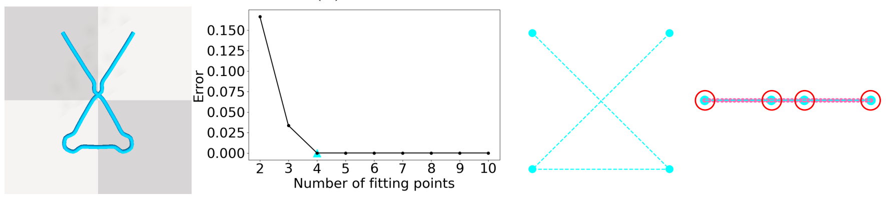
To demonstrate the effectiveness of the proposed action space reduction method and assess its impact on the time cost of motion planning, I conduct experiments comparing the performance of various baseline action spaces. These experiments use the original dynamics model across all tasks.

Implementation & Baselines

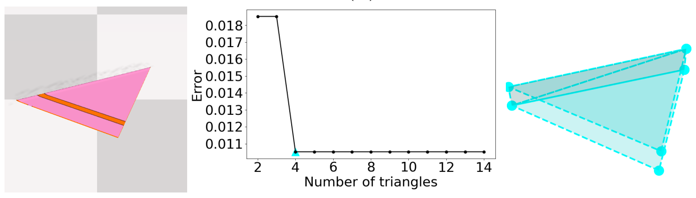
For each planner, I compare the proposed goal-informed action space with various baseline action spaces, as listed below.



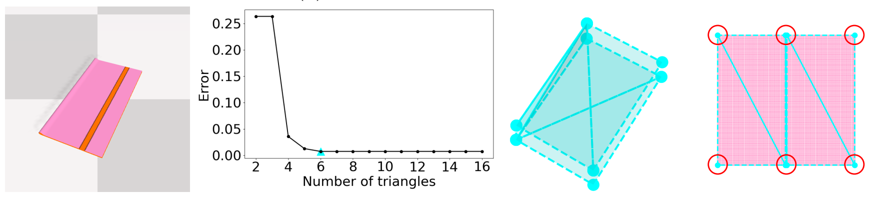
(a) Rope straightening



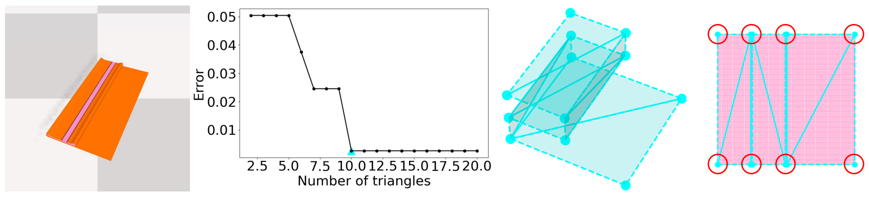
(b) Rope folding



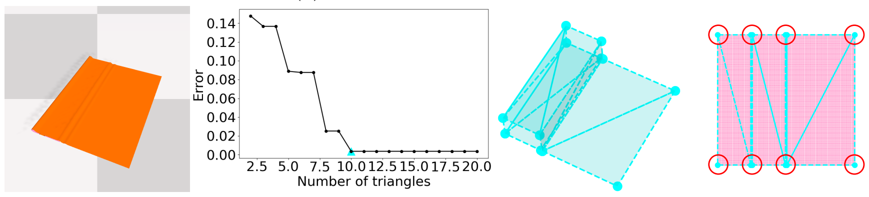
(c) Cloth diagonal folding



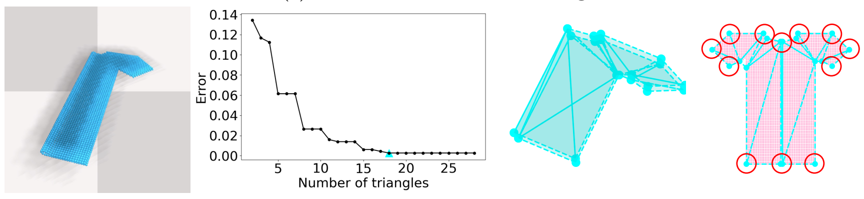
(d) Cloth side folding



(e) Cloth reflective folding



(f) Cloth underneath folding



(g) T-shirt side folding

Figure 3.6: Geometric model simplification. The figure displays the goal images in SoftGym (left), the approximation error curve (middle left), the simplified geometric model at the goal state (middle right), and the simplified geometric model at the flattened state (right).

- **Reduced Action Space**: it refers to picking the key particles, which are extracted by simplifying the geometric model at the goal state. For each task, the key particles are marked by red circles in the last column of Figure 3.6.
- **Original Action Space**: this is the baseline where the gripper can pick any particle on the object.
- **Random Action Space**: in this baseline, I uniformly randomly sample the same number of picking points as the reduced action space, using it as a random comparison.
- **Grid Action Space**: instead of choosing picking points randomly, a grid of different resolutions is overlaid on the original geometric model and the particles nearest to the intersections are selected as picking points. Three grid baselines are considered in this section, including Grid Action Space (2×2), Grid Action Space (3×3), and Grid Action Space (4×4). Example picking points of grid action space are shown in Figure 3.3 which are marked by red circles.

Furthermore, to demonstrate the effectiveness of the planned trajectory, I manually script a manipulation strategy based on the *Reduced Action Space* and denote it as **Scripted Policy**, in which the gripper picks each key particle, moves above its target position, and releases it.

Results

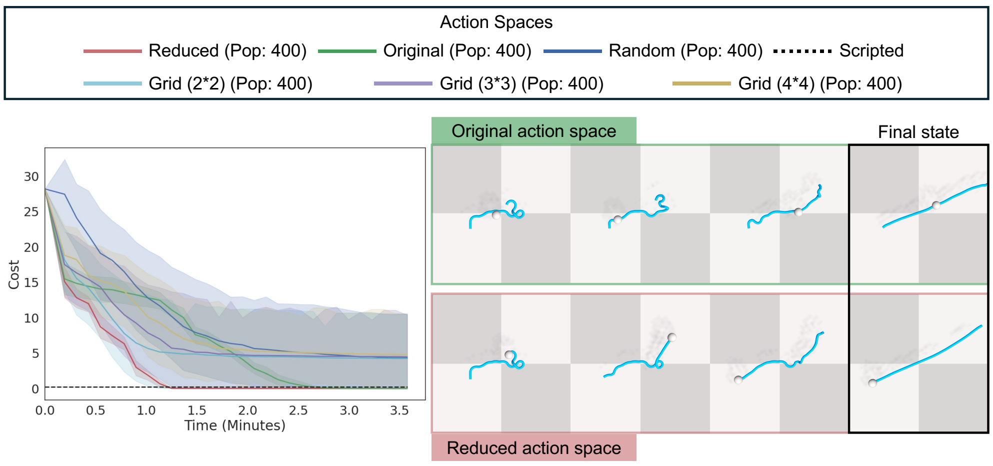
Since CEM and RRT are stochastic methods, the experiments are repeated ten times for each planner on each task.

Specifically, I run CEM for 30 iterations, with a population size of 400 for all experiments. The planning costs versus time for CEM with different action spaces are shown in Figure 3.7, Figure 3.8, and Figure 3.9. In the plots, the lines show the mean cost, while the line shadows show the 95% confidence interval over the ten runs.

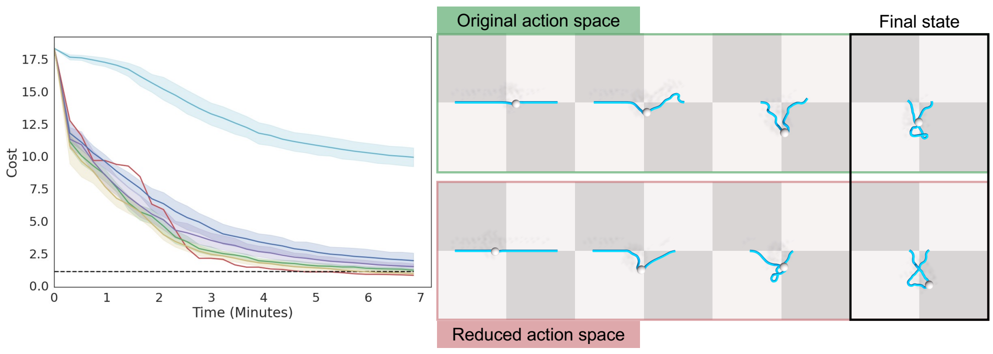
For rope straightening, *Reduced Action Space* finds a better plan than *Original Action Space* using much less time, which is shown in Figure 3.7(a). The *Random Action Space* does not converge to an acceptable solution; the *Original Action Space* converges but takes much longer time; the grid action spaces converge at different speeds depending on the grid resolution. The scripted policy finds a slightly worse solution because it keeps disturbing previously achieved states.

For the rope folding task, as shown in Figure 3.7(b), *Reduced Action Space* still achieves a smaller cost. The *Original Action Space* converges to a similar cost as the *Grid Action Space (4×4)*, whereas the *Grid Action Space (2×2)* yields the worst solution, which reflects that a limited number of picking points are insufficient for handling complex folding tasks with ropes.

For the rope object, the size difference between the *Original Action Space*, ($N_O = 40$), and the *Reduced Action Space* ($\hat{N}_O = 2$ for the straightening task, and $\hat{N}_O = 4$ for the folding task), is not significant. Therefore only a modest gain is seen between the two methods. For

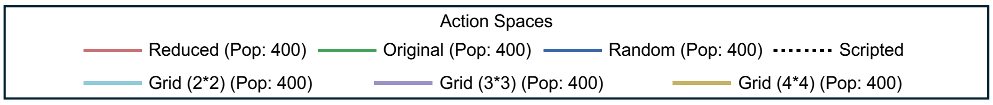


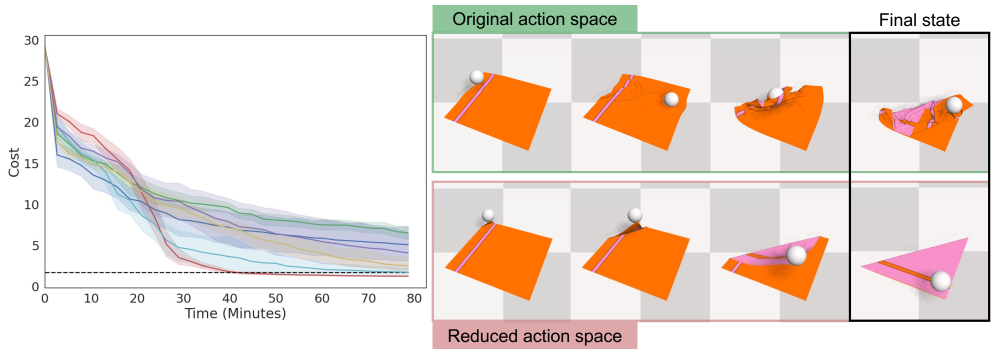
(a) Rope straightening



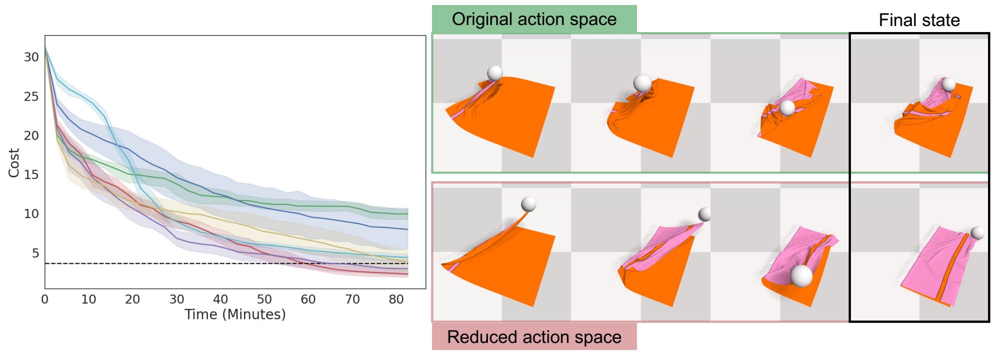
(b) Rope folding

Figure 3.7: The cost versus time of different action spaces for rope manipulation tasks, with CEM as the underlying motion planner, and the snapshots of manipulation plan found by `Original Action Space` and `Reduced Action Space`.

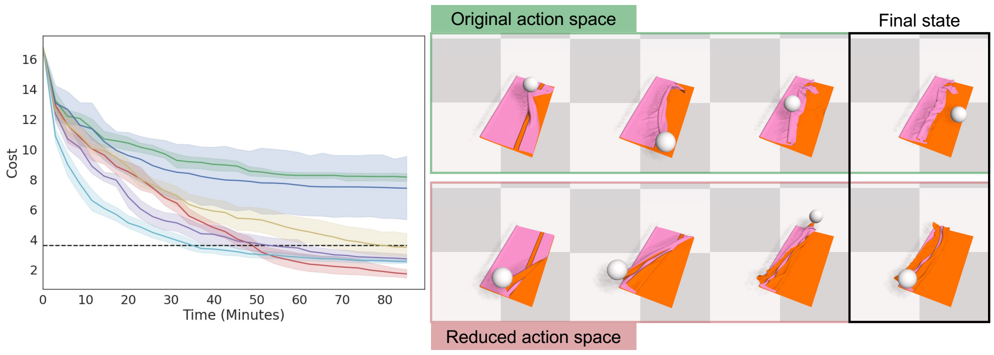




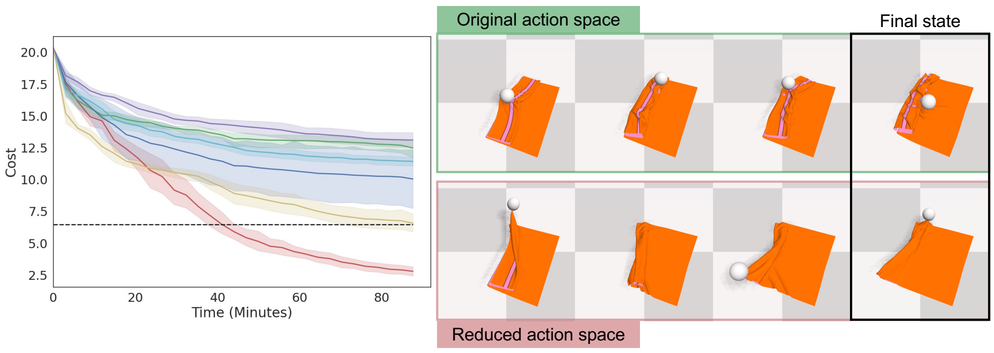
(a) Cloth diagonal folding



(b) Cloth side folding



(c) Cloth reflective folding



(d) Cloth underneath folding

Figure 3.8: The cost versus time of different action spaces for cloth folding tasks, with CEM as the underlying motion planner, and the snapshots of manipulation plan found by `Original Action Space` and `Reduced Action Space`.

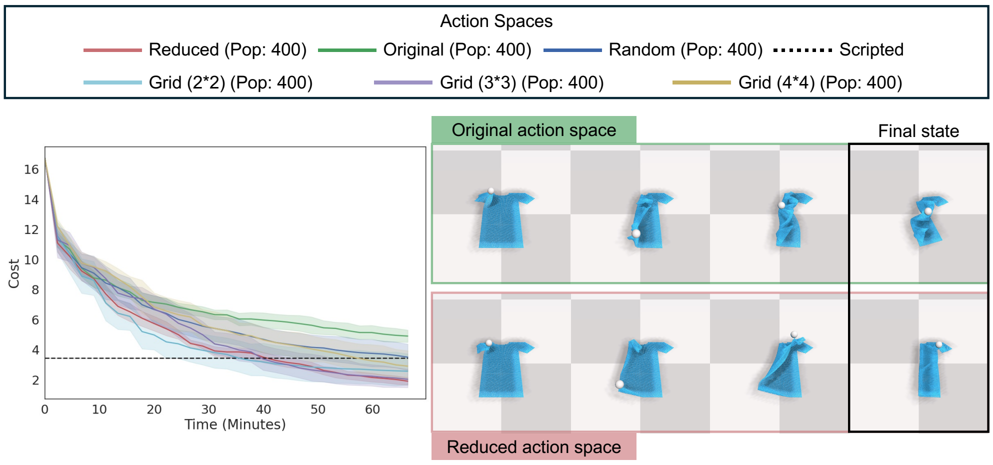


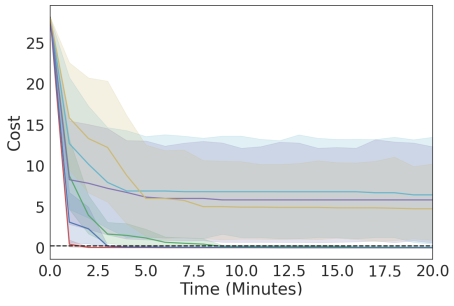
Figure 3.9: The cost versus time of different action spaces for t-shirt side folding task, with CEM as the underlying motion planner, and the snapshots of manipulation plan found by `Original Action Space` and `Reduced Action Space`.

the cloth object however, the original model has $N_O = 10000$ particles, which is much larger than the number of extracted key particles $\hat{N}_O$, as shown in Figure 3.6. Therefore, from Figure 3.8(a-d), it is shown that *Reduced Action Space* achieves a better solution than *Original Action Space* consistently across all four cloth manipulation tasks. For instance, as shown in Figure 3.8(a), the planning cost of *Reduced Action Space* converges to an optimal plan in forty minutes, successfully folding the cloth to match the goal state. In contrast, *Original Action Space* converges to a plan with a much higher cost which results in an incorrect final state. As the difficulty increases from side folding to reflective folding and underneath folding tasks, the gap between the cost curve of *Reduced Action Space* and *Original Action Space* widens. Furthermore, in these more complex folding tasks, the advantage of planning based on the goal-conditioned action space over the simple scripted policy is also increasingly evident.

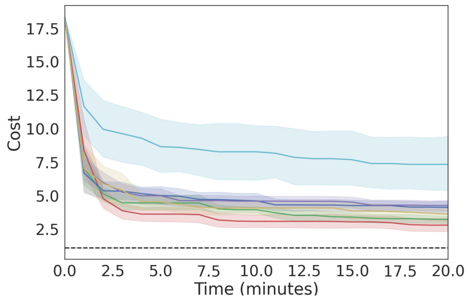
Besides, in all cloth folding tasks, *Random Action Space* achieves slightly better results than *Original Action Space*, which further demonstrates that not every particle on the original model is equally relevant to achieving the goal. For different tasks, the resolution of the *Grid Action Space* has significantly different influences: for cloth diagonal and reflective folding, the *Grid Action Space (2×2)* provides a better solution; for side folding, *Grid Action Space (3×3)* is more effective; for underneath folding, *Grid Action Space (4×4)* achieves better results.

From Figure 3.9, for t-shirt side folding tasks, *Reduced Action Space* converges to a similar cost as the *Grid Action Space* of resolution 4×4, but faster. The *Original Action Space* results in the worst cost.

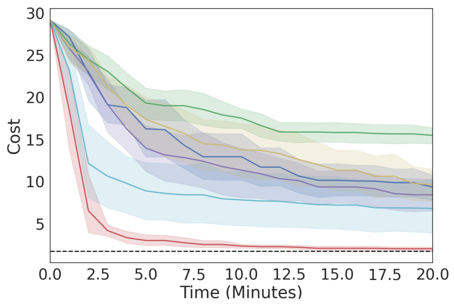
I also run experiments with RRT, a classic sampling-based planning method, to demonstrate that our method is independent of the underlying algorithms of the planner. I implement RRT from Open Motion Planning Library [85] (OMPL), and set the goal bias and error threshold to 0.5 and $1e^{-6}$ respectively. For each task and different action space, I run the planner from



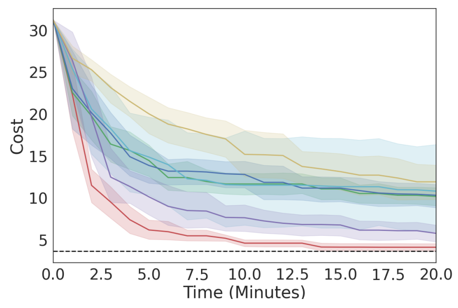
(a) Rope straightening



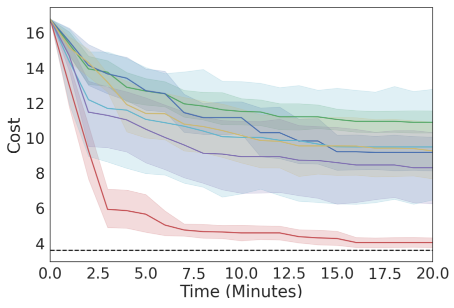
(b) Rope folding



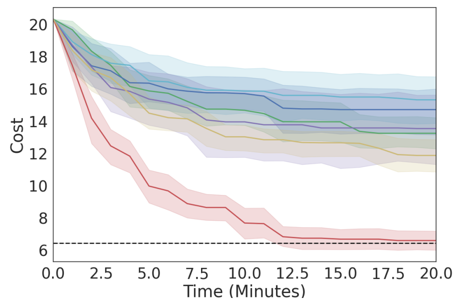
(c) Cloth diagonal folding



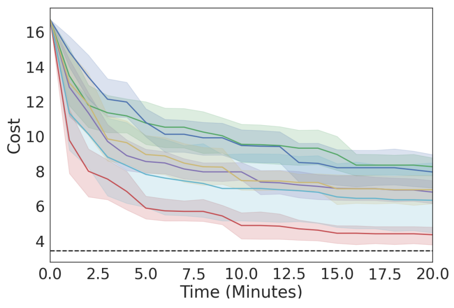
(d) Cloth side folding



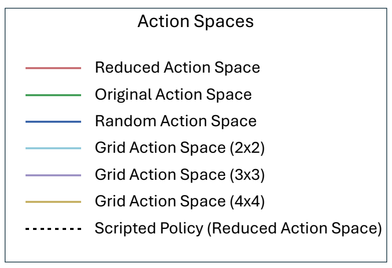
(e) Cloth reflective folding



(f) Cloth underneath folding



(g) T-shirt side folding



(h)

Figure 3.10: The cost versus time of different action spaces, with RRT as the underlying motion planner.

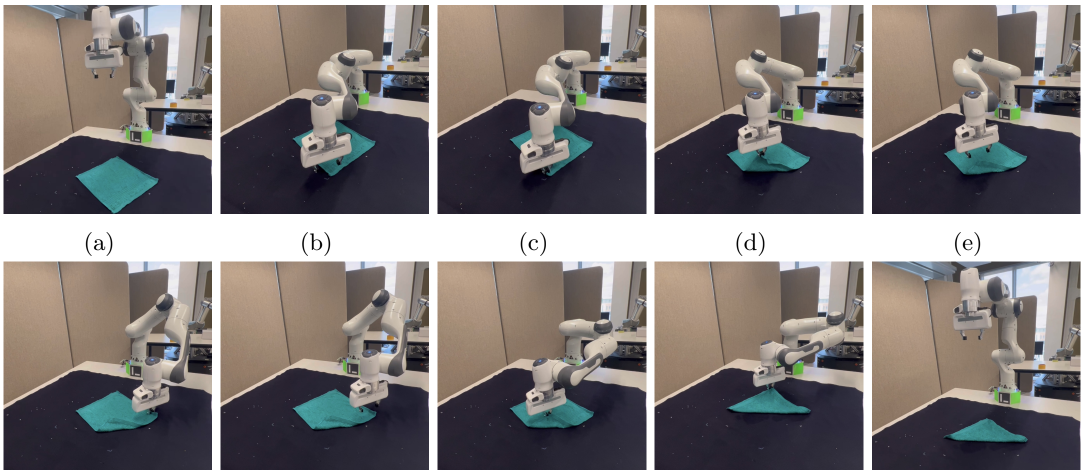

(a) (b) (c) (d) (e)
(f) (g) (h) (i) (j)

Figure 3.11: Real robot demonstration of the cloth diagonal folding task.

one minute to twenty minutes. As shown in Figure 3.10, the results of various action spaces are consistent with those from CEM-based planners, with *Reduced Action Space* consistently achieving the best cost across all tasks.

3.5 Real Robot Experiments

To validate the effectiveness of the planned trajectory in the real world, a Franka Panda robot is used to perform cloth manipulation tasks, including diagonal and side folding. The robot is mounted on a stationary table and a square cloth (30 cm $\times$ 30 cm) is placed in front of it at a pre-determined initial pose. For these demonstrations, no perception is used; instead, the robot execute the computed plan in an open-loop manner.

Firstly, the planned trajectory is converted to several pick and place motions according to the defined action space. For picking a particular point on the cloth, Moveit [81] is used to plan a valid trajectory to a certain height above the target using a position controller. After that an impedance controller is switched on to gradually lower the gripper until it contacts the cloth. The reason for using impedance control is to apply a certain amount of pressure before closing the gripper on the cloth.

Images of the real robot folding the cloth diagonally and sideways are shown in Figure 3.11 and Figure 3.12 respectively. Please refer to the video[1] to view the whole manipulation process. The Panda robot successfully achieves both tasks. Within each task, the gripper can re-grasp the cloth showing that the proposed controlling scheme for executing the plan is effective and the planned trajectory in simulation can be executed successfully to fold the cloth. Nonetheless, the

[1]https://youtu.be/nAYp42WV2g4

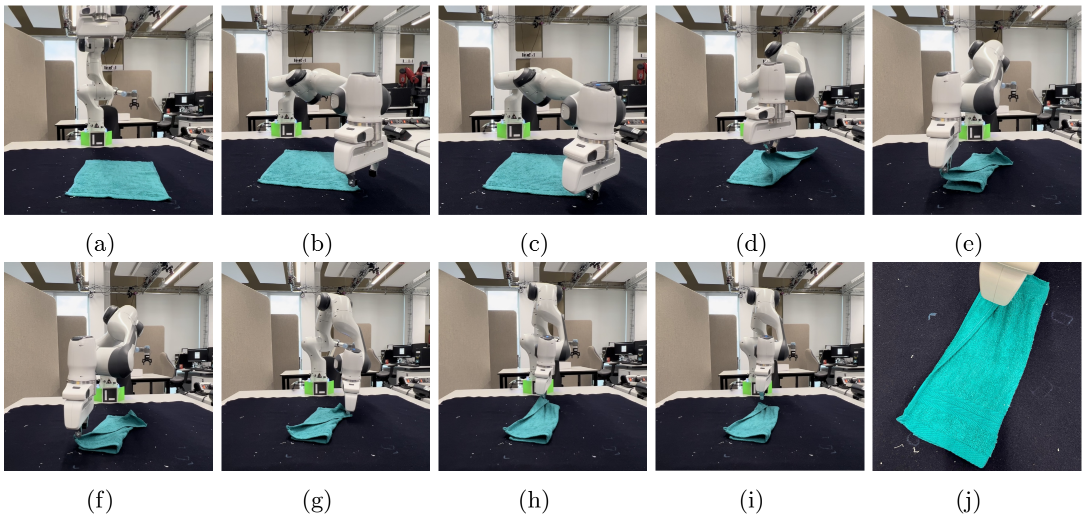


Figure 3.12: Real robot demonstration of the cloth side folding task.

implementation of the real-world experiments is open-loop, and it suffers from the gap between the real cloth physics and the simulation.

3.6 Summary

In this section, I propose reducing the action space for motion planning based on geometric model simplification methods. Two workflows of geometric model simplification and key particle extraction are developed for 1-D linear and 2-D flat objects, respectively. Simulation and real robot experiments demonstrate that the proposed method can improve both the efficiency and performance of a motion planner. The improvement of the reduced action space over the original one increases with the number of particles in the original model and the number of actions required to solve the task. This suggests that model simplification and action space reduction may be critical for handling more complex tasks. Moreover, the proposed method is independent of the underlying dynamics model and the motion planning method used.

However, the computational bottleneck still exists due to time-consuming dynamics models and inefficient planning methods, indicating new directions for future research. Additionally, an exciting future line of investigation will be to combine the proposed method with a high-level planner that generates intermediate goals for complex tasks.

Chapter 4

Dynamics Model Simplification for Deformable Object Manipulation with Closed-Loop Execution

4.1 Introduction

In Chapter 3, I proposed methods to adaptively simplify the geometric model and reduce the action space based on specific goals, leading to improved planning efficiency when compared to using the original action space. However, there are still some limitations:

– The underlying dynamics model used for motion planning is highly detailed and computationally expensive. Although reducing the action space improves planning efficiency to some extent, performing trajectory rollouts on the original model remains costly. With thousands of rollouts required per planning loop, the time cost becomes a significant bottleneck for efficient motion planning in deformable object manipulation.

– The planned trajectory is executed in an open-loop manner on the real robot without feedback from the perception system. This lack of feedback, combined with discrepancies between the simulated and real-world dynamics, results in a low success rate during real-world experiments. This issue is particularly pronounced in tasks like cloth side folding, which involves the switching between neighboring corners, and grasping one corner could make the other change unexpectedly.

In this chapter, I address these drawbacks by introducing simplified dynamics models for

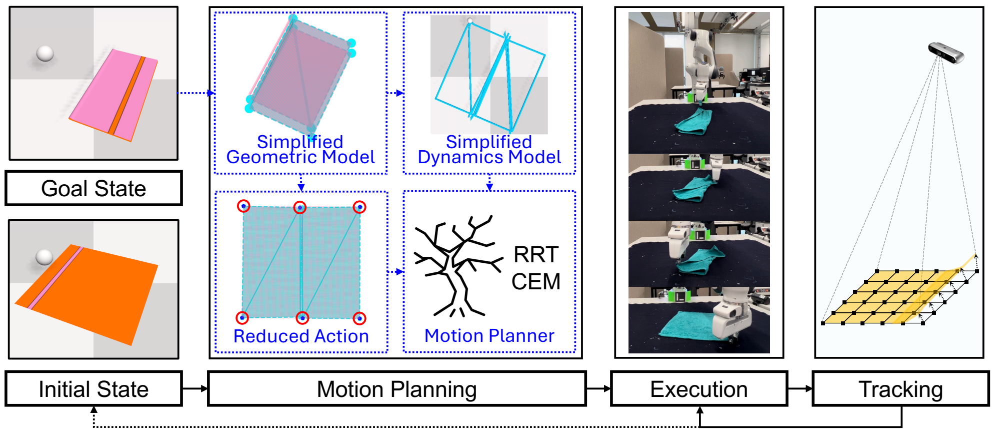


Figure 4.1: Overview of the proposed dynamics model simplification for deformable object manipulation planning with closed-loop execution. Based on the reduced geometric model, a simplified dynamics model is built and used for rolling out trajectories inside the planner. After that, the planned trajectory is converted to pick-and-place actions, and executed in a closed-loop manner with a perception system continually tracking the state of the object.

motion planning and incorporating a perception system to track the shape of the deformable continuously during manipulation.

To tackle the high dimensional state space and computational cost associated with expensive dynamics models in motion planning, previous researchers have proposed using approximate models instead of high fidelity ones, typically tailored to specific tasks [102, 101]. For particle based models, a straightforward way to simplify the system is by using fewer particles. For example, as shown in Figure 4.4, grid-based models with coarse dynamics can be used as effective approximations for the original dynamics model for certain tasks, instead of having dense particles throughout a piece of cloth. However, this uninformed simplification lacks adaptability to different goals and only suitable for specific tasks. The grid resolution, which needs to be predetermined by humans, also impacts its effectiveness. To overcome these limitations, I again take a goal-informed approach, and propose building simplified dynamics models tailored to specific goals. This approach leverages the geometric model simplification methods described in Chapter 3.

An exemplary case is shown in Figure 4.1, where a simplified geometric model of six triangles is derived from the goal state of the original geometric model; the six key particles corresponding to this simplified model are used as the picking points in the reduced action space, and a simplified dynamics model is constructed with only the particles on the edges of the simplified geometric model. Compared to the uninformed grid-based model, such as Figure 4.4(d), this goal-informed simplified model provides a more task-relevant representation, which has two distinctive halves, with the central edge aligned along the folding line. This model enables much faster trajectory rollout within the planning framework while adapting to various goals.

Based on the simplified dynamics model established in this chapter and the reduced action space from Chapter 3, a motion planner is invoked to plan a trajectory, which should be significantly faster than planning with the high-fidelity original model. Once planning is complete, the trajectory is ready for execution by the robot. However, due to the challenge of accurately modeling a random deformable object in the real world, coupled with the fact that the plan is generated based on a reduced model optimized for efficiency, the behavior of the object, in reality, may differ from the simulation. To address this discrepancy, a perception system is needed to continuously track the shape of the deformable object during manipulation, ensuring robust execution of the planned trajectory despite uncertainties in the dynamics of the object and the environment. If the planned trajectory fails to bring the object to the goal, with the help of the perception system, a new trajectory can be generated accordingly.

Despite the importance of the tracking system, accurately tracking deformable objects is a highly complex task for several reasons. First, the high-dimensional state space of the deformable object makes it difficult to precisely describe its shape. Second, the object can be occluded by the environment—including the robot itself—and can experience severe self-occlusion. Third, continuously tracking the shape with a limited computational budget adds further difficulty.

To tackle this, I implement the CDCPD method from [64], which does not rely on any physics model and is thus fast enough for real-time tracking. However, CDCPD struggles in scenarios involving multiple pick-and-place actions and when the deformable object is relatively free, as opposed to being pre-grasped. To enhance tracking accuracy and robustness, I make two modifications to the tracking system:

– Point cloud occlusion reasoning: By explicitly reasoning about point cloud occlusion, the system generates a thicker point cloud, improving the tracking of deformable objects with multiple layers especially when being folded.

– Selective tracking point updates: Instead of relocating every tracking point with each updated point cloud, the system retains tracking points in regions of the point cloud that remain unchanged between frames. Only tracking points in areas where significant changes are detected are updated.

These modifications enhance the ability of the tracking system to accurately monitor and adapt to changes in the shape of the deformable object.

To verify the effectiveness of the proposed methods, I first conduct simulation experiments in Section 4.5, comparing the performance of motion planning with the goal-informed simplified dynamics model against baseline models. Then, in Section 4.6.1, I evaluate the performance of the improved CDCPD algorithm through cloth shape tracking experiments. Finally, in Section 4.6.2, I conduct real-world cloth folding experiments with closed-loop execution, demonstrating the practical applicability of the approach.

In summary, the contributions of this chapter include:

– A systematic approach for constructing simplified dynamics models based on the original dynamics model and the simplified geometric model.

– A modified shape tracking algorithm for real-time monitoring of deformable object states during manipulation.

– A closed-loop experimental system integrating perception, planning, and control for real-world deformable object folding tasks using a Franka Panda robot.

The organization of this chapter is as follows. In Section 4.2, I illustrate the problem formulation of the motion planning problem for deformable object manipulation with a simplified dynamics model, and describe the shape tracking problem of deformable objects given the point cloud from the depth sensor. In Section 4.3 and Section 4.4, I introduce the methods for simplifying the dynamics model and the improved deformable object shape tracking algorithm based on CDCPD. In Section 4.5, I present the results from simulation experiments, comparing the performance of planning with the goal-informed simplified dynamics model against several baseline dynamics models. In Section 4.6, I demonstrate the effectiveness of the improved tracking algorithm through cloth shape tracking experiments, followed by real robot experiments on cloth folding tasks with the closed-loop execution system. In Section 4.7, I summarize the advantages and limitations of the proposed methods in this chapter.

4.2 Problem Formulation

4.2.1 Motion Planning for Deformable Object Manipulation

In this chapter, I address the same motion planning problem for deformable object manipulation as defined in Section 3.2, including the state space, action space, and cost function. However, unlike the previous chapter where the original, high-fidelity dynamics model is used throughout the motion planning process, which is extremely time-consuming, I introduce the use of simplified dynamics models to enhance efficiency. The computationally expensive high-fidelity model is only utilized once to roll out the planned trajectory after motion planning, while the simplified dynamics model is employed within the planner to boost computational efficiency.

4.2.2 Deformable Object Shape Tracking

The aim of tracking the shape of the deformable object is to monitor its state, denoted as ξ^t, during manipulation while ensuring the geometric model $\mathcal{G}(V, E, L)$ remains stable, based on the raw point cloud input $\mathcal{I}^t$ from the depth sensor. However, given the high-dimensional nature of the original geometric model and the limited computational resources for tracking, I instead compress the geometric model and the state into a lower-dimensional representation, $\bar{\xi}^t$ and $\bar{\mathcal{G}}(\bar{V}, \bar{E}, \bar{L})$, from which the full state can be recovered.

Tracking the deformable object can then be formulated as a point registration problem, where the goal is to estimate $\bar{\xi}^{t+1}$ by aligning $\bar{\xi}^{t}$ to the new point cloud input $\mathcal{I}^{t+1}$:

$$\bar{\xi}^{t+1} \leftarrow \texttt{Align}(\bar{\xi}^{t}, \mathcal{I}^{t+1}) \tag{4.1}$$

Let the true configuration of the deformable object at time step $t+1$ be represented as $\tilde{\xi}^{t+1}$. The objective is to minimize the difference between the estimated state $\bar{\xi}^{t+1}$ and the true configuration $\tilde{\xi}^{t+1}$, expressed as:

$$\min_{\bar{\xi}^{t+1}} \|\bar{\xi}^{t+1} - \tilde{\xi}^{t+1}\|_2 \tag{4.2}$$

4.3 Motion Planning with Simplified Dynamics Model

In this section, based on the geometric model simplification and action space reduction methods proposed in Section 3.3, I present a planning framework for deformable object manipulation that utilizes simplified dynamics models.

Specifically, in Section 4.3.1, I provide an overview of the proposed planning framework. Then in Section 4.3.2, I detail the dynamics model simplification informed by the goal. Lastly in Section 4.3.3, I introduce the uninformed dynamics model simplification method.

4.3.1 Framework Overview

To alleviate the heavy computational burden of motion planning using time-consuming, high-fidelity dynamics models with thousands of particles, I propose a motion planning framework based on coarse yet more efficient simplified models, as outlined in Algorithm 3. This framework is general and does not rely on the specific model simplification techniques employed.

The inputs of the framework include the original geometric model $\mathcal{G}_O$, the original dynamics model $\mathcal{D}_O$, the initial and goal state of the original model ξ_O^0 and ξ_O^G, and a simplified geometric model $\hat{\mathcal{G}}_S$ to be used for dynamics model simplification. The output is the planned trajectory for the robot to execute

Since the goal-conditioned action space reduction has been shown to be effective in Chapter 3, it forms the basis of Algorithm 3 (Line 1). The next step is to build a simplified dynamics model using the original dynamics model and the simplified geometric model (Line 2). Then, the initial state, goal state, and the reduced picking indexes are mapped from the original model to the simplified model (Line 3). Finally, a motion planner is invoked to plan a trajectory based on the simplified dynamics model and the corresponding initial state, goal state, and picking indexes in the action space (Line 4).

In the following subsections, I will introduce two representative implementations of the dynamics model simplification function (`Simplify_Dynamics`), including a goal-informed approach in Section 4.3.2 and an uninformed approach in Section 4.3.3.

Algorithm 3: Motion Planning with Simplified Dynamics Model

```
Input: $\mathcal{G}_O$, $\mathcal{D}_O$, $\xi_O^0$, $\xi_O^G$, $\hat{\mathcal{G}}_S$
Output: $\tau$
1 $V_O^{\mathcal{A}} \leftarrow$ Reduce_Action_Space($\mathcal{G}_O, \xi_O^G$);
2 $\mathcal{D}_S \leftarrow$ Simplify_Dynamics($\mathcal{D}_O, \hat{\mathcal{G}}_S$);
3 $\xi_S^0, \xi_S^G, V_S^{\mathcal{A}} \leftarrow$ Map($\mathcal{D}_O, \mathcal{D}_S, \xi_O^0, \xi_O^G, V_O^{\mathcal{A}}$);
4 $\tau \leftarrow$ Planner($\mathcal{D}_S, V_S^{\mathcal{A}}, \xi_S^0, \xi_S^G$);
```

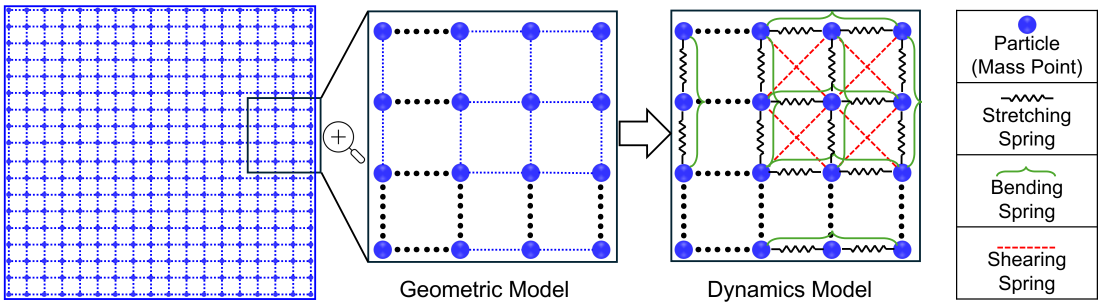


Figure 4.2: Original geometric model and dynamics model. The geometric model is composed of nodes and edges with weights assigned to each edge representing the resting length. While the dynamics model builds on this structure by placing particles at each node and establishing various spring connections between these particles.

4.3.2 Goal-Informed Dynamics Model Simplification

Before I explain how the `Simplify_Dynamics` function in Algorithm 3 works, I will first explain how the original dynamics model $\mathcal{D}_O(\mathcal{G}_O, \mathcal{M}_O, \mathcal{R}_O)$ and the original geometric model $\mathcal{G}_O$ are related. As illustrated in Figure 4.2, for each vertex of the geometric model, a particle with mass and radius is generated in the dynamics model. All particles in the dynamics model are assigned the same mass depending on the materials and properties of the object, forming the dynamics properties of particles $\mathcal{M}_O$. Then, for neighboring particles, a stretching spring is added; for particles that are two steps away, a bending spring is added; for diagonal neighboring particles, a shearing spring is added; additional constraints of self-collision and environmental collision are also considered, forming the dynamics relations of particles $\mathcal{R}_O$.

I build the simplified dynamics model $\mathcal{D}_S$ similarly, with a key feature being its adaptive construction based on the specific goal. Thus, I propose a goal-informed dynamics model simplification method, as detailed in Algorithm 4. The inputs include the original dynamics model $\mathcal{D}_O$ and a simplified geometric model $\hat{\mathcal{G}}_S$, extracted from the goal using the geometric model simplification function defined in Algorithm 2. However, the reduced geometric model $\hat{\mathcal{G}}_S$ may contain significantly long edges, e.g., the nine edges in Figure 4.3(b). To express

Algorithm 4: Dynamics Model Simplification (`Simplify_Dynamics`)

```
   Input: $\mathcal{D}_O(\mathcal{G}_O, \mathcal{M}_O, \mathcal{R}_O)$, $\hat{\mathcal{G}}_S(\hat{V}_S, \hat{E}_S, \hat{L}_S)$
   Output: $\mathcal{D}_S(\mathcal{G}_S, \mathcal{M}_S, \mathcal{R}_S)$
1  $V_S \leftarrow \hat{V}_S$; $E_S, L_S, \mathcal{M}_S, \mathcal{R}_S \leftarrow None$;
2  for $\hat{e}(u, v)$ in $\hat{E}_S$ do
3      $V_S^{\hat{e}} \leftarrow None$; $u_{new}$, $v_{new} \leftarrow u$;
4      do
5          $w_{new} \leftarrow u_{new}$, $u_{new} \leftarrow v_{new}$;
6          $v_{new} \leftarrow$ New_Vertice($u_{new}, radius$);
7          $V_S \leftarrow V_S \cup \{v_{new}\}$, $V_S^{\hat{e}} \leftarrow V_S^{\hat{e}} \cup \{v_{new}\}$;
8          $E_S \leftarrow E_S \cup \{(u_{new}, v_{new})\}$;
9          $L_S \leftarrow L_S \cup \{radius\}$;
10         $\mathcal{R}_S \leftarrow \mathcal{R}_S \cup$ Add_Strech($u_{new}, v_{new}$) $\cup$ Add_Bend($w_{new}, v_{new}$);
11     while $\hat{L}_S(u, v) - L_S(u, v_{new}) > radius$;
12     $\mathcal{M}_S^{\hat{e}} \leftarrow$ Mass_Aggregate($\mathcal{M}_O, \hat{e}$);
13     $\mathcal{M}_S \leftarrow \mathcal{M}_S \cup$ Mass_Average($\mathcal{M}_S^{\hat{e}}, V_S^{\hat{e}}$);
14 end
15 $\mathcal{G}_S \leftarrow (V_S, E_S, L_S)$, $\mathcal{D}_S \leftarrow (\mathcal{G}_S, \mathcal{M}_S, \mathcal{R}_S)$;
```

the deformation capabilities of these long edges in the dynamics model, I interpolate particles along these edges (blue points in Figure 4.3(d)). For each edge $\hat{e}(u, v)$ (Line 2) in the reduced geometric model $\hat{\mathcal{G}}_S$, vertices are gradually added along the edge between the start vertex u and the end vertex v, based on a preset radius (Line 5-6). This results in a set of interpolated vertices V_S, and corresponding edges E_S and rest lengths L_S (Line 7-9). Stretching constraints (black springs in Figure 4.3(d)) are applied between neighboring particles, and bending constraints (green connections in Figure 4.3(d)) are applied between particles two steps apart (Line 10). Next, I align the simplified geometric model with the original dynamics model at the flattened state, as shown in Fig. 4.3(c), and aggregate the masses of the particles from the original dynamics model, $\mathcal{D}_O$, that are closest to the current edge $\hat{e}(u, v)$ (Line 12):

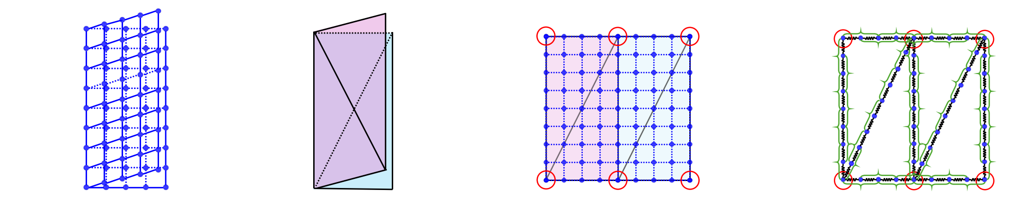
$$V_O^{\hat{e}} \leftarrow \{v_o \in V_O | \arg\min_{\hat{e}' \in \hat{E}_S} dis(v_o, \hat{e}') = \hat{e}\} \tag{4.3}$$

$$\mathcal{M}_S^{\hat{e}} \leftarrow \sum_{v_o \in V_O^{\hat{e}}} \mathcal{M}_O^{v_o} \tag{4.4}$$

where $V_O^{\hat{e}}$ denotes the vertices in the original model closest to the edge $\hat{e}$, and $\mathcal{M}_O^{v_o}$ is the mass of v_o. Then, this mass is distributed equally among the particles added to the edge (Line 13):

$$\mathcal{M}_S^{v} \leftarrow \frac{\mathcal{M}_S^{\hat{e}}}{|V_S^{\hat{e}}|}, \text{ for } v \in V_S^{\hat{e}}, \tag{4.5}$$

where $V_S^{\hat{e}}$ represents the vertices interpolated along the edge $\hat{e}$ of the simplified geometric model.



(a) Original geometric model (b) Simplified geometric model (c) Reduced action space (d) Simplified dynamics model

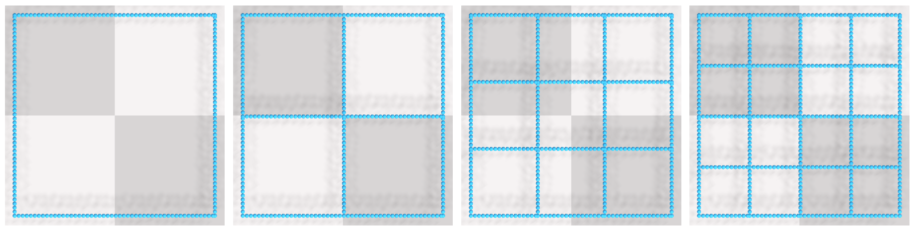
Figure 4.3: Goal-informed model simplification for cloth side folding. The original geometric model at the folded state is shown in (a); I simplify it using the proposed Algorithm 2 and get a simplified geometric model in (b); by extracting the nearest particle on the original model, six particles on the original model are identified as picking points in the reduced action space, as shown in (c); a simplified dynamics model is built based on the simplified geometric model, with particles and springs along the edges, and the corresponding particles for picking are extracted accordingly, as illustrated in (d)

By avoiding the redistribution of mass for each particle separately based on the distances between the corresponding particles on the original dynamics model and the simplified dynamics model, which could lead to uneven mass distribution, the simplified dynamics model performs better. Finally, the complete simplified geometric model $\mathcal{G}_S$, mass distribution $\mathcal{M}_S$, and dynamics relations $\mathcal{R}_S$ are acquired, giving us the simplified dynamics model $\mathcal{D}_S$ for the target task (Line 15). As demonstrated in Figure 4.3(d) a simplified mass-spring model with only the particles along the edges is built, which, compared to the original model, has a much smaller number of particles and constraints, thereby allowing for faster simulation.

4.3.3 Uninformed Dynamics Model Simplification

As mentioned earlier, another approach to simplifying the dynamics model is to uniformly reduce the resolution of the original dynamics model without relying on specific task goals. This can be achieved in various ways, but one effective method is to build grid-based dynamics models at different resolutions systematically. For example, as illustrated in Figure 4.4, a set of grid-based dynamics models is created by overlaying grids of different resolutions on top of the original model. In this approach, only the particles and springs located along the grid edges are retained, while those inside the grid are discarded, resulting in a sparse, hollow grid structure. This uniform reduction method simplifies the model by removing internal particles, focusing solely on the structure along the edges of the grid. By doing so, the simplified model reduces the computational burden associated with simulating the dynamics of the object. This grid-based simplification approach is general and does not rely on specific goals or task-specific information. It can adapt to various shapes by mapping the grid nodes to their nearest corresponding particles on the original model.

Usually, a lower resolution grid model is faster to simulate but offers lower precision, whereas a higher resolution grid model provides better precision at the cost of increased computation



(a) Grid (2×2) (b) Grid (3×3) (c) Grid (4×4) (d) Grid (5×5)

Figure 4.4: Grid-based dynamics model simplification. In grid models, particles are placed along the edges, leaving hollow spaces inside, which maintain a general approximation uninformed by specific goals and tasks.

time.

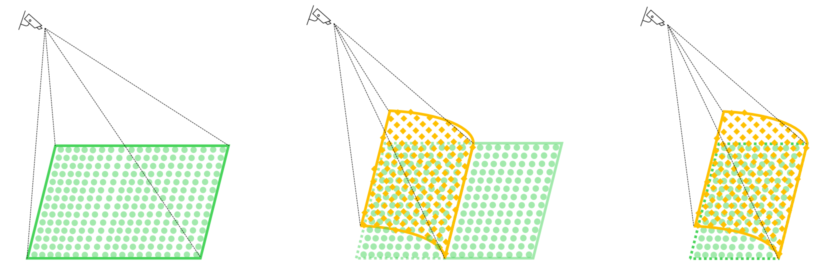
4.4 Deformable Object Shape Tracking

After motion planning, the planned trajectory is converted into a sequence of pick-and-place actions that the robot executes. However, due to the discrepancies between the dynamics model used in simulation and the actual dynamics of deformable objects in the real world, these actions may not produce the expected results. For example, if a pick-and-place action fails to manipulate the deformable object to the desired intermediate state, subsequent pick action can pick at the wrong location on the deformable object if performed in an open-loop manner.

To address this issue, I propose executing the trajectory in a closed-loop manner. This involves continuously monitoring the state of the deformable object (e.g., cloth) during manipulation, allowing the system to adapt and make real-time adjustments as necessary. This approach ensures that if a pick-and-place action does not yield the expected result, subsequent actions can be adjusted based on the actual state of the object.

Given the general applicability of the tracking method and the time-sensitive nature of the tracking task, I implement the CDCPD method from [64], and enhance its performance for the specific tasks tested in this chapter by introducing improvements in two ways: 1) reasoning explicitly about self-occlusion to generate a thicker point cloud, and 2) selectively updating the tracking points to maintain accuracy and robustness.

In the following sections, I first describe how self-occlusion is handled to generate a more complete point cloud (Section 4.4.1). Then, I introduce the general pipeline of CDCPD and explain the selective tracking point update method used to achieve better performance (Section 4.4.2). These improvements enable more reliable tracking during manipulation, enhancing the overall robustness of the execution.



(a) Previous point cloud (green)

(b) Current point cloud (yellow)

(c) Combined point cloud

Figure 4.5: A schematic illustration of the point cloud combination process. The first image displays the initial point cloud in green, while the second image shows the current point cloud in yellow, overlapping with the initial one. Given that the movement of the underlying layers of the cloth is penalized, it is reasonable to speculate that some points of the cloth are occluded by the yellow surface point cloud. By combining these point clouds, we can thicken the overall point cloud, which enhances the tracking algorithm to accurately measure the state of the cloth.

4.4.1 Point Cloud Combination with Occlusion Reasoning

A point cloud is a collection of data points representing the surface of an object in 3D space, where each point corresponds to a specific location sampled by a depth sensor. While point clouds provide accurate spatial information, they capture only the visible portions of an object, leaving areas that are occluded by other parts of the object or the environment inaccessible. This makes it challenging to precisely track the complete configuration of deformable objects, especially those with complex shapes like folded cloth.

One common method to address occlusion involves fusing data from multiple depth cameras placed at different angles. Although this approach can improve the overall coverage of the object, it introduces extra complexity due to the need for additional hardware calibration efforts. Furthermore, for highly deformable objects like layered cloth, even multiple depth sensors may not suffice to resolve occlusions fully.

To tackle these limitations, I propose a method to combine consecutive point clouds from a single depth camera by explicitly reasoning about self-occlusion. This approach generates a “thicker” point cloud, providing a more complete representation of the shape of the object by incorporating information from previous frames into the current estimation. By doing so, the perception system can better track and estimate the state of the deformable object, even when parts of it are hidden from direct view.

For example, as shown in Figure 4.5(a), the initial point cloud (green color) of the deformable object in its flattened state is captured by the depth camera. After executing a pick and place action, a new point cloud is captured, depicted by yellow dots in Figure 4.5(b), overlapping with the previous point cloud. In this case, the previous point cloud (in green) is split into two halves: the left half is occluded by the current point cloud (in yellow), while the right half should remain visible but relocated. Given that the planner penalizes the movement of the underlying layer during manipulation, it is reasonable to assume that the occluded half is still there even though not captured by the camera. Thus, it can be combined with the current point cloud to generate a thicker point cloud, as shown in Figure 4.5(c), facilitating the tracker to reason about the distribution of the layers better.

4.4.2 CDCPD with Selective Point Updating

Given the combined point cloud $\mathcal{I}^t$ from the depth camera, CDCPD is implemented and improved to track the shape changes of the deformable object during manipulation.

In CDCPD, the shape tracking is formulated as a point set registration problem, as defined in Equation 4.1, with the goal specified in Equation 4.2. However, directly optimizing this goal is challenging due to the unavailability of the ground-truth shape of the deformable object, particularly in real-world scenarios.

To address this, in [64], CDCPD first performs registration between $\mathcal{I}^{t+1}$ and $\bar{\xi}^t$ based on a Gaussian Mixture Model (GMM) and Expectation-Maximization (EM) method. A visibility prior is introduced to adaptively adjust the weights, and the outputs are regularized for topological consistency using Coherent Point Drift (CPD) and Locally Linear Embedding (LLE). Finally, tracking points are optimized to adhere to stretching constraints and prior correspondence constraints, and additional improvement of selective point updating is applied in this post-process phase.

In the following subsections, I will briefly introduce the core ideas of the GMM & EM, visibility prior, CPD, LLE from the original work of [64, 65, 133], and emphasize the improvement of selective point updating in the post-process phase.

Gaussian Mixture Model & Expectation Maximization

Given the point registration problem defined in Equation 4.1, a GMM is readily generated with the mean of each Gaussian distribution at the position of each point in $\bar{\xi}^t$. All points from the perceived point cloud $\mathcal{I}^{t+1}$ is deemed to be generated from this GMM with equal variance σ^2 and membership probability $\frac{1}{N_\xi}$, where N_ξ denotes the number of tracking points in $\bar{\xi}$. The probability of a point $q_j \in \mathcal{I}^{t+1}$ originating from the tracking point $\bar{p}_i \in \bar{\xi}^t$ can be written as:

$$\mathbb{P}(q_j) = \sum_{i=1}^{N_\xi} \frac{1}{N_\xi} \frac{1}{(2\pi\sigma^2)^{\frac{D}{2}}} \exp(-\frac{\|q_j - \bar{p}_i\|^2}{2\sigma^2}) \tag{4.6}$$

where $D = 3$ is the dimensionality of the position.

To account for outliers, an additional noise component is included with weight ω, where $0 \leq \omega \leq 1$. The joint probability density function of the GMM is given by:

$$\mathbb{P}(\mathcal{I}^{t+1}) = \prod_{j=1}^{N_\mathcal{I}} \mathbb{P}(q_j) = \prod_{j=1}^{N_\mathcal{I}} \sum_{i=1}^{N_\xi+1} \mathbb{P}(i)\mathbb{P}(q_j|i) \tag{4.7}$$

where $N_{\mathcal{I}}$ is the number of points in the point cloud $\mathcal{I}^{t+1}$, and

$$\mathbb{P}(q_j|i) = \begin{cases} \frac{1}{(2\pi\sigma^2)^{\frac{D}{2}}} \exp(-\frac{\|q_j - p_i\|^2}{2\sigma^2}), & i = 1, ..., N_\xi \\ \frac{1}{N_{\mathcal{I}}}, & i = N_\xi + 1 \end{cases} \tag{4.8}$$

$$\mathbb{P}(i) = \begin{cases} \frac{1-\omega}{N_\xi}, & i = 1, ..., N_\xi \\ \omega, & i = N_\xi + 1 \end{cases} \tag{4.9}$$

The objective is to find $\bar{\xi}^{t+1}$ and σ^2 that maximize the log-likelihood of Equation4.7. This is achieved through the EM algorithm. During the E-step, the posterior probability is computed using Bayes rule:

$$\mathbb{P}^{cur}(i|q_j) = \frac{\exp(-\frac{1}{2}\|q_j - p_i\|^2)}{\sum_{i=1}^{N_\xi} \exp(-\frac{1}{2}\|\frac{q_j - p_i}{\sigma}\|^2) + \frac{(2\pi\sigma^2)^{\frac{D}{2}}\omega N_\xi}{(1-\omega)N_{\mathcal{I}}}} \tag{4.10}$$

And in the M-step, the optimal $\bar{\xi}^t + 1$ and σ^2 are found by minimizing the following cost:

$$Q(\bar{\xi}^{t+1}, \sigma^2) = -\sum_{j=1}^{N_{\mathcal{I}}} \sum_{i=1}^{N_\xi} \mathbb{P}^{cur}(i|q_j)(\log(\frac{1-\omega}{N_\xi(2\pi\sigma^2)^{\frac{D}{2}}}) - \frac{\|q_j - p_i\|^2}{2\sigma^2}) - \sum_{j=1}^{N_{\mathcal{I}}} \mathbb{P}(N_\xi + 1|q_j) \log(\frac{\omega}{N_{\mathcal{I}}}) \tag{4.11}$$

The E-step and M-step are iterated until convergence.

Visibility Prior

When the deformable object is occluded by other objects, the GMM tends to overfit to the visible part of the point cloud, which can adversely affect the tracking of points that are hidden or occluded. To mitigate this issue, it is crucial to adjust the weights of the Gaussian distributions in the GMM to better reflect the visibility of the tracking points. A visibility prior is then introduced to allocate more weight to the visible tracking points, while tracking points that are occluded are assigned smaller weights or even zero weight. This adjustment ensures that occluded points do not contribute to the distribution of the current point cloud. For more details on implementing this visibility prior and its effects, refer to [64].

Coherent Point Drift

In the GMM registration process described earlier, all Gaussian distributions are assumed to be independent to each other, which is not accurate for the deformable object model. In reality, neighboring tracking points are likely to have similar movement between consecutive frames. To address this issue, the Coherent Point Drift (CPD) method is employed to regulate the registration process. CPD uses a Gaussian Radial Basis Function Network to model the spatial transformation of the tracking points between frames, ensuring that nearby points move coherently. This adjustment leads to more accurate tracking of the deformable object. For further details on the Coherent Point Drift method and its application to point set registration, refer to [64] and [133].

Locally Linear Embedding

Another limitation of the GMM-based registration is that the topology of the tracking points, as defined by the geometric model G, is overlooked. This can lead to drift from the original model due to accumulated errors. To address this issue, the Locally Linear Embedding method is employed to preserve the original structure of the tracking points. LLE builds local relationships between each tracking point and its k-nearest neighbors, and penalizes the deviations in the tracking results across frames. This approach helps maintain the geometric integrity of the deformable object over time. For a detailed explanation of the LLE process, refer to [64].

Constrained Optimization with Selective Point Updating

To avoid overstretching the edges in the tracking result, the output of GMM and EM can be further optimized through post-processing. A constrained optimization problem is formulated accordingly, with the goal of refining the tracking result while enforcing certain constraints such as the stretching limits:

$$\begin{aligned} \bar{\xi}^{*t+1} &= \underset{\bar{p}_k^* \in \bar{\xi}^*,\ \bar{p}_k^{t+1} \in \bar{\xi}^{t+1}}{\arg\min} \sum_{k=1}^{N_\xi} \|\bar{p}_k^* - \bar{p}_k^{t+1}\|^2 \\ s.t. \quad & \|\bar{p}_i^* - \bar{p}_j^*\| \le \lambda \|\bar{p}_i^0 - \bar{p}_j^0\|,\ \forall (i,j) \in \bar{E} \end{aligned} \tag{4.12}$$

where λ denotes the coefficient of maximum allowable stretching of the edges.

This is solved using the Gurobi Package [134] efficiently.

In [64], a prior correspondence constraint is also added to the optimization problem, with the gripper pre-grasping a given point on the deformable object. Whereas in the scenarios studied in this thesis, the gripper does not pre-grasp the deformable object, and requires multiple regrasps during manipulation. Thus, employing such a constraint may not be suitable and could potentially degrade performance.

Nevertheless, it is still useful when coupled with explicit reasoning of self-occlusion, which I use to combine the point clouds previously. Specifically, in Section 4.4.1, the previous point cloud is split into occluded and unoccluded parts. It is reasonable to assume that the tracking points within the occluded area remain unchanged after the pick and place action, while only the tracking points inside the unoccluded part need to be updated. Thus, I introduce an additional constraint to ensure that the tracking points, which do not move between consecutive point clouds, are fixed at their previous positions. The modified constrained optimization formulation can then be written as:

$$\begin{aligned} \bar{\xi}^{*t+1} = & \underset{\bar{p}_k^* \in \bar{\xi}^*,\ \bar{p}_k^{t+1} \in \bar{\xi}^{t+1}}{\arg\min} \sum_{k=1}^{N_\xi} \|\bar{p}_k^* - \bar{p}_k^{t+1}\|^2 \\ s.t. \quad & \|\bar{p}_i^* - \bar{p}_j^*\| \leq \lambda \|\bar{p}_i^0 - \bar{p}_j^0\|,\ \forall (i,j) \in \bar{E} \\ & \bar{\xi}_{fixed}^* = \bar{\xi}_{fixed}^{*t} \end{aligned} \tag{4.13}$$

where $\bar{\xi}_{fixed}^{*t}$ represents the set of tracking points from previous time step t that located in occluded area of the point cloud, and thus is fixed at time step $t+1$. In this way, only the tracking points located in the unoccluded area are being updated at each time step.

4.5 Simulation Experiments

The experiments conducted in this section aim to answer three key questions:

– Efficiency Gains: How much can the simplified dynamics model improve the efficiency of motion planning for deformable object manipulation?

– Optimality Trade-offs: How much optimality does the simplified dynamics model sacrifice to achieve this efficiency?

– Comparison of Simplification Approaches: What are the advantages and disadvantages of goal-informed model simplification compared to uninformed model simplification?

The tasks tested in this section include the cloth and t-shirt manipulation tasks from Section 3.4.1. Rope manipulation tasks are excluded because the rope model is already efficient for motion planning, but rope-like models are used as approximations for the cloth dynamics model due to their computational efficiency.

The framework remains independent of the planning algorithm used. To provide a more comprehensive evaluation, I test different dynamics models using both the optimization-based planning method CEM and the sampling-based method RRT.

All simulations are performed using the SoftGym simulator, and the same workstation, with Intel(R) Core(TM) i9-11900 CPU @2.50GHz, 32 GB of RAM, and Nvidia GeForce GTX 4090 GPU, as in Section 3.4.

In the following sections, I will detail the implementation of the proposed method and baseline methods in Section 4.5.1, and present the comparative results and an analysis of the findings in Section 4.5.2.

4.5.1 Implementation & Baselines

For a fair comparison, I run both the CEM and RRT planners with various underlying dynamics models, as well as the original dynamics models, as listed below:

- **Goal-Informed Simplified Model**: The dynamics model is built upon the simplified geometric model, which is informed by the goal state, and adaptively changes according to different goals and tasks, as detailed in Section 4.3.2.
- **Uninformed Simplified Model**: The dynamics model is built based on the process given in Section 4.3.3. These models are referred to as grid-based dynamics models, which are general simplifications that do not depend on specific goals or tasks. To evaluate performance across different resolutions, I test several grid models including **Grid Model** 2×2, **Grid Model** 3×3, and **Grid Model** 4×4.
- **Original Model**: This is the full, computationally expensive dynamics model, which serves as the baseline. It allows us to assess how much efficiency the simplified models offer and how well the planned trajectories from the simplified models compare to those based on the original dynamics model.

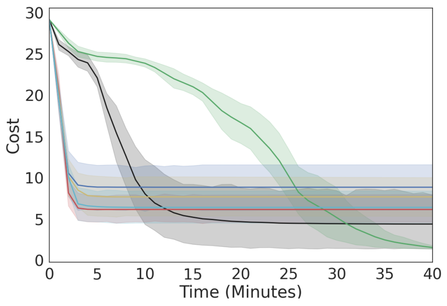
For model parameters such as mass and spring coefficients, I use the default values in SoftGym for both the original model and the reduced models (including our goal-informed simplified models and the grid simplified models). The default settings assign each particle a mass of 1.0, set the stretching coefficient to 0.8, the bending coefficient to 1.0, and the shear coefficient to 0.9. Besides, one crucial aspect to mention is that the reduced action space extracted from the goal is used for all these models. This has been proven more effective than other action spaces in Section 3.4.4 and ensures that the comparison between different models is fair by keeping the action space consistent across different tests.

4.5.2 Results

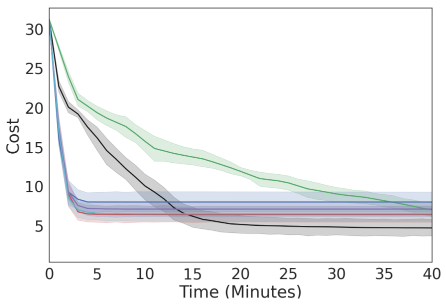
For all cloth and t-shirt folding tasks, I conduct motion planning experiments using both CEM and RRT. Each experiment is repeated ten times, utilizing both the proposed methods and baseline methods for comparison.

The results of motion planning based on CEM are presented in Figure 4.6. To enhance the planning efficiency, I reduce the population size of CEM to 100, while maintaining a larger population size for the *Original Model* baseline for a more consistent comparison.

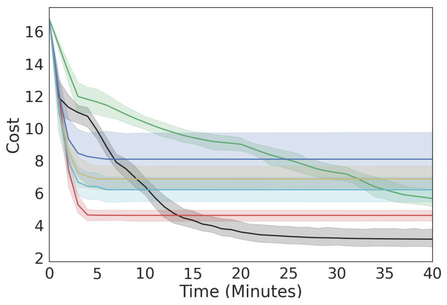
From the results, it is observed that the costs of trajectories planned using simplified models drop significantly faster than those planned with the original dynamics model, validating the effectiveness of simplified models in boosting efficiency for motion planning. However, planning based on the *Original Model* still achieves the best cost across all models, indicating that some degree of optimality is sacrificed when using simplified models. For CEM specifically, the population size also influences planning efficiency—while a smaller population size helps to plan faster, a larger population size yields trajectories with lower costs if given enough time (which is shown in Section 3.4.4). Among the simplified models, the results show that planning with the *Goal-Informed Simplified Model* converges faster and leads to better trajectories across all



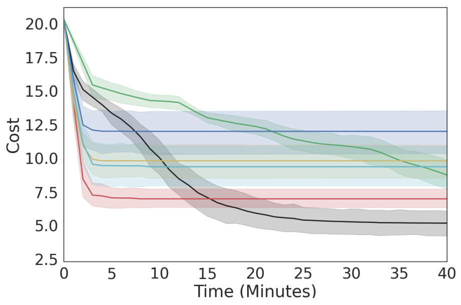
(a) Cloth diagonal folding



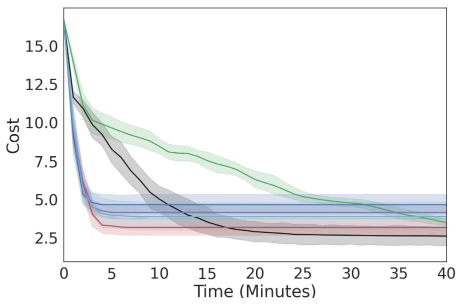
(b) Cloth side folding



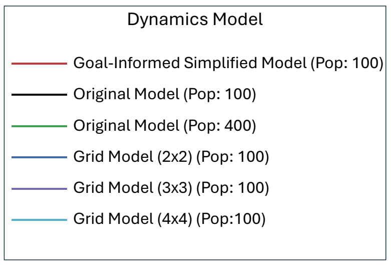
(c) Cloth reflective folding



(d) Cloth underneath folding



(e) T-shirt side folding



(f)

Figure 4.6: Planner (CEM) performance over time for different tasks using different dynamics models.

cloth and t-shirt manipulation tasks. This demonstrates that utilizing information from the goal allows for building more informative models, enabling the planner to find more effective trajectories more efficiently. In comparison, *Uninformed Simplified Model* converges more slowly than the goal-informed simplified models, and as the resolution of the grid model increases, the cost of the planned trajectory gradually decreases, highlighting the trade-off between grid resolution and planning performance.

In Figure 4.7, the cost versus time results for different dynamics models using RRT as the underlying planning algorithm are presented.

From the results, all models find trajectories with similar costs for the diagonal folding task, but planning with simplified models converges faster. For other tasks, the *Goal-Informed Simplified Model* converges the fastest and even reaches better costs than the original model. This further verifies that extracting more information from the goal can help build more effective dynamics models. In contrast, for the uninformed grid models, *Grid Model 2×2* consistently finds the worst trajectories due to its resolution being too low to accurately capture the behavior of the original dynamics. As the resolution increases, *Grid Model 3×3* and *Grid Model 4×4* generate better trajectories, but they still do not match the performance of the *Goal-Informed Simplified Model*. This highlights the superiority of goal-informed models over uninformed ones in terms of planning efficiency and trajectory quality.

4.6 Real World Experiments

In this section, I conduct experiments in the real world to verify the effectiveness of the proposed closed-loop deformable manipulation framework.

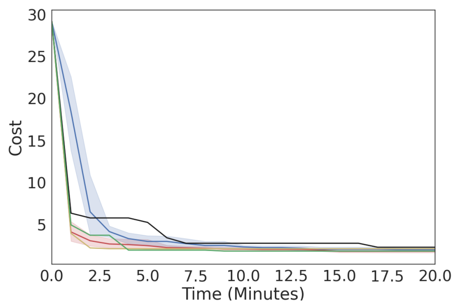
Specifically, in Section 4.6.1, I conduct experiments on the perception system, to demonstrate the ability of the tracking system to consistently monitor the shape of the deformable object during manipulation. The same experiments are conducted using both the original CDCPD and the improved CDCPD, to show the effectiveness of the modifications proposed in Section 4.4.

Next, in Section 4.6.2, the entire framework is tested by performing cloth folding tasks with the Franka Panda robot and the Realsense depth camera. These tasks are designed to demonstrate the enhanced robustness of the closed-loop execution framework compared to fully open-loop execution, particularly in handling the complexities of deformable object manipulation.

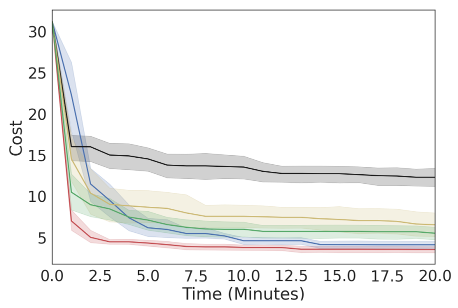
4.6.1 Cloth Shape Tracking

To verify the performance of the perception system in tracking the shape of a cloth during manipulation and to assess the effectiveness of the improvements proposed in Section 4.4, I conduct cloth shape tracking experiments using a Realsense D455 depth camera with a piece of square cloth being folded manually by a human.

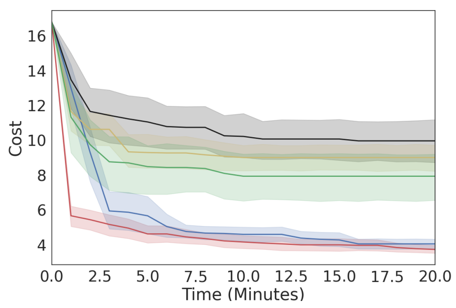
The process begins by isolating the point cloud of the cloth from the depth camera input, masking out the background using color filtering to focus exclusively on the cloth. This cloth-only point cloud serves as the input to the tracking algorithm. A human operator then manually fold the cloth, which is initially flattened on a table, in either diagonal or side directions while the tracking algorithm monitors the deformation of the cloth in real-time. Simultaneously, the tracked state of the cloth is reconstructed in a simulated environment of SoftGym to visualize the performance of the tracking algorithm more accurately.



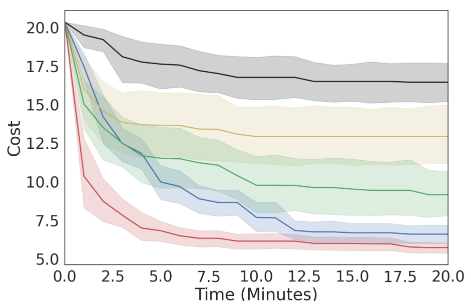
(a) Cloth diagonal folding



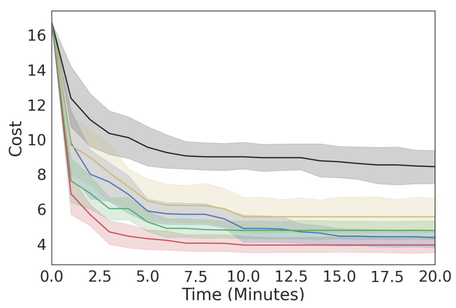
(b) Cloth side folding



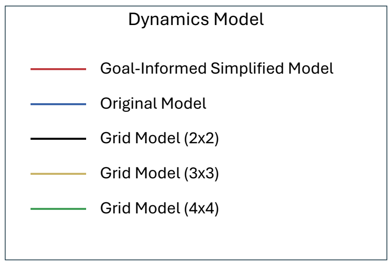
(c) Cloth reflective folding



(d) Cloth underneath folding



(e) T-shirt side folding



(f)

Figure 4.7: Planner (RRT) performance over time for different tasks using different dynamics models.

Figure 4.8(a) and Figure 4.9(a) show snapshots of the original CDCPD algorithm tracking the shape of the cloth as it is being folded diagonally and sideways, respectively. While the original CDCPD algorithm is able to successfully track the shape of the cloth when it is in a flattened state, the tracking degrades when the cloth is being folded. The main issue is self-occlusion, which becomes severe during folding, limiting the depth camera to capturing only surface point clouds. This causes the original CDCPD to struggle in distinguishing different layers of the cloth. Consequently, the algorithm reallocates all tracking points to the visible surface, compressing them into a smaller area rather than accurately tracking the layers as they fold.

This highlights the limitations of the original CDCPD algorithm in handling complex deformations like folding, especially when self-occlusion is present.

In contrast, the improved CDCPD, utilizing combined point clouds and selective point updates, successfully tracks the shape of the cloth during both diagonal and side folding tasks, as illustrated in Fig 4.8(b) and Figure 4.9(b). The reconstructed cloth in simulation clearly shows that it has distinct layers, reflecting the importance of thickened point cloud based on explicit reasoning of occlusion. Besides, only parts the tracking points are updated during the manipulation, which validates that the proposed improvements in Section 4.4 are effective.

4.6.2 Cloth Folding with Closed-Loop Execution

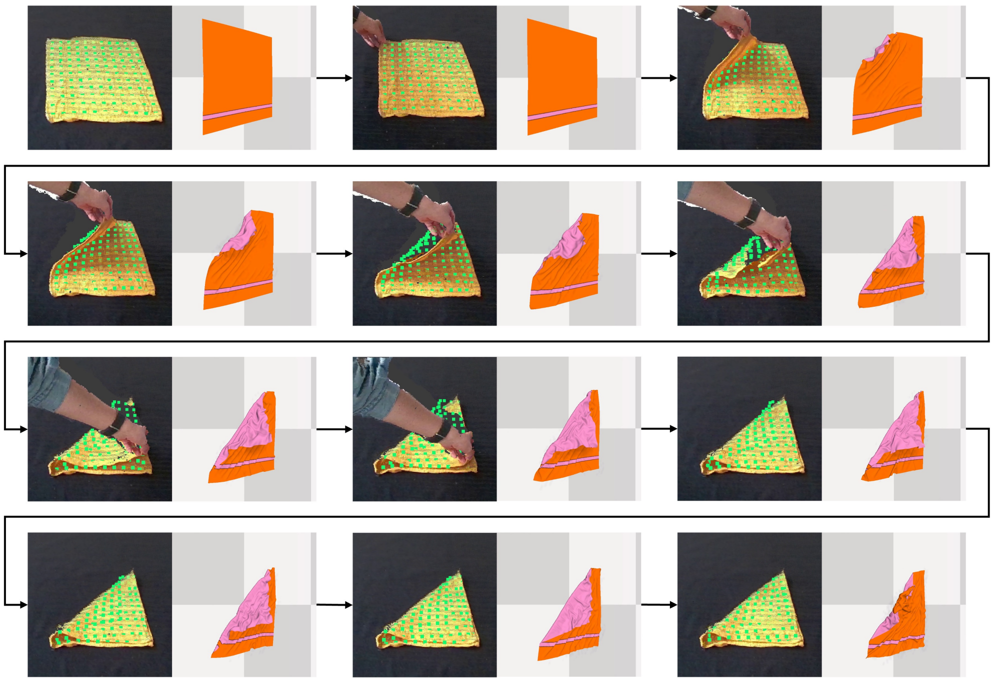
In the real-world experiments described in the previous chapter, the planned trajectory is executed in an open-loop manner, leading to low success rates due to discrepancies between the simulated and actual dynamics of the deformable objects. In this section, I conduct the same cloth folding tasks, but in a closed-loop manner, with the improved planning and perception methods. The aim is to test both the effectiveness of the planned trajectories based on the simplified dynamics model and the robustness of executing these plans with real-time perception feedback integrated within the execution loop.

Overview of the Real-World Experiment Setups

In this section, a closed-loop robotic experiment system is established, integrating the previously proposed motion planning and shape-tracking methods. The basic setup, as depicted in Figure 4.10, includes a Franka Panda robot mounted on a stationary table and a flattened square cloth (30 cm $\times$ 30 cm) randomly positioned in front of it, with various initial poses. The system utilizes a workstation, identical to the one used in simulation experiments, to conduct motion planning. Cloth shape tracking is performed using a Realsense D455 depth camera, which captures the point cloud of the scene. Additionally, an Optitrack system is employed to acquire the poses of both the robot and the depth camera, facilitating the calibration of the transformation between their respective coordinate systems. All subsystems are coordinated using the Robot Operating System (ROS) [135], ensuring synchronized operations that are essential for the successful execution of closed-loop control tasks.

Results

In this section, real-world cloth folding experiments are conducted using different planners with a simplified dynamics model. Each experiment is repeated five times to assess the efficacy and reliability of the framework. A summary of these experiments is provided in Table 4.1, in which


(a) Tracking with original CDCPD.

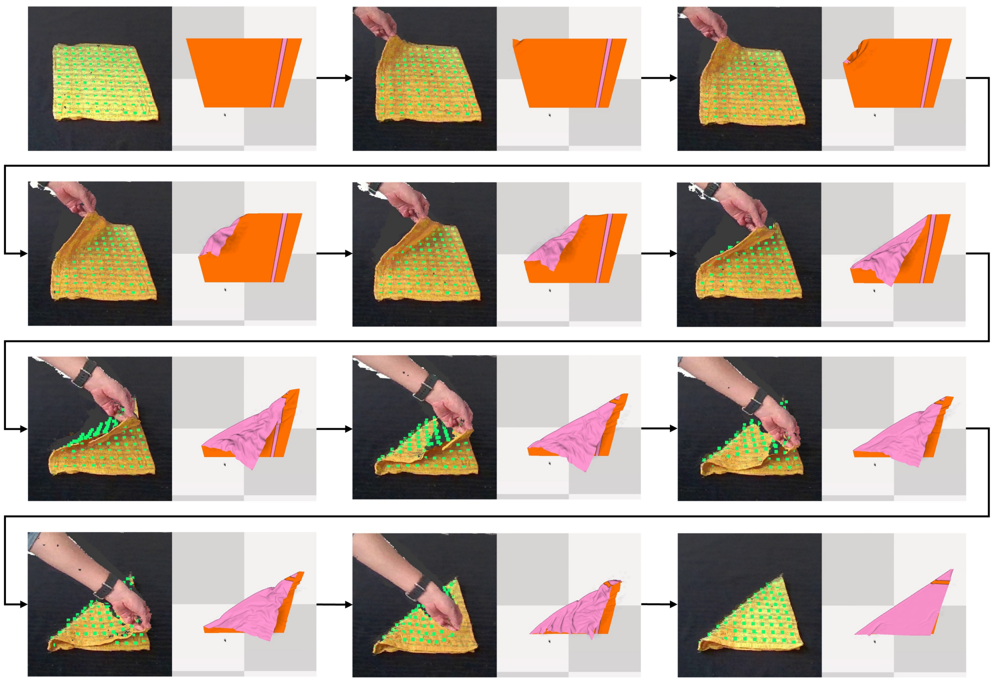
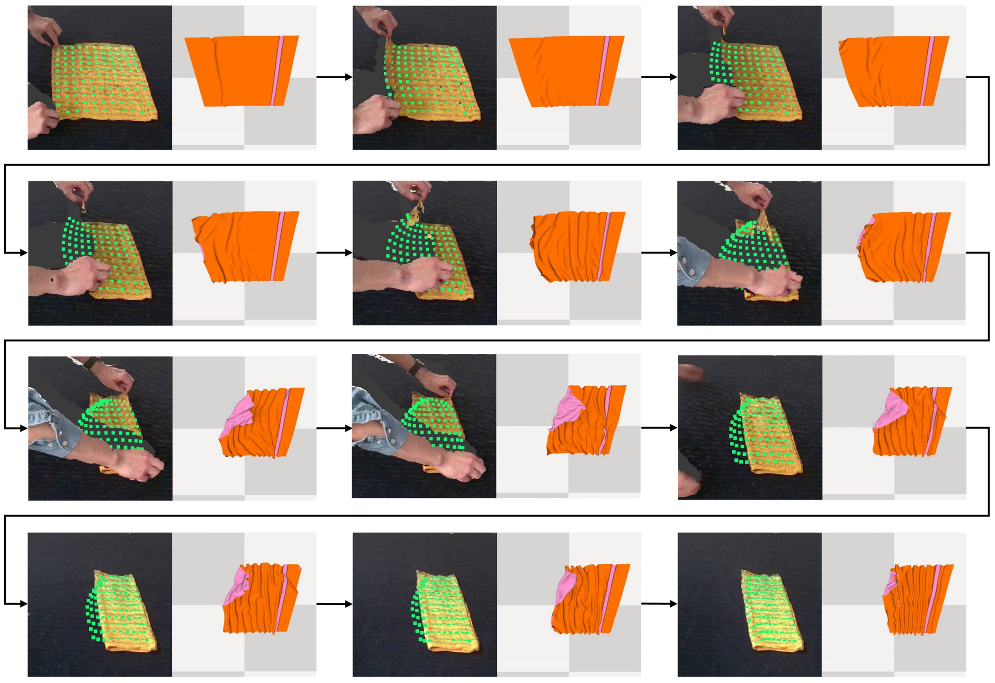

(b) Tracking with improved CDCPD.

Figure 4.8: Cloth shape tracking when it is being folded diagonally. In each frame, the left image displays the raw point cloud data captured by the depth camera (in yellow), with the tracked result shown in green dots; the right image illustrates the reconstructed cloth in simulation.


(a) Tracking with original CDCPD.

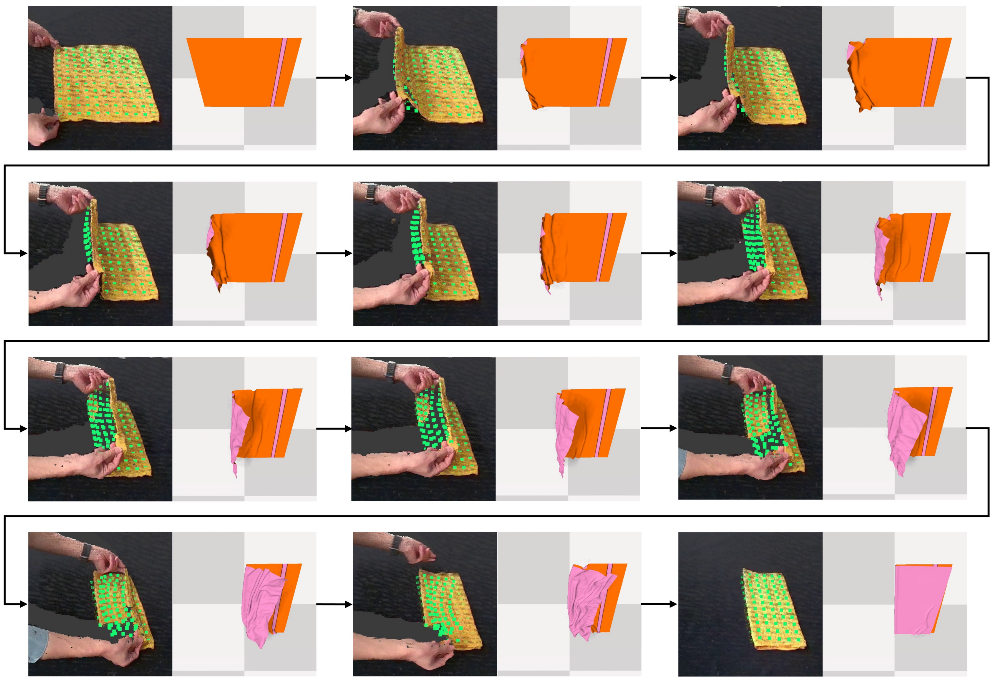

(b) Tracking with improved CDCPD.

Figure 4.9: Cloth shape tracking when it is being folded sideways. In each frame, the left image displays the raw point cloud data in yellow alongside the tracked result in green; the right image illustrates the reconstructed cloth in simulation.

I show the tasks performed, the planner used, the time spent on motion planning, the success rate, and the average number of re-grasps required.

For cloth diagonal folding, CEM spends an average 2.07 minutes to generate a trajectory, achieving a 100% success rate with an average of 4.8 re-grasps. An experiment is defined as successful if the robot finishes executing all the actions in the planned trajectory and brings the cloth to a reasonable goal determined by the user. RRT also achieves a 100% success rate but requires significantly more re-grasps, approximately 67% more than CEM, though it outputs a trajectory in just one minute.

For cloth side folding, CEM requires slightly more time than for diagonal folding, which is on average 3 minutes per plan. The success rate drops to 60% due to the complexity of alternating grasps between different corners of the cloth. As the robot manipulates one corner, the uncontrolled parts of the cloth may move unpredictably. This is also reflected by the average

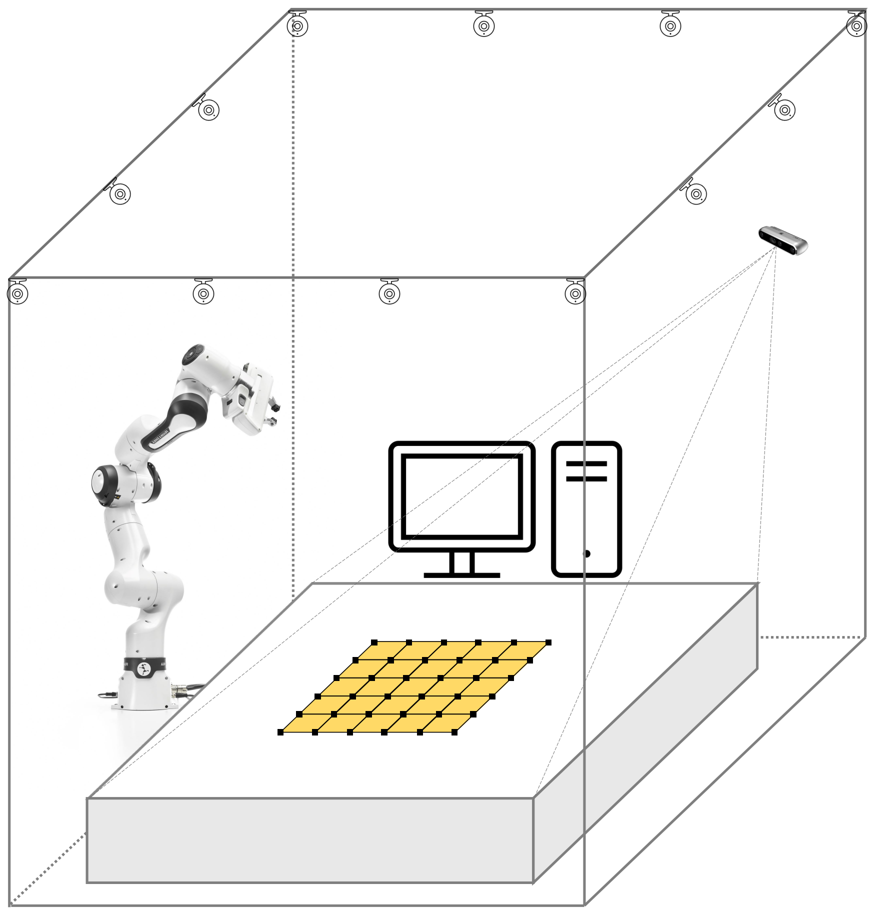


Figure 4.10: Real-world experiment setup, in which we have an OptiTrack system with 12 infrared cameras, a Realsense 455 depth camera, a Franka Emika Panda robot, and a server with a powerful GPU.

Table 4.1: Real-world experiment results based on single simplified model

Task	Planner	Planning time (min)	Success Rate	Average re-grasps
Diagonal Folding	CEM	2.07	5/5	4.80
	RRT	1.00	5/5	8.00
Side Folding	CEM	3.01	3/5	6.40
	RRT	1.00	0/5	18.60

number of re-grasps, which increases by 33.3% than the diagonal folding task. RRT fails all 5 experiments on the cloth side folding tasks, which is due to the high number of re-grasps in the planned motion, an average of 18.6 times. This makes the experiment more vulnerable to robot execution errors and perception failures.

Common causes of failure across these real-world experiments include: 1) failure of the tracking system due to severe occlusion by the robot arm; 2) the gripper grasping an incorrect position due to tracking errors; 3) the gripper grasping multiple layers of the cloth; and 4) the picking position being out of the robot's workspace, causing joint limit violations.

Images of the real robot folding the cloth diagonally and sideways, using the simplified dynamics model and CEM as the underlying motion planner, are shown in Figure 4.11 and Figure 4.12. The Panda robot successfully achieves both tasks. Within each task, the gripper can detect grasping failures and re-grasp the cloth showing that the proposed controlling scheme for executing the plan is effective and the planned trajectory in simulation can be executed successfully to achieve the desired cloth folding. Additionally, when the real-world behavior of the cloth deviates from the simulation predictions, the tracking system enables the robot to adapt its actions accordingly, demonstrating the robustness and adaptability of the closed-loop execution framework.

4.7 Conclusion

In this section, I propose accelerating motion planning by utilizing simplified dynamics models, including both goal-informed and uninformed approaches. The goal-informed simplification is built based on the simplified geometric model given the goal state, while the uninformed simplification reduces the resolution of the original model uniformly to a set of grid-based models. Experiments show that both simplified models significantly reduce the time cost of motion planning, albeit with some loss in precision. The goal-informed method, however, performs much better than the uninformed approach, demonstrating that goal-specific information can help build better models and thus improve performance of motion planning. Additionally, I improve the CDCPD tracking algorithm for monitoring the shape of the deformable object during manipulation. These improvements include explicitly reasoning about self-occlusion, combining

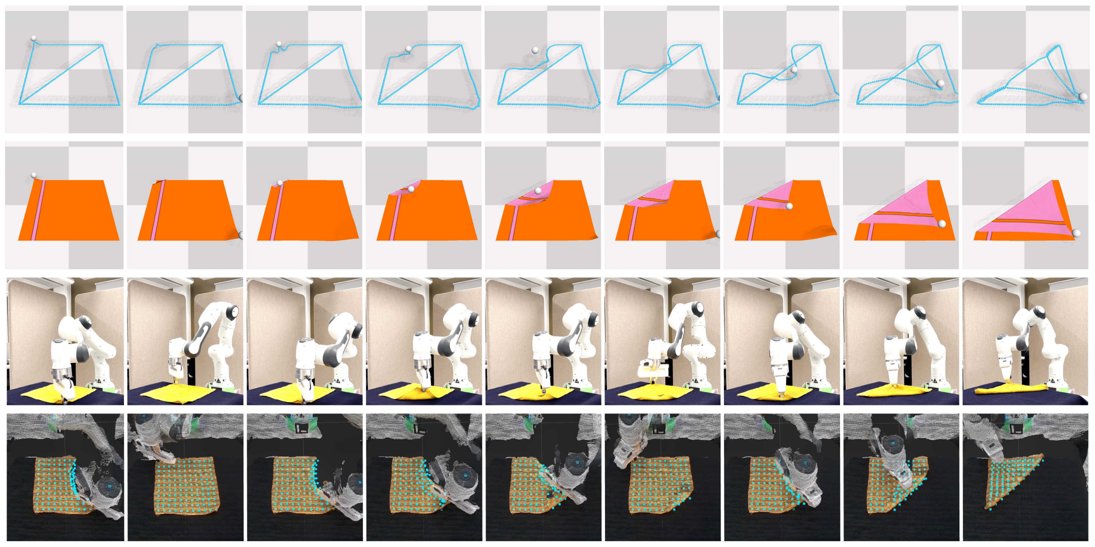


Figure 4.11: Real-world experiment for cloth diagonal folding with simplified models. The first row displays the planned trajectory on the simplified model. The second row shows how this trajectory is rolled out on the original model. The third row captures the robot executing this trajectory, while the fourth row presents the tracking results.

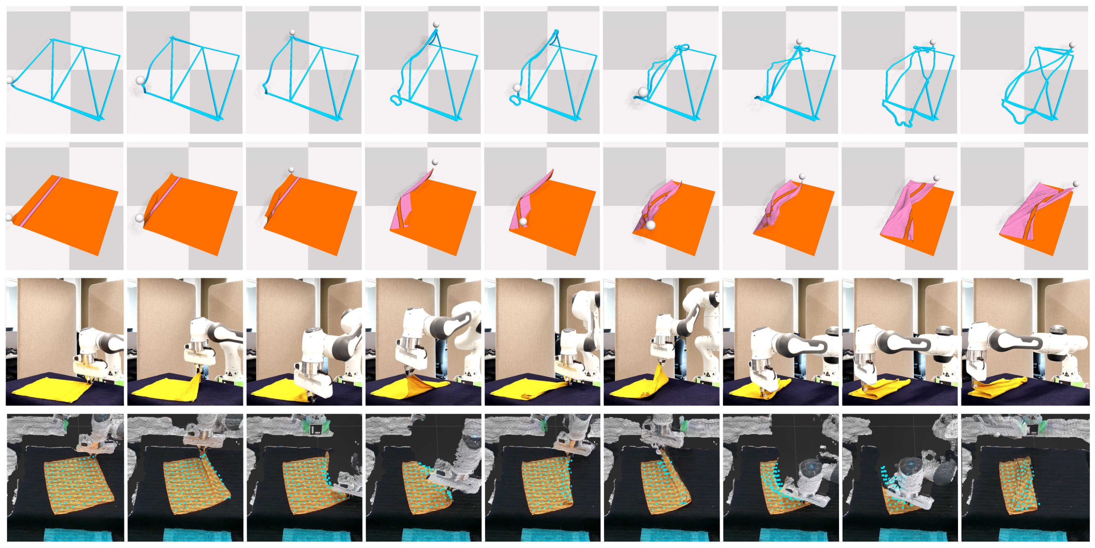


Figure 4.12: Real-world experiment for cloth side folding with simplified models. The first row displays the planned trajectory on the simplified model, the second row for rolling out on the original model, the third for the robot executing this trajectory, and the fourth row for the tracking results.

consecutive point clouds, and selectively updating tracking points. Experiments confirm that these modifications lead to a significant improvement in tracking precision compared to the original method. Based on the methods proposed above, I establish a closed-loop real-world experiment system incorporating planning, control, and perception. This system demonstrates significant improvements in both performance and robustness for executing deformable object manipulation tasks in real-world scenarios.

Chapter 5

Iterative Model Simplification & Motion Planning for Deformable Object Manipulation

5.1 Introduction

In the previous chapter, I propose planning based on simplified dynamics models for deformable object manipulation, which greatly boosts efficiency compared to using the computationally expensive high-fidelity model. While this approach significantly reduces time costs, it sacrifices precision for efficiency. As a result, the trajectories planned using these simplified dynamics models may not perform well on the original dynamics model due to significant differences between the two. This underscores the need to enhance the simplified model, maintaining simplicity while improving fidelity. Therefore, this chapter aims to further refine this approach by developing a framework that can balance efficiency and optimality in trajectory generation.

The saying from statistician George Box, "All models are wrong, but some are useful," captures the essence of using model approximations in computationally expensive tasks. Mitrano et al. [136]propose a strategy to make approximate models more effective by learning where they are accurate for specific tasks and confining the planner to search only within this constrained space where the approximate model behaves similarly to the real model. This approach leverages the efficiency of approximate models while mitigating their inaccuracies. McConachie et al. [115] take a different approach by using multiple approximate models at different stages of manipulation, acknowledging that some models may offer higher precision for certain types of deformations but not others. They treat the selection of the most suitable model as a multi-armed bandit problem, allowing the system to adaptively choose the best model at each stage during manipulation.

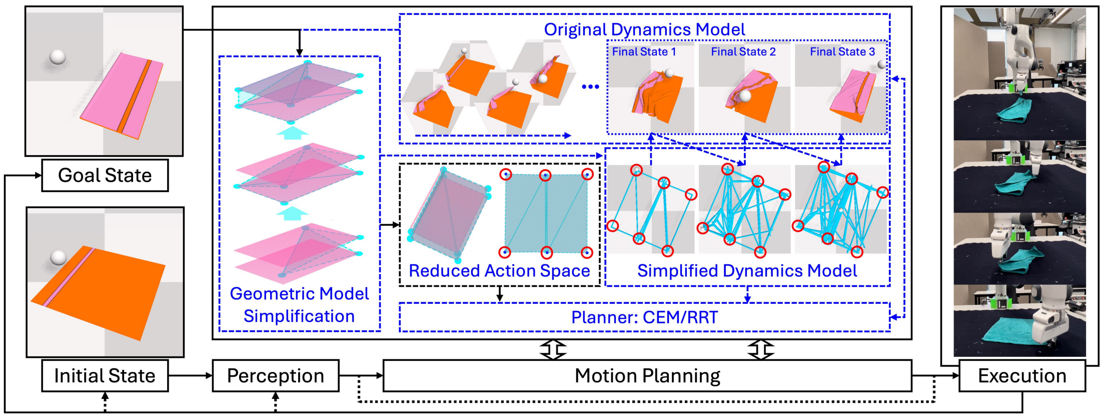


Figure 5.1: Overview of the iterative model simplification and motion planning framework with robot execution in the real world. Initially, a simplified geometric model is identified and used to extract key picking points in the reduced action space. A simplified dynamics model is then built and utilized to plan a trajectory in a significantly shorter time. The trajectory is executed on the original model, and if the goal is not reached, the loop iterates, refining the simplified model until a satisfactory trajectory is found. Once a valid trajectory is identified, it is executed on the robot, with the perception system continuously tracking deformation during manipulation.

However, these methods involve manually designed or selected models and require extensive offline training. They also lack consistency across different models. To address these issues, I propose an iterative framework to refine the simplified dynamics model continuously, reducing the behavioral gap between it and the original model. If a trajectory planned with a simplified model fails to achieve the goal state, I use the final state achieved to enhance the simplified model. This process, illustrated in the Simplified Dynamics Model block of Figure 5.1, involves iteratively extracting features from the achieved final state to progressively refine the model. Besides, the previously planned trajectory serves as a good warm start for the new round of planning with a finer simplified model. By utilizing this initial trajectory as a starting point, the motion planner can potentially reduce the time needed for further optimization and exploration. This iterative process not only improves the precision of the simplified model but also enhances planning efficiency by building on the knowledge gained from earlier iterations. Consequently, the system can balance between efficiency and optimality more effectively as the model becomes progressively more accurate. By the third iteration in Figure 5.1, a trajectory is successfully found that brings the original model to a satisfactory final state, demonstrating the effectiveness of this iterative approach in improving model utility and accuracy.

This framework is independent of the specific methods used for model simplification. Instead of relying solely on the goal-informed model simplification, users can opt for an uninformed method as well. For instance, using the grid model proposed in Section 4.3.3, one can begin by planning with a low-resolution grid model and progressively move toward higher-resolution grid models. The trajectories planned using the lower-resolution grid models can serve as a warm start for motion planning on higher-resolution models, improving efficiency in subsequent iterations. This iterative process continues until a satisfactory trajectory is found or further improvements become negligible with higher-resolution models.

To verify the effectiveness of the proposed methods, in Section 5.4, I conduct simulation experiments on the iterative model simplification and motion planning framework for deformable object manipulation. Both goal-informed models and uninformed grid models are tested within this framework. Additionally, a variant approach that uses the simplified model as an efficient warm start, followed by fine-tuning the planned trajectory on the original model, is evaluated. In Section 5.5, based on the closed-loop real robot experiment platform established in Section 4.6.2, I conduct experiments with the Franka Panda robot folding cloth diagonally and sideways to examine the performance of the iterative planning framework.

In summary, the contributions of this chapter include:

– An iterative model simplification & motion planning framework that generates manipulation plans faster, effectively balancing computational cost and trajectory optimality.

– An implementation of the iterative framework with configuration-informed model simplification, where the simplification process is conditioned on specific task goals to enhance efficiency.

– An implementation of the iterative framework with non-informed model simplification, which applies general simplifications independent of task-specific goals.

– An extensive evaluation of the proposed model simplification and motion planning framework in simulation, covering a range of representative tasks involving regular and irregular shaped cloth, using both sampling-based and optimization-based motion planning methods.

The organization of this chapter is as follows. In Section 5.2, I reiterate the problem formulation of the motion planning problem for deformable object manipulation. In Section 5.3, I introduce the framework of iterative model simplification and motion planning for deformable object manipulation, detailing both the configuration-informed and non-informed implementations. In Section 5.4, I present the results from simulation experiments, comparing the performance of the iterative framework with different implementations and various baselines. In Section 5.5, I demonstrate the effectiveness of the iterative planning framework in the real world for cloth folding tasks. In Section 5.6, I summarize the advantages and limitations of the proposed methods in this chapter.

5.2 Problem Formulation

In this chapter, I address the same motion planning problem for deformable object manipulation as defined in Section 3.2, encompassing the state space, action space, and cost function.

Compared to the problem solved in Chapter 3, where the underlying dynamics model is the original high-fidelity model, and the problem solved in Chapter 4, where a single simplified dynamics model is used, in this chapter, I introduce a sequence of progressively simplified dynamics models along with the original dynamics model as the underlying state transition function in the motion planning problem. Solving the planning problem with the simplified dynamics models serves as an inner loop for efficiently solving the planning problem with the original dynamics model.

5.3 Iterative Model Simplification & Motion Planing

In this section, building on the geometric model simplification and action space reduction methods proposed in Section 3.3, and the dynamics model simplification method from Section 4.3, I present an iterative model simplification and motion planning framework for deformable object manipulation that progressively improves the trajectory by using finer and finer simplified models.

Specifically, in Section 5.3.1, I provide an overview of the proposed iterative framework. In Section 5.3.2, I describe the approach for combining and refining the simplified models. Then in Section 5.3.3, I detail the implementation of the framework using configuration-informed model simplification. Lastly, in Section 5.3.4, I introduce the uninformed implementation of the iterative model simplification.

5.3.1 Overview of the framework

The overall process of the proposed iterative model simplification for motion planning is illustrated in Algorithm 5.

This framework starts by computing the simplified geometric model $\hat{\mathcal{G}}_S$ based on the goal state of the original model (Line 2), and then uses this simplified geometric model to reduce the set of picking points $V_O^{\mathcal{A}}$ on the original model (Line 3). Next, a simplified dynamics model $\mathcal{D}_S$ is constructed using the simplified geometric model $\hat{\mathcal{G}}_S$ and the original dynamics model $\mathcal{D}_O$ (Line 5). The initial state ξ_O^0, the goal state ξ_O^G, and the reduced picking points $V_O^{\mathcal{A}}$ are then mapped to the simplified model based on the correspondence between the two models (Line 6). A planner is subsequently invoked to generate a trajectory that moves the simplified model from the initial state toward the goal within the reduced action space (Line 7). The planned trajectory is then rolled out on the original dynamics model (Line 8). During each loop, the original dynamics model, which is computationally expensive to simulate, is called only once, while numerous trajectory rollouts within the planner use coarse but computationally cheap models. If a satisfying trajectory is found or marginal improvement is observed for the original dynamics model, the process terminates (Line 10); otherwise, the geometric model simplification is invoked again to extract more details from the actually achieved final state (Line 11). The new simplified geometric model $\hat{\mathcal{G}}_S'$ is then combined with the previously simplified model $\hat{\mathcal{G}}_S$

Algorithm 5: Iterative Model Simplification & Motion Planning

```
   Input: $\mathcal{G}_O$, $\mathcal{D}_O$, $\xi_O^0$, $\xi_O^G$
   Output: $\tau$
1  $\tau \leftarrow None$;
2  $\hat{\mathcal{G}}_S, \hat{\xi}_S \leftarrow$ Simplify_Geometry($\mathcal{G}_O, \xi_O^G$);
3  $V_O^{\mathcal{A}} \leftarrow$ Reduce_Action_Space($\mathcal{G}_O, \xi_O^G$);
4  do
5      $\mathcal{D}_S \leftarrow$ Simplify_Dynamics($\mathcal{D}_O, \hat{\mathcal{G}}_S$);
6      $\xi_S^0, \xi_S^G, V_S^{\mathcal{A}} \leftarrow$ Map($\mathcal{D}_O, \mathcal{D}_S, \xi_O^0, \xi_O^G, V_O^{\mathcal{A}}$);
7      $\tau \leftarrow$ Planner($\mathcal{D}_S, V_S^{\mathcal{A}}, \xi_S^0, \xi_S^G, \tau$);
8      $\xi_O \leftarrow$ Rollout($\mathcal{D}_O, V_O^{\mathcal{A}}, \tau$);
9      if $\xi_O^G$ reached or convergent then
10         break;
11     $\hat{\mathcal{G}}'_S, \hat{\xi}'_S \leftarrow$ Simplify_Geometry($\mathcal{G}_O, \xi_O$);
12     $\hat{\mathcal{G}}_S \leftarrow$ Combine_Geometry($\hat{\mathcal{G}}'_S, \hat{\mathcal{G}}_S$);
13 while True;
```

to ensure that no information is lost from the previous step (Line 12). The trajectory planned from the previous simplified model is used as a warm start for the new iteration of motion planning, based on the updated simplified model (Line 7).

This framework is general and does not depend on specific algorithms for action space reduction, dynamics model simplification, or the underlying planning methods. Since it has been validated in Chapter 3 that the goal-conditioned action space performs at least as well as the best action spaces, I retain it as a common foundation for the framework. The primary focus is on different approaches to constructing and refining the dynamics model (Section 5.3.2), as detailed in Section 5.3.3, where models are progressively built in a configuration-informed way, and in Section 5.3.4, where the models are progressively refined based on grid resolutions. The choice of planners can be left to user preference.

5.3.2 Model Refinement and Combination

A key component of the iterative model simplification framework is model refinement and combination. When the planned trajectory leads the original model to a final state that is not sufficiently close to the goal, I reapply the geometric model simplification process, this time using the actually achieved final state as input (line 11 in Alg. 5).

To ensure consistency in the simplified model throughout the planning process, I retain the initially simplified geometric model while integrating additional details from the newly

Algorithm 6: Geometric Model Combination (`Combine_Geometry`)

```
Input: $\hat{\mathcal{G}}'_S(\hat{V}'_S, \hat{E}'_S, \hat{L}'_S)$, $\hat{\mathcal{G}}_S(\hat{V}_S, \hat{E}_S, \hat{L}_S)$
Output: Updated $\hat{\mathcal{G}}_S$
1  for $v'_S \in \hat{V}'_S$ do
2      if $v'_S \notin \hat{V}_S$ then
3          $\hat{V}_S \leftarrow \hat{V}_S \cup \{v'_S\}$;
4      end
5  end
6  for $e'_S(u, v) \in \hat{E}'_S$ do
7      if $u \in \hat{V}_S$ && $v \in \hat{V}_S$ then
8          $\hat{E}_S \leftarrow \hat{E}_S \cup \{e'_S\}$;
9      end
10 end
11 for $(u, v) \in \hat{V}_S \times \hat{V}_S$ do
12     if $(u, v) \notin \hat{E}_S$ && not Intersect$(u, v, \hat{E}_S)$ then
13         $\hat{E}_S \leftarrow \hat{E}_S \cup \{(u, v)\}$;
14     end
15 end
```

simplified model, as shown on line 12 (`Combine_Geometry`) in Alg. 5. The process follows three main steps, which correspond to the key operations in Algorithm 6 explained below.

Particle Integration (Lines 1–3): I first identify and incorporate new particles from the newly simplified geometric model, $\hat{\mathcal{G}}'_S$, that do not exist in the initially simplified model, $\hat{\mathcal{G}}_S$. This ensures that the refined model includes all relevant structural elements while retaining the original simplified representation.

Edge Retention (Lines 4-6): Once the particle set is updated, I retain edges from $\hat{\mathcal{G}}'_S$ where both end particles exist in the combined particle set. This step ensures that existing connections between incorporated particles are preserved, maintaining structural consistency.

Edge Completion and Consistency Check (Lines 7-9): After merging edges from both models, the new graph may contain incomplete triangular structures where necessary edges are missing. To address this, I add missing edges while ensuring that newly introduced edges do not intersect with existing ones. This step preserves the geometric integrity of the simplified model.

Through these steps, the refined geometric model retains all information from the initial model while incorporating additional structural details from the newly simplified version. This enhances the accuracy of subsequent planning iterations by better representing the actually achieved state.

5.3.3 Configuration Informed Iterative Model Simplification

The configuration of the goal usually conserves relevant information necessary to achieve it. Therefore, one approach to constructing the dynamics model is by building a simplified dynamics model based on the simplified geometric model derived from the goal. As formulated in Section 4.3.2, the function `Simplify_Dynamics` can be implemented in a configuration-informed way. If this simplified model does not perform well for the original model, the actual final configuration achieved during execution can provide insights for improving the dynamics model. More details and information can be progressively extracted from these actually achieved configurations, allowing the model to be refined iteratively.

An example is shown in the Geometric Model Simplification block of Figure 5.1, where an original model with a large number of particles at the folded goal state, is reduced to a simplified geometric model consisting of six triangles and eight nodes. The corresponding particles on the original model are then extracted as the potential picking points in the reduced action space (Reduced Action Space block of Figure 5.1). A simplified dynamics model is subsequently built based on the simplified geometric model, with particles and springs placed along the edges (first image in the Simplified Dynamics Model block of Figure 5.1). Furthermore, the reduced picking points on the original model are extracted accordingly on the simplified dynamics model, marked by red circles. Using the reduced action space and simplified dynamics model, a planner is invoked to generate a trajectory, which is then rolled out on the original model. As is shown in the Final State 1 of Figure 5.1, the initially planned trajectory does not bring the cloth to the goal state. Thus, a new simplified model is created by approximating the achieved final state, incorporating more details than the previous model. A new trajectory is then planned based on this updated simplified dynamics model, which improves a bit, but still not satisfying. The same model simplification and planning loop runs another iteration, bringing the original model closer to the goal and making it ready for execution on the real robot.

5.3.4 Uninformed Iterative Model Simplification

The framework also applies to uninformed model simplification methods. As described in Section 4.3.3, the original dynamics model can be approximated by uniformly reducing the resolution, such as using grid-based dynamics models depicted in Figure 4.4. One challenge with uninformed model simplification is that the resolution of the grid is predetermined by the user, which can significantly affect precision depending on the specific goals and tasks.

This limitation can be mitigated by the iterative simplification and motion planning framework. Initially, the lowest resolution grid model, as shown in Figure 4.4(a), is used to plan a trajectory. If this trajectory fails to perform well on the original dynamics model, the grid resolution is increased, as depicted in Figure 4.4(b), with the previously planned trajectory serving as a warm start for planning with the higher resolution grid model.

This iterative process continues, progressively refining the resolution of the grid model as shown in Figure 4.4, until the planner either finds a satisfactory trajectory for the original dynamics model or further resolution increases yield only marginal improvements.

5.4 Simulation Experiments

In this section, I conduct experiments with the proposed iterative model simplification and motion planning framework, aiming to answer the following questions:

– Can the iterative framework progressively improve the quality of the plan, and at what extra time cost?

– Which iterative model simplification approach is more efficient and effective: configuration-informed or non-informed?

– What are the limits of the iterative model simplification, and how can we balance optimizing for quality with prioritizing efficiency?

The tasks tested in this section are the same as those in Chapter 4, including cloth and t-shirt manipulation tasks as defined in Section 3.4.1.

The framework remains independent of the planning algorithm used. To provide a more comprehensive evaluation, both the optimization-based planning method CEM and the sampling-based method RRT are tested.

All simulations are performed using the SoftGym simulator, and the same workstation (Intel(R) Core(TM) i9-11900 CPU @2.50GHz, 32 GB of RAM, and Nvidia GeForce GTX 4090 GPU) as used in previous Chapters.

The following sections detail the implementation of the proposed method and baseline methods in Section 5.4.1, followed by the presentation of comparative results and an analysis of the findings in Section 5.4.2. Additionally, Section 5.4.3 presents a comparison between the learned dynamics model and the iteratively simplified models.

5.4.1 Implementation & Baselines

For the designed framework, the dynamics model and motion planner can be changed flexibly depending on various requirements. Specifically, for dynamics models, we have the configuration-informed simplified dynamics models, the original dynamics model, and the uninformed grid-based dynamics models of different resolutions and orientations. To ensure a fair comparison, all planners use the same action space, which is the `Reduced Action Space`. The details of the implementation of the proposed methods and baselines are listed below:

– **Simplified Model**: this is the implementation of the proposed framework, where the simplified dynamics model is iteratively improved and used for motion planning as in Alg. 5. To terminate, I use a small cost threshold (0.02) to determine whether the trajectory found is satisfactory. In addition to this threshold, the iterative method will also stop if it "converges" (i.e., the method stops if there are no improvements in the achieved cost for successive iterations (currently set to 5), even if the cost is not under the threshold).

- **Simplified & Original Models**: this is a variation of the configuration-informed framework, where the dynamics model is simplified only once. The planned trajectory, based on the simplified model, is used as a warm start and is continually optimized on the original dynamics model.

- **Grid Models**: for this baseline, the uninformed grid-based models are implemented in the iterative planning framework, with gradually increased granularity. For example, a set of grid-based dynamics models of various resolutions are shown in Figure 4.4.

- **Original Model**: in this baseline, the original high-fidelity but time-consuming dynamics model is used for motion planning.

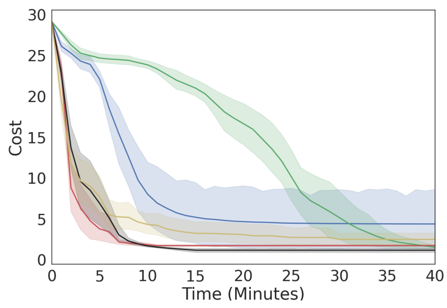
5.4.2 Compare with Non-Informed Models

For all cloth folding and t-shirt folding tasks, I run the motion planning experiments with both CEM and RRT separately. I repeat the experiment using the proposed methods and the baseline methods for ten times.

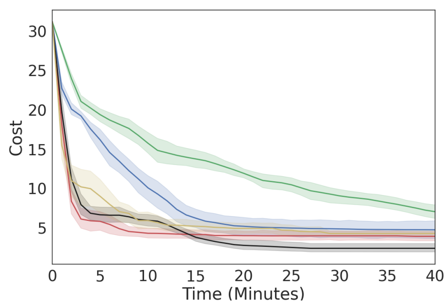
The results of motion planning based on CEM are shown in Figure 5.2. To speed up planning, I reduce the population size of CEM to 100, while keeping a large population-sized *Original Model* baseline for better comparison.

From the results, it is observed that for the *Original Model*, CEM with a smaller population size converges faster but results in a higher cost compared to CEM with a larger population for all cloth folding and t-shirt folding tasks. In contrast, *Simplified & Original Models* achieve the best cost across all tasks and converge more quickly than the *Original Model*, indicating that using the simplified model as a warm start helps guide the planner toward better trajectories more efficiently when optimizing on the original model. Additionally, *Simplified Models* converge the fastest among all model selections for all tasks and produce costs similar to or better than the *Original Model* with the same population size. *Grid Models* achieve acceptable costs for all tasks, but they are less efficient than the configuration-informed simplified dynamics model, demonstrating that informed dynamics model simplification outperforms universally simplified models in terms of efficiency.

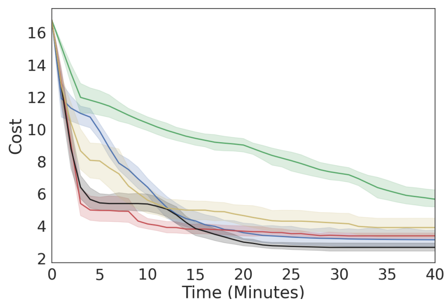
Example trajectories planned based on multiple configuration-informed simplified models for various tasks are shown in Figure 5.4, Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.9 respectively. Specifically, for the t-shirt side folding task, the first trajectory is planned using the most simplified dynamics model and brings the simplified model close to the goal. However, when this trajectory is executed on the original model, the left half of the t-shirt does not fully cover the right half, which demonstrates the discrepancy between the simplified and original models. After one more iteration, a new simplified model is created with a slight adjustment on the left



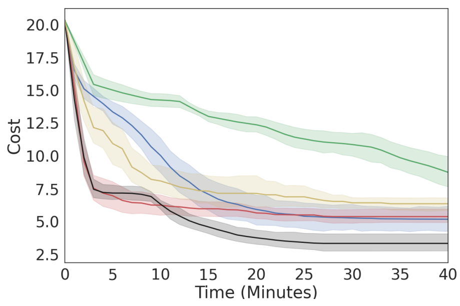
(a) Cloth diagonal folding



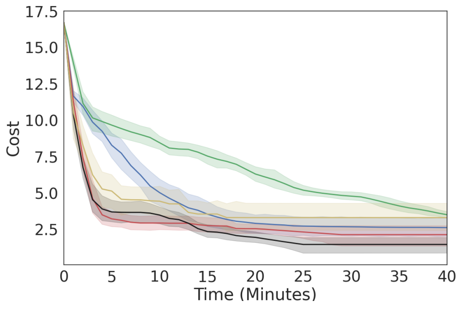
(b) Cloth side folding



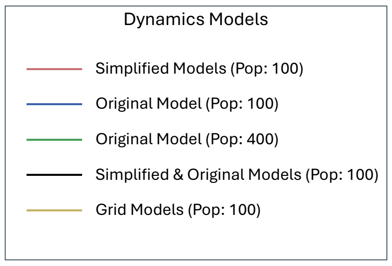
(c) Cloth reflective folding



(d) Cloth underneath folding



(e) T-shirt side folding

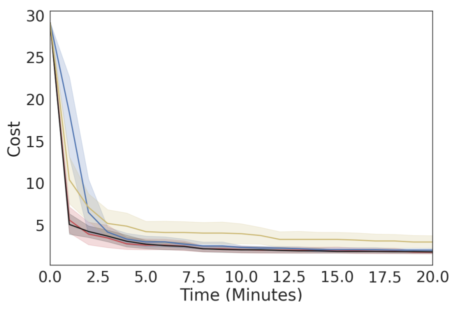


(f)

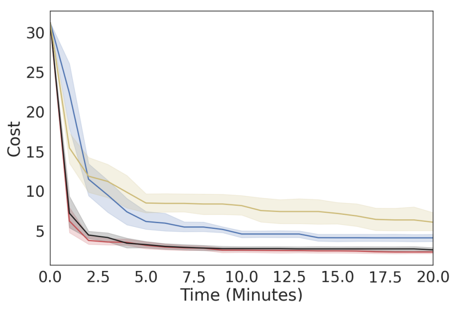
Figure 5.2: Planner (CEM) performance over time for different tasks using different dynamics models.

side. A new trajectory is planned based on this refined model, successfully manipulating the original model to the desired goal.

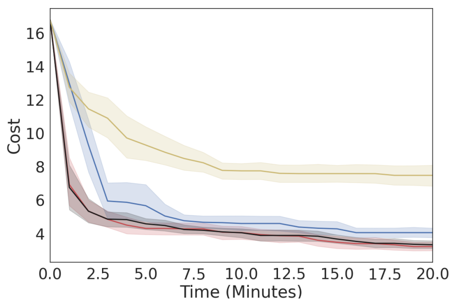
In Figure 5.3, I present the cost versus time results for different dynamics models using RRT as the underlying planning algorithm. One notable difference between RRT and CEM implementations is that OMPL does not support a warm start. Therefore, I use the models in a manner similar to model predictive control: the state achieved by the previous plan serves as the initial state for the next planning loop, with the process continuing iteratively until



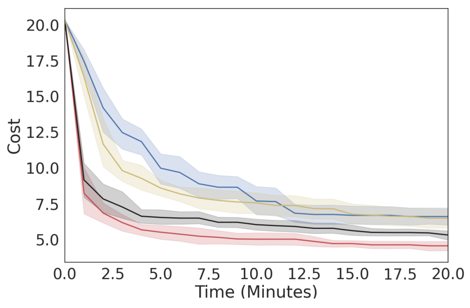
(a) Cloth diagonal folding



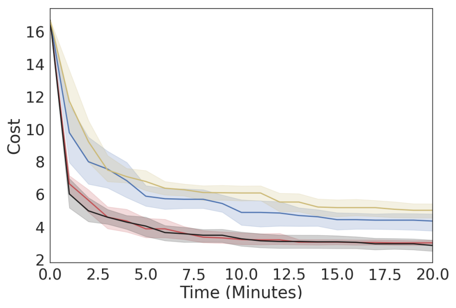
(b) Cloth side folding



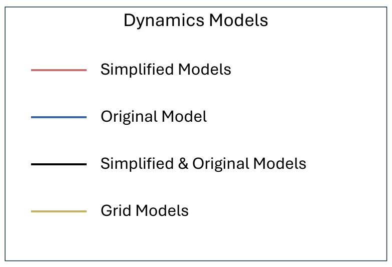
(c) Cloth reflective folding



(d) Cloth underneath folding

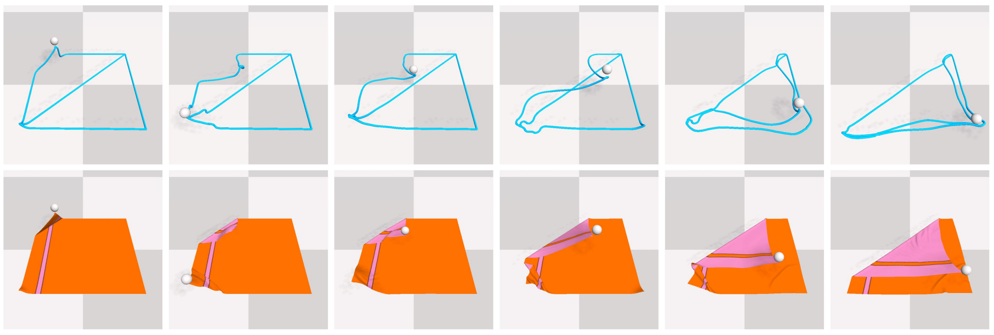


(e) T-shirt side folding

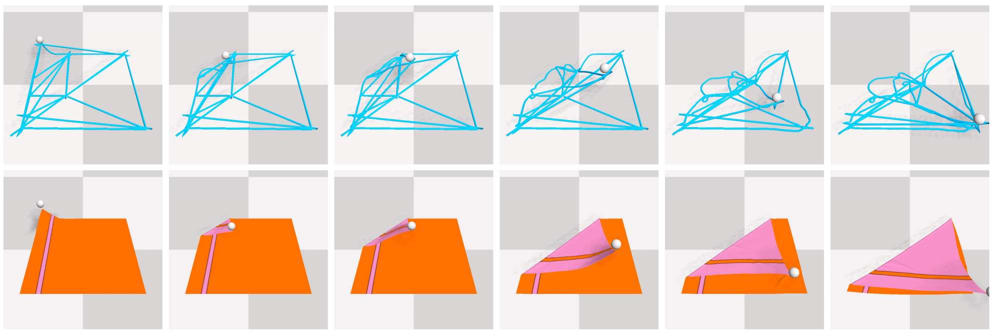


(f)

Figure 5.3: Planner (RRT) performance over time for different tasks using different dynamics models.

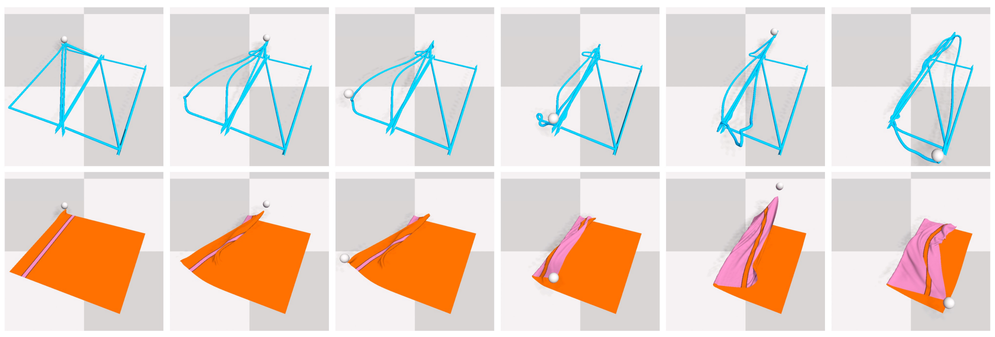


(a) First simplified model



(b) Second simplified model

Figure 5.4: Cloth diagonal folding with simplified models. In image (a), a trajectory is initially planned using the simplified model, depicted in the first row. This trajectory is then rolled out on the original model, shown in the second row, but it fails to achieve the desired goal. Consequently, a new round of model simplification is conducted based on the actual final state observed in the last image of the second row in (a). This results in a more detailed simplified model. Using this refined model, a new trajectory is successfully planned and executed, effectively bringing the cloth to the goal, as illustrated in image (b).


(a) First simplified model

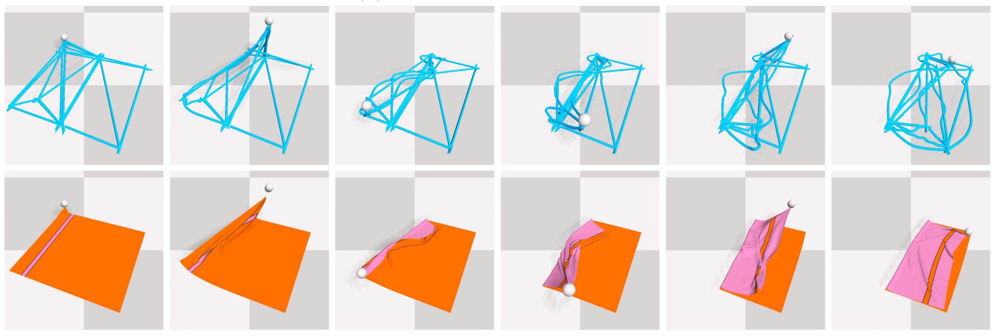
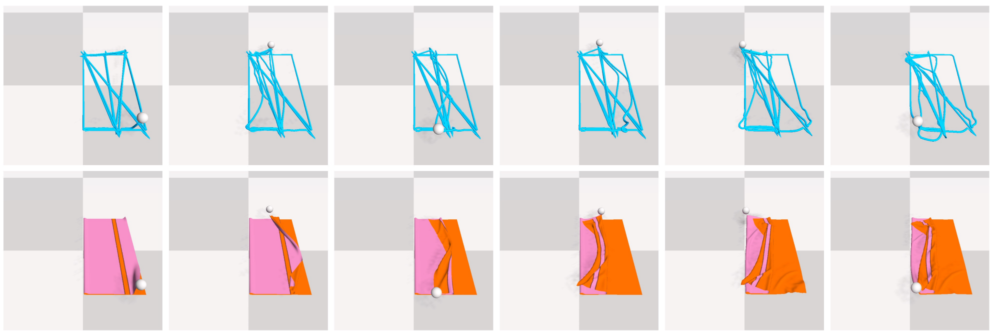

(b) Second simplified model

Figure 5.5: Cloth side folding with simplified models. Image (a) shows the trajectory planned based on the first simplified model and rolled out on the original model. Image (b) shows the refined simplified model from the actually achieved final state, and a new trajectory optimized from the second simplified model, which successfully bring the original model to a satisfying state.


(a) First simplified model

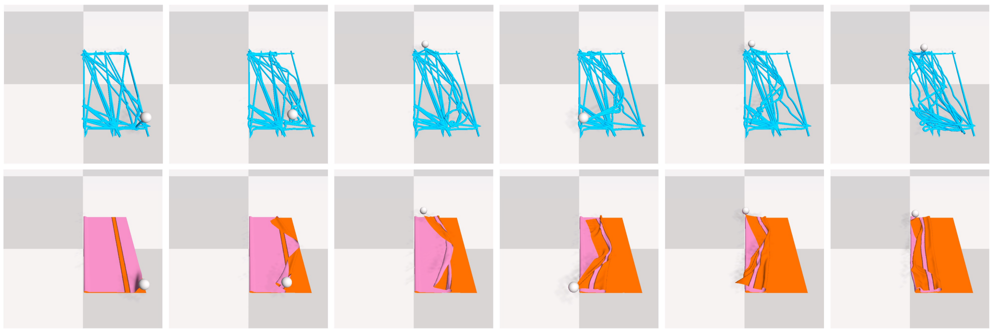
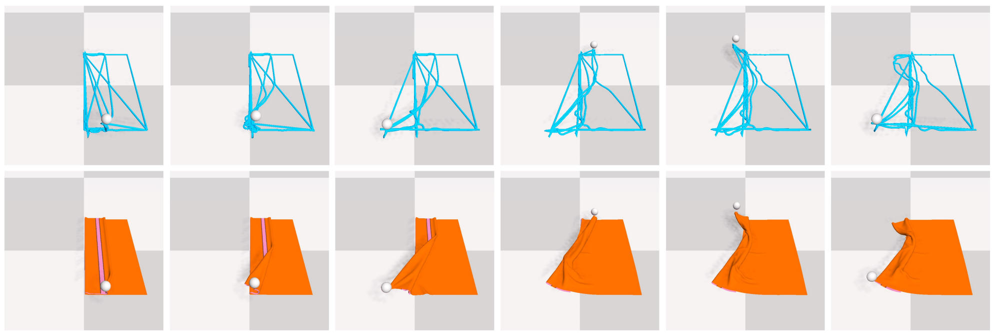

(b) Second simplified model

Figure 5.6: Cloth reflective folding with simplified models. Image (a) shows the trajectory planned based on the first simplified model and image (b) shows the improved model as well as the newly generated trajectory.

convergence.

From the results, the *Simplified Models* drops fastest for all tasks as well, and converges to better cost than *Original Model* and *Grid Models*, demonstrating the effectiveness of the proposed configuration-informed model simplification scheme. However, the *Simplified & Original Models* with RRT performs worse than CEM, especially on the cloth underneath folding task, where it converges to a worse cost than the *Simplified Models*, although it achieves similar costs to *Simplified Models* for the other tasks. This reflects the shortcomings of the altered warm start scheme, as the latter planning loop is highly dependent on the initial trajectory, reducing flexibility and performance. The *Grid Models* gets the worst cost among all models, further underscoring the drawbacks of the modified warm start scheme and indicating a lack of consistency in the planned trajectories for grid models of varying resolutions. An example of the trajectory planned based on the grid model is shown in Figure 5.8, which does not bring the original cloth to the goal state. The *Original Model* achieves acceptable results only for the cloth diagonal folding task, and converges to worse costs for other tasks due to the fact that the original model is more expensive and the state space is much larger compared to the simplified models.


(a) First simplified model

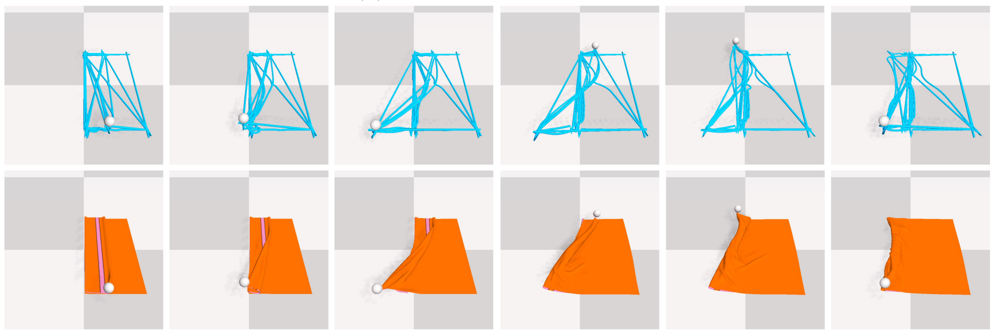

(b) Second simplified model

Figure 5.7: Cloth underneath folding with simplified models. Image (a) shows the plan generated by the first simplified model, and image (b) shows the plan refined on the second simplified model.

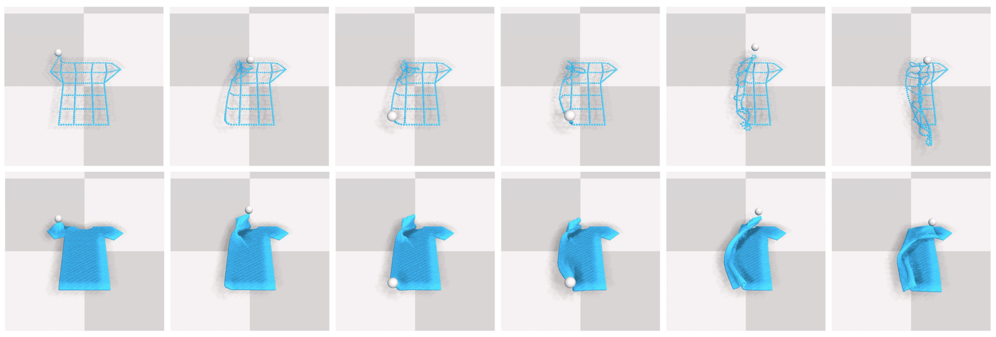
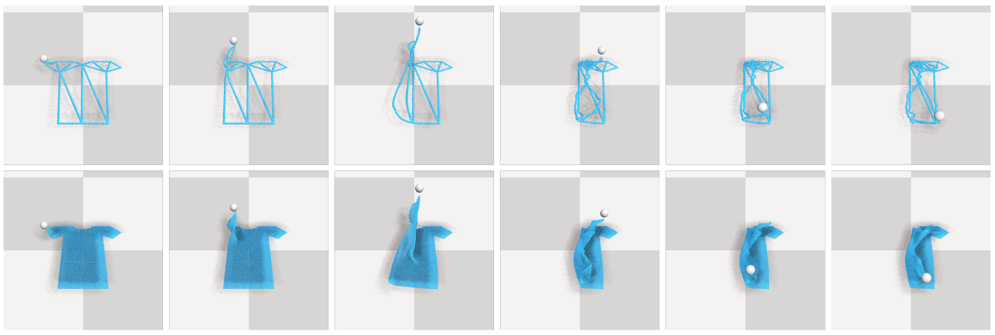


Figure 5.8: T-shirt Folding with Grid Models. The first row shows the planned trajectory based on the grid model, while the second row shows the trajectory rolled out on the original model.


(a) First simplified model

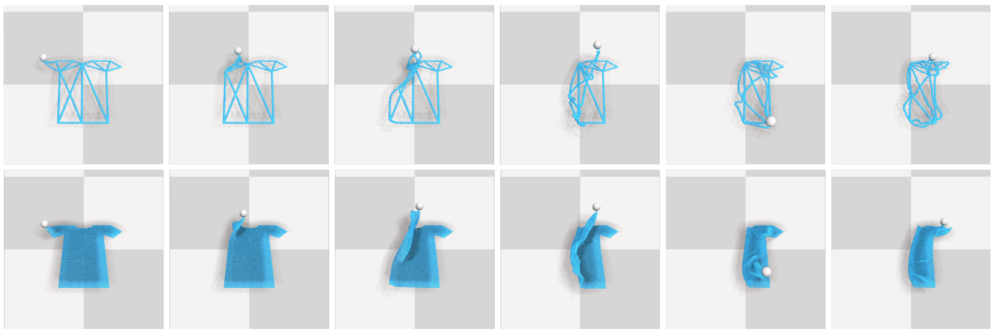

(b) Second simplified model

Figure 5.9: T-shirt side folding with simplified models. In image (a), a trajectory is planned using the initial simplified model, depicted in the first row, and then rolled out on the original model, shown in the second row. However, this does not achieve the intended goal, necessitating a new round of model simplification and motion planning. As demonstrated in image (b), these subsequent efforts successfully bring the cloth to the desired goal.

Regarding the iterative framework itself, I provide additional results on the average number of iterations required to achieve the best result. I also compare the trajectory cost obtained using the first single simplified model against the one found through the iterative framework, as shown in Tab. 5.1. It is observed that while the first simplified model already yields near-satisfactory results, the iterative framework is still able to further optimize the trajectory, leading to improved performance.

Table 5.1: Comparison of single simplified model and iterative simplified models. Results show: Final trajectory cost achieved (number of model iterations required).

Task	Diag.	Side	Reflec.	Undern.	T-shirt
Single	6.24	5.9	5.0	7.5	3.9
Iterative	1.76 (2.3)	3.8 (3.6)	3.8 (3.1)	5.3 (4.1)	2.5 (3.4)

5.4.3 Comparison with a Learned Model

To further verify the effectiveness of our proposed method, I compare it with a learning-based dynamics model originally developed by Lin et al. [12], called Visible Connectivity Dynamics (VCD). VCD employs a graph neural network (GNN) trained to infer both the connections and the dynamics of the visible part of the deformable object. For training the GNN, they collect 2000 trajectories, each containing a single pick-and-place action, but decompose each trajectory into 100 smaller steps. The picking point is biased toward the highest point, while the placing location is selected randomly, making it more suitable for flattening tasks. After training, a single-step Random Shooting planner is used to iteratively generate pick-and-place actions towards a goal.

Besides, to measure the performance, they also introduced a metric named Normalized Improvement (NI), which quantifies the increase in the covered area relative to the maximum possible improvement. It is defined as $NI = \frac{s-s_0}{s_{max}-s_0}$, where s_0, s, s_{max} represent the initial, achieved, and maximum possible covered areas of the cloth, respectively.

I used the official implementation of VCD[1]. The original implementation uses visible point cloud data, which we call VCD(PC) here. Additionally, to ensure a fair comparison, I implemented a different version of VCD, where we made two key modifications. First, instead of using point cloud data, I directly used the ground-truth positions of the particles from the original model of the object. Second, I provided the actual edges explicitly, eliminating the need for the trained neural network to infer them. These modifications were made both to ensure fairness in comparison and to facilitate the precise specification of goal states for subsequent folding tasks. We call this modified version that uses *ground truth* particles and edges, VCD(GT).

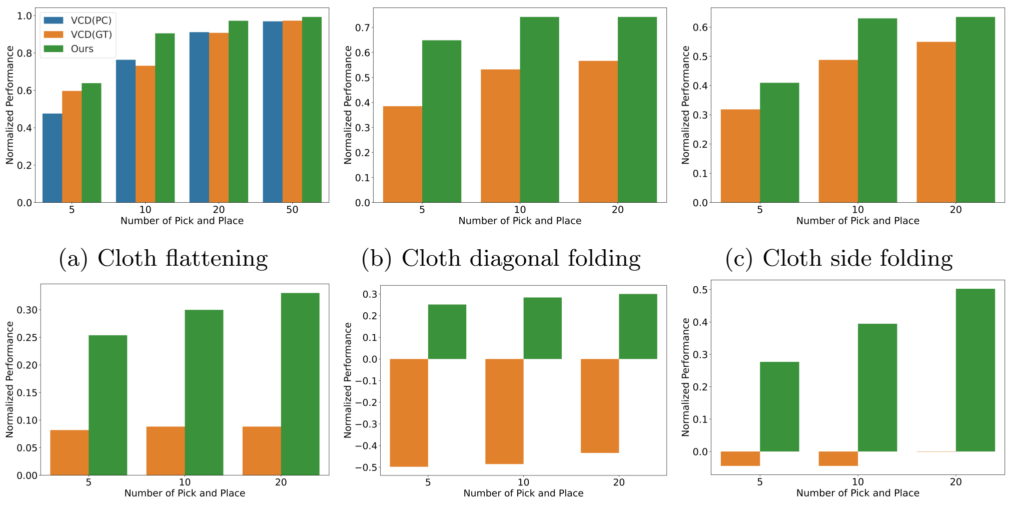
For comparing our method against VCD, I used the validation set from Lin et al., which includes sampled cloth pieces with different sizes and initial configurations.

First, I attempted to compare our method and VCD, by using the same CEM planner that I used over the learned dynamics model. However, the VCD performed very poorly with CEM, because CEM tries to generate multi-action plans, which required running the VCD predictions over the outputs of itself, which resulted in accumulation of drastic drifts in the predicted states, already showing the limits of learned dynamics models.

Next, I compared our method to VCD, using the single-step Random Shooting planner used in their original implementation. To do this, I had to modify our method such that, after extracting a simplified model for the overall goal of the task, I used Random Shooting to plan an action. After the next state was reached with the execution of this action, I used this new state to extract a new simplified model (similar to line 11 in Alg. 5) and I combined this with the first simplified model extracted from the goal (similar to line 12 in Alg. 5).

The results for the cloth flattening task are shown in Fig. 5.10(a) using the NI metric. VCD(PC) and VCD(GT) achieve similar results, which are also very similar to the results reported in Lin et al., confirming VCD(GT)'s effectiveness. Our goal-informed simplified model

[1]https://github.com/Xingyu-Lin/VCD



(a) Cloth flattening (b) Cloth diagonal folding (c) Cloth side folding

(d) Cloth reflective folding (e) Cloth underneath folding (f) T-shirt side folding

Figure 5.10: Planner (Random Shooting) performance over number of actions (pick and place) executed for different tasks using VCD and our goal-simplified models.

Table 5.2: Planning time (s) for each pick and place action

	Flatten	Diag.	Side	Refle.	Under.	T-shirt
No. of particles	1720	1600	1849	1849	2025	3812
Time (VCD(GT))	70.6	70.2	70.8	70.4	71.0	608
Time (Ours)	27.4	27.3	27.4	27.3	27.5	28.2

achieves slightly better performance, despite requiring no training.

For the remaining folding tasks, I extended the definition of Normalized Improvement (NI) by replacing the coverage metric with the reward function (negative of the cost) specific to each task. Each task was evaluated for up to 20 pick-and-place steps, as further improvements beyond this were minimal. The results in Fig. 5.10 show that for diagonal folding, VCD achieves around 50% improvement. As task complexity increases, its performance degrades slightly for side folding and drops significantly for reflective folding. For cloth-underneath folding and T-shirt folding, the VCD fails to generate any feasible plans.

I hypothesize that for cloth-underneath folding, the model fails because it has not encountered similar state transitions during training. For T-shirt folding, the problem dimensionality is significantly higher than that of the training dataset, the shape is different, and the folding task was not specifically trained in the learned model.

The average planning time for each pick-and-place action is reported in Table 5.2. For cloth folding tasks, the average planning time using VCD is approximately 70 seconds, which is consistent with the findings of Huang et al. [137], who reported a planning time of 100 seconds. In contrast, with our simplified dynamics model, the planner requires only 27 seconds per action, demonstrating a significant speed-up. For the T-shirt folding task, the increased number of particles leads to a substantial increase in the inference time of the learned dynamics model. This is further exacerbated by the fact that we did not apply downsampling to the particles or edge connections, preserving the original topology. However, our simplified model remains computationally efficient, requiring only a fraction of the time needed by the learned model.

5.5 Real Robot Experiments

As demonstrated in the simulation experiments, the variation of the iterative framework—planning a coarse trajectory based on the simplified dynamics model and fine-tuning it with the original dynamics model—results in the lowest-cost trajectories among all methods, although it takes slightly more time. Therefore, in this section, real-world cloth folding experiments are conducted to verify the feasibility of the planned trajectory using the *Simplified & Original Models* approach with different planners.

The experiments are conducted based on the established closed-loop system described in Section 4.6.2, with the tasks including cloth diagonal folding and side folding. Both CEM and RRT-based planners are tested, and each experiment is run 5 times to assess the efficacy and reliability of the framework. The metrics measured include planning time, success rate, and the average number of re-grasps. A summary of these experiments is provided in Table 5.3.

For cloth diagonal folding, CEM spends an average of 15.08 minutes to generate a trajectory, achieving a 100% success rate with an average of 4.2 re-grasps. As previously defined, an experiment is considered successful if the robot completes all the actions in the planned trajectory and brings the cloth to a reasonable goal state. RRT also achieves a 100% success rate but requires slightly more re-grasps, averaging 7.0 re-grasps. However, the time cost for RRT is smaller, averaging 10 minutes, and can be further reduced.

For cloth side folding, CEM requires more time than for diagonal folding, averaging 21.26 minutes per plan. The success rate is acceptable, achieving 4 successes out of 5 tests, even though the task is more complex than diagonal folding, as it requires alternating grasps between different corners of the cloth. Specifically, the cloth may behave unexpectedly due to its under-actuated dynamics. The average number of re-grasps is slightly higher than for diagonal folding,

Table 5.3: Real-world experiment results based on the simplified & original dynamics models

Task	Planner	Planning time (min)	Success Rate	Average re-grasps
Diagonal Folding	CEM	15.08	5/5	4.20
	RRT	10.00	5/5	7.00
Side Folding	CEM	21.26	4/5	5.60
	RRT	10.00	0/5	18.20

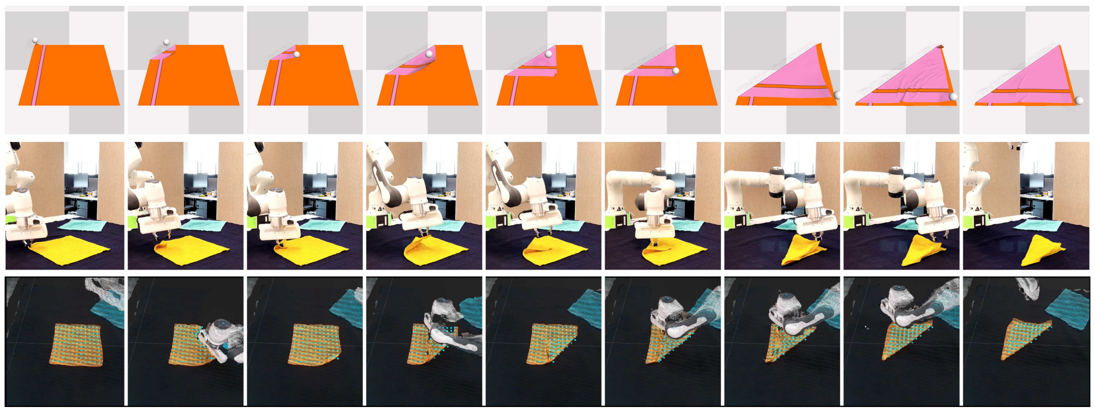


Figure 5.11: Real-world experiment for cloth diagonal folding with fine models. The first row displays the planned trajectory rolled out on the original model. The second row captures the robot executing this planned trajectory. The third row presents the tracking results, with green points overlapping the yellow cloth to show the alignment between the tracked and actual cloth positions.

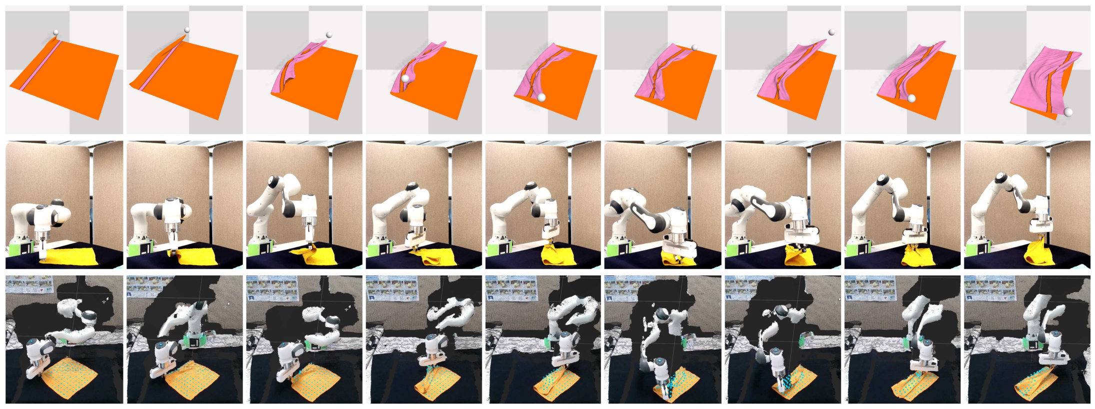


Figure 5.12: Real-world experiment for cloth side folding with fine models. The first row displays the planned trajectory rolled out on the original model. The second row captures the robot executing this trajectory. The third row shows the tracking results.

at 5.6 re-grasps on average. On the other hand, RRT fails all 5 experiments on the cloth side folding tasks, primarily due to the high number of re-grasps in the planned motion, an average of 18.2 re-grasps. This makes the experiment more susceptible to robot execution errors and perception failures, contributing to the overall failures.

The primary reasons for the failures in the experiments stem from issues with the tracking system, often caused by occlusion from the robot itself. In some cases, tracking inaccuracies lead the robot to pick the wrong location. Additionally, even when the robot selects the correct position, it sometimes inadvertently grasps multiple layers of the cloth, significantly altering its shape and disrupting the intended manipulation.

I also evaluate the performance of the real robot experiments by reconstructing the cloth's final state (as reported by our CDCPD-based shape tracking method) in simulation and assessing its similarity to the goal state using two metrics: NI and Intersection over Union (IoU). I considered an experiment as successful if either NI or IoU exceeds 70. For the diagonal folding tasks, the average successful NI is 0.85 with standard deviation 0.07, and the IoU is 0.92 with standard deviation 0.04. For the side folding tasks, the average successful NI is 0.69 with standard deviation 0.12, and the IoU is 0.87 with standard deviation 0.06.

Compared to the real robot experiments where planning is based on a single simplified dynamics model as described in Section 4.6.2, the experiments with *Simplified & Original Models* conducted in this section show some notable differences. First, the time cost of planning is significantly higher, though still considerably less than planning exclusively with the high-fidelity original dynamics model. This longer planning time makes it more appropriate for offline planning scenarios. In terms of performance, the success rate is slightly better than that of the previously planned trajectories with single simplified models. For all tasks and planners, the trajectories involve fewer re-grasps compared to the results in Section 4.6.2. This reduction in re-grasps suggests that the final state is achieved with more concise and efficient plans, and highlights that sub-optimal trajectories often contain unnecessary or even counterproductive motions.

Images of the real robot folding the cloth diagonally and sideways, using the *Simplified & Original Models* and CEM as the underlying motion planner, are shown in Figure 5.11 and Figure 5.12. The Panda robot successfully achieves both tasks.

During manipulation, when the cloth does not behave as predicted in simulation, the perception system can accurately track the state of the cloth, allowing the robot to adaptively adjust its actions to pick at the correct location. In cases where the robot fails to pick at the target location, it can detect the failure through the sensed force from the gripper. The robot then widens the width of the gripper and successfully re-grasps at the intended location. These behaviors demonstrate the robustness of the closed-loop execution system, enhancing the overall reliability of the manipulation process despite unexpected deviations.

5.6 Conclusion

In this section, I propose an iterative model simplification and motion planning framework that enables the planner to progressively generate motion plans while simultaneously refining the simplified dynamics models. This approach allows for the gradual reduction of the behavioral gap between the simplified dynamics model and the original high-fidelity dynamics model. By balancing efficiency and optimality, the planner can adaptively decide when to terminate the planning loop. Two implementations of the iterative framework are introduced. One is configuration-informed iterative simplification, which simplifies and incrementally improves the dynamics model based on specific configurations. Another is uninformed grid-based iterative simplification, which employs a sequence of grid-based models with increasing resolutions without relying on specific goal configurations. Extensive simulation experiments are conducted in simulation, showing that both implementations significantly decrease the planning time compared to using the original dynamics model. The planned trajectories are more optimized than those generated using a single simplified model. The configuration-informed implementation, in particular, provides better efficiency and lower trajectory costs, demonstrating the effectiveness of utilizing goal information in planning. In the real-world experiment, the success rates and trajectory qualities achieved with the iterative framework are superior to those obtained when planning with a single simplified dynamics model, albeit with some sacrifice in planning efficiency. However, failure cases in real-world experiments indicate that the tracking system requires further improvement to enhance robustness. One potential solution is to incorporate the established simplified dynamics into the perception pipeline, which could improve both the robustness and accuracy of the system.

Chapter 6

Conclusion & Future Work

In this chapter, I summarize the key contributions of this thesis, highlight the limitations, and suggest potential directions for future research.

6.1 Conclusion

This thesis focuses on deformable object manipulation, a task that is common in everyday life but presents significant challenges for robots due to high-dimensional configuration space, large action space, complex dynamics, and severe self-occlusion. These factors make motion planning, perception, and control for deformable objects extremely difficult and computationally intensive.

To address these challenges, I investigated methods to alleviate the computational burden associated with motion planning for deformable object manipulation, as well as related shape tracking and control methods.

Firstly, in Chapter 3, I proposed reducing the action space to accelerate motion planning. This reduction is achieved in a goal-conditioned manner by geometric model simplification, utilizing information from various goals and tasks to truncate the action space. Specifically, I simplified the geometric model based on the goal by constructing a minimal geometric model that has as few elements as possible while still providing a good approximation of the goal state of the original model. Two general implementations were designed, including piece-wise line fitting for 1-D linear objects (e.g., ropes and cables) and mesh simplification for 2-D surface objects (e.g., cloth). By extracting corresponding particles on the original model as key picking points in the action space, the planner can generate more effective trajectories more efficiently. The simulation results confirmed the effectiveness of this approach, demonstrating generalizability to different tasks and planners. Additionally, the feasibility of the planned trajectories was validated in real-world experiments using a Franka Panda robot, employing a combined control scheme of position control and impedance control.

Then, in Chapter 4, I further reduced the computational cost of motion planning for deformable object manipulation by simplifying the dynamics models and enhanced the robustness of execution through an improved tracking system and a closed-loop execution framework. Specifically, I introduced two dynamics model simplification methods: one is a goal-informed dynamics model simplification, based on the geometric model simplification from Chapter 3, and the other is an uninformed model simplification, which uniformly reduces the resolution of the original dynamics model into grids of various sizes. By integrating these simplified dynamics models into the planner, the computational time required to evaluate multiple candidate trajectories is significantly reduced, though this comes at the cost of sacrificing certain degrees of precision. Simulation experiments validated the efficiency boost achieved with the simplified dynamics models, and showed that the goal-informed simplification method is more effective than the uninformed method by leveraging information from the goal state. Additionally, to improve the robustness of executing the planned trajectories in real-world scenarios, the tracking algorithm was enhanced in two ways: 1) by explicitly accounting for self-occlusion when the deformable object folds into layers, where a thicker point cloud is generated by combining current and previous point clouds for more accurate tracking, and 2) by selectively updating the tracking points so that only those positioned within the changed point cloud are relocated. These enhancements to the tracking system enable more accurate monitoring of the shape of the deformable object during manipulation. A closed-loop experiment system, consisting of the efficient motion planning subsystem, the tracking subsystem, and the robot, was then established. This closed-loop system allows the robot to adaptively adjust its actions in real time based on feedback from the perception system, addressing discrepancies between the planned and actual object behaviors. Real-world robot experiments demonstrated improved tracking precision of the enhanced tracking system and increased robustness of manipulation with the closed-loop execution system.

Finally, in Chapter 5, based on the proposed geometric model simplification, action space reduction, and dynamics model simplification methods, I proposed an iterative model simplification & motion planning framework. In this framework, the dynamics model was simplified and gradually improved along the course of motion planning. The behavioral gap between the simplified and original models is gradually reduced, enabling continual improvement of the trajectory by using the previously planned trajectory as a warm start for the next iteration. This iterative loop continues as long as time permits, allowing for users to balance between optimality and efficiency depending on the specific requirements of different tasks. Two implementations of this framework are explored, including a configuration-informed iterative model simplification, where the dynamics model is initially simplified based on the goal and then iteratively refined based on the actual state achieved, and an uninformed grid-based iterative simplification, which progressively refines the dynamics model by increasing the resolution of grid-based models without relying on specific goal information. Simulation experiments demonstrated that the quality of the planned trajectories improves progressively as the dynamics model is refined from its simplest form to more detailed versions. The configuration-informed approach outperforms the uninformed method, highlighting the value of goal-specific information in creating more effective simplified dynamics models, which in turn enhance the motion planning process. Real-world robot experiments validated the effectiveness of the framework, showing higher success rates compared to planning based solely on a single simplified model. The resulting trajectories were more optimized, with lower costs and fewer re-grasps in general, further demonstrating the practical benefits of the iterative approach.

6.2 Limitations & Future Work

Although the results in this thesis showed improvements in planning efficiency, tracking precision, and execution robustness, there are still some limitations. Firstly, the time cost of motion planning is still relatively high for real-time applications, even with the most simplified dynamics models, which indicates the need for further optimization to enhance planning efficiency. Secondly, the differences in the material properties between the simulated deformable objects and their real-world counterparts are not accounted for, which may affect the accuracy of the simulation and the feasibility of the planned trajectory. Thirdly, the interaction between the gripper and the deformable object is also simplified, which is abstracted as a sphere anchored to a specific particle. This abstraction lacks realism and may not capture the complexities of real-world interactions, potentially impacting the effectiveness of the manipulation strategies. Additionally, the gripper used in real-world experiments is an off-the-shelf two-fingered gripper that is relatively wide and lacks versatility and agility, which affects the delicacy of grasping motions. Moreover, in some cases of failure in real robot experiments, the gripper fails to differentiate between different layers of the deformable object. Besides, the perception system is not sufficiently robust, particularly under severe occlusion or after a long sequence of actions, where error accumulation can lead to failures. Lastly, the motions of the gripper are restricted by the assumption of quasi-static conditions, resulting in a slow manipulation process, which may not be optimal for tasks requiring more dynamic interactions. In this section, I consider these limitations and propose potential future directions to improve the work further.

6.2.1 Using Better Planners, Parallelization, or Cloud Computing

In this thesis, I addressed the high computational cost of motion planning by reducing the action space, and simplifying deformable object models, while leveraging off-the-shelf motion planners without customization. Future work could focus on designing more efficient motion planning algorithms tailored specifically for deformable object manipulation. These planners could incorporate advanced optimization techniques, hierarchical planning, or other strategies that significantly reduce the computational burden.

Another approach could involve utilizing large-scale parallel computing or distributed processing, allowing multiple trajectory rollouts to be computed simultaneously. This would lead to faster decision-making, especially for real-time applications. Moreover, leveraging more powerful remote computing resources, such as cloud computing, could further enhance the efficiency of the planner by offloading intensive calculations to cloud servers, thus reducing reliance on local hardware and potentially lowering costs.

6.2.2 Deformable Object Property Identification

In this work, the dynamics model used in simulation is prebuilt and coarsely tuned to approximate the behavior of the deformable object in the real world. However, this may result in inaccurate modeling and lead to suboptimal or ineffective motion plans. A promising future direction is to integrate deformable object property identification directly into the planning and execution pipeline. Tactile sensors could potentially be used for this task, as they provide detailed information about the interaction forces between the gripper and the object. The physical properties of the deformable object, such as stiffness, elasticity, and damping, can then be estimated based on sensed data and used to refine the deformable object model in simulation, thereby reducing the gap between simulated and real-world dynamics.

6.2.3 Modeling of Interaction between Rigid Grippers and Deformable Objects

The interaction between the rigid gripper and the deformable object is simplified in this thesis for the sake of simplicity and efficiency. This is because modeling the interaction between rigid grippers and deformable objects is inherently complex and difficult to capture accurately. However, if this interaction can be accurately modeled, it could generate more realistic and potentially more robust motion plans, enabling more precise control and manipulation, leading to reduced failure rates and improved task execution. Such advancements could also allow for safer manipulation of delicate deformable objects, such as soft fabrics or biological tissues, broadening the application to more sensitive tasks and domains.

6.2.4 Designing and Using Dexterous Hands and Deformable Grippers

The grippers used for manipulation have a significant influence when dealing with deformable objects. In this thesis, I employed a basic two-fingered gripper capable of simple open-and-close motions. In contrast, humans manipulate deformable objects with five dexterous fingers, allowing for far more flexible and precise control. One potential direction for future research is to explore the use of multi-fingered dexterous robotic hands, which could replicate the ability of the human hand to grasp, pinch, and reposition objects with a higher degree of control. Additionally, soft or deformable robotic hands can also be used to further enhance safety and reduce the risk of damage to sensitive deformable objects.

6.2.5 Consider Multiple Layers for Cloth Manipulation

When a cloth is folded into multiple layers, it becomes extremely difficult for the robot gripper to differentiate between the layers, often leading to unexpected shape changes if multiple layers are grasped simultaneously. This challenge remains under-researched, and one potential solution is to develop more precise shape tracking methods, in conjunction with more sophisticated grippers that can handle such intricate manipulations.

Chen et al. [23] propose singulating layers of a bag using interactive perception for a downstream task such as inserting an item. The folding is more complex than that, and in such cases, tactile sensors could be a viable option, allowing the gripper to sense if multiple layers are being grasped, enabling more careful and controlled manipulation of the object.

6.2.6 Improve Tracking with Physics-based Simulation

The tracking method used in this thesis does not account for the kinodynamics of the deformable object, which can lead to unrealistic shape changes when occlusion occurs. One way to eliminate this limitation is to use a physics simulator to regulate the tracking results, as demonstrated in both rigid and deformable object tracking, such as in [138] and [62]. However, for deformable object tracking, including such physics-based models introduces a significant computational burden that may hinder real-time applications. A potential solution is to incorporate the efficient simplified models proposed in this thesis to maintain real-time performance while benefiting from physics-based constraints. This approach could balance the need for physical accuracy with computational efficiency, enabling more practical and scalable tracking methods for deformable object manipulation.

6.2.7 Dynamic Manipulation of Deformable Objects

Due to the high complexity of the dynamics model of the deformable object, even a small action may cause drastic and unpredictable shape changes. To prevent this, the robot motions in all experiments in this thesis were deliberately confined, resulting in very slow execution times. For future work, dynamic motions with higher speeds could potentially improve efficiency. As proposed by Ha et al. in [139], a simple fling motion makes it much easier to flatten the cloth that may be out of the manipulation range of the robot arm. Similarly, Chi et al. [140] and Avigal et al. [141] explored dynamic manipulation techniques that leverage motion primitives for speed and efficiency. While these dynamic motions show promise in improving task completion times, they are typically achieved through predefined motion primitives, which may be efficient but lack the flexibility needed for more general or complex manipulation tasks. Future research could aim to develop more adaptive and versatile dynamic motion strategies.

6.2.8 Embracing Learning-based Methods for Deformable Object Manipulation

Instead of relying on model-based methods, learning-based approaches offer an interesting avenue for further exploration and comparison. For example, as demonstrated in [66] and [67], neural networks can be trained to predict the underlying state of a mesh model for specific garment categories, whether for state estimation or shape tracking. However, such methods require huge datasets, which are difficult to acquire, and the trained networks tend to only perform well on specific object types. Nevertheless, learning-based approaches hold promise due to their ease of use and end-to-end structure if trained effectively. Another direction is learning a dynamics model, which could enable much faster trajectory rollouts compared to using the original high-fidelity but time-consuming models. However, this approach faces similar challenges as learning-based perception, including the difficulty of acquiring suitable training datasets and ensuring generalizability across different types of objects. For instance, as discussed in [137], a trained graph neural network-based dynamics model still takes around 100 seconds to roll out 500 pick-and-place actions during planning. This is far from real-time planning and not significantly faster than our untrained models. Additionally, most learning-based methods aim to capture the full dynamics of the high-fidelity models. There is potential to explore the use of learning-based approaches to develop coarser, more efficient models instead. Furthermore, combining planning based on either explicitly simplified or learned models with learning-from-demonstration techniques may offer enhanced generalizability and faster motion generation.

6.3 Summary

In this chapter, I summarized the main contributions of this thesis for deformable object manipulation, including geometric simplification, goal-conditioned action space reduction, dynamics model simplification, enhanced deformable object shape tracking, and an iterative model simplification & motion planning framework. These methods promote the development in this field, but more importantly, they also expose limitations, challenges, and potential directions for future research. I believe that by addressing these challenges, we as a community can bring it closer to realizing the vision of having robots everywhere in our daily life, capable of handling complex deformable object manipulation tasks autonomously and efficiently.

Bibliography

[1] Aviv Adler, Ayah Ahmad, Yulei Qiu, Shengyin Wang, Wisdom C Agboh, Edith Llontop, Tianshuang Qiu, Jeffrey Ichnowski, Thomas Kollar, Richard Cheng, Mehmet Dogar, and Ken Goldberg. The teenager's problem: Efficient garment decluttering as probabilistic set cover. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*. None, 2024.

[2] Aviv Adler, Ayah Ahmad, Shengyin Wang, Wisdom C Agboh, Edith Llontop, Tianshuang Qiu, Jeffrey Ichnowski, Mehmet Dogar, Thomas Kollar, Richard Cheng, et al. The teenager's problem: Efficient garment decluttering with grasp optimization. *arXiv preprint arXiv:2310.16951*, 2023.

[3] Hang Yin, Anastasia Varava, and Danica Kragic. Modeling, learning, perception, and control methods for deformable object manipulation. *Science Robotics*, 6(54):eabd8803, 2021.

[4] Jihong Zhu, Andrea Cherubini, Claire Dune, David Navarro-Alarcon, Farshid Alambeigi, Dmitry Berenson, Fanny Ficuciello, Kensuke Harada, Jens Kober, Xiang Li, et al. Challenges and outlook in robotic manipulation of deformable objects. *IEEE Robotics & Automation Magazine*, 29(3):67–77, 2022.

[5] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021.

[6] David Blanco-Mulero, Oriol Barbany, Gokhan Alcan, Adrià Colomé, Carme Torras, and Ville Kyrki. Benchmarking the sim-to-real gap in cloth manipulation. *IEEE Robotics and Automation Letters*, 2024.

[7] Yunfei Bai, Wenhao Yu, and C Karen Liu. Dexterous manipulation of cloth. In *Computer Graphics Forum*, pages 523–532. Wiley Online Library, 2016.

[8] Veronica E Arriola-Rios, Puren Guler, Fanny Ficuciello, Danica Kragic, Bruno Siciliano, and Jeremy L Wyatt. Modeling of deformable objects for robotic manipulation: A tutorial and review. *Frontiers in Robotics and AI*, 7:82, 2020.

[9] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef 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.

[10] Jadav Das and Nilanjan Sarkar. Autonomous shape control of a deformable object by multiple manipulators. *Journal of Intelligent & Robotic Systems*, 62:3–27, 2011.

[11] Simon Zimmermann, Roi Poranne, and Stelian Coros. Dynamic manipulation of deformable objects with implicit integration. *IEEE Robotics and Automation Letters*, 6(2): 4209–4216, 2021.

[12] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, pages 256–266. PMLR, 2022.

[13] Mitul Saha and Pekka Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007.

[14] Jihong Zhu, Benjamin Navarro, Robin Passama, Philippe Fraisse, André Crosnier, and Andrea Cherubini. Robotic manipulation planning for shaping deformable linear objects withenvironmental contacts. *IEEE Robotics and Automation Letters*, 5(1):16–23, 2019.

[15] Alessio Caporali, Piotr Kicki, Kevin Galassi, Riccardo Zanella, Krzysztof Walas, and Gianluca Palli. Deformable linear objects manipulation with online model parameters estimation. *IEEE Robotics and Automation Letters*, 2024.

[16] Irene Garcia-Camacho, Júlia Borràs, Berk Calli, Adam Norton, and Guillem Alenyà. Household cloth object set: Fostering benchmarking in deformable object manipulation. *IEEE Robotics and Automation Letters*, 7(3):5866–5873, 2022.

[17] Halid Abdulrahim Kadi and Kasim Terzić. Data-driven robotic manipulation of cloth-like deformable objects: The present, challenges and future prospects. *Sensors*, 23(5):2389, 2023.

[18] Rishabh Jangir, Guillem Alenya, and Carme Torras. Dynamic cloth manipulation with deep reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4630–4636. IEEE, 2020.

[19] Bokui Shen, Zhenyu Jiang, Christopher Choy, Silvio Savarese, Leonidas J Guibas, Anima Anandkumar, and Yuke Zhu. Action-conditional implicit visual dynamics for deformable object manipulation. *The International Journal of Robotics Research*, 43(4):437–455, 2024.

[20] Alison Bartsch, Charlotte Avra, and Amir Barati Farimani. Sculptbot: Pre-trained models for 3d deformable object manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12548–12555. IEEE, 2024.

[21] Bao Thach, Alan Kuntz, and Tucker Hermans. Deformernet: A deep learning approach to 3d deformable object manipulation. *arXiv preprint arXiv:2107.08067*, 2021.

[22] P Jiménez. Survey on model-based manipulation planning of deformable objects. *Robotics and computer-integrated manufacturing*, 28(2):154–163, 2012.

[23] Lawrence Yunliang Chen, Baiyu Shi, Roy Lin, Daniel Seita, Ayah Ahmad, Richard Cheng, Thomas Kollar, David Held, and Ken Goldberg. Bagging by learning to singulate layers using interactive perception. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3176–3183. IEEE, 2023.

[24] Shiyu Jin, Wenzhao Lian, Changhao Wang, Masayoshi Tomizuka, and Stefan Schaal. Robotic cable routing with spatial representation. *IEEE Robotics and Automation Letters*, 7(2):5687–5694, 2022.

[25] Junlei Hu, Dominic Jones, Mehmet R Dogar, and Pietro Valdastri. Occlusion-robust autonomous robotic manipulation of human soft tissues with 3d surface feedback. *IEEE Transactions on Robotics*, 2023.

[26] Willow Mandil, Vishnu Rajendran, Kiyanoush Nazari, and Amir Ghalamzan-Esfahani. Tactile-sensing technologies: Trends, challenges and outlook in agri-food manipulation. *Sensors*, 23(17):7362, 2023.

[27] Carl Qi, Xingyu Lin, and David Held. Learning closed-loop dough manipulation using a differentiable reset module. *IEEE Robotics and Automation Letters*, 7(4):9857–9864, 2022.

[28] Jihong Zhu, Michael Gienger, Giovanni Franzese, and Jens Kober. Do you need a hand?–a bimanual robotic dressing assistance scheme. *IEEE Transactions on Robotics*, 40:1906–1919, 2024.

[29] Prabhakar Ray and Matthew J Howard. Robotic untangling of herbs and salads with parallel grippers. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2624–2629. IEEE, 2020.

[30] Kanata Suzuki, Momomi Kanamura, Yuki Suga, Hiroki Mori, and Tetsuya Ogata. In-air knotting of rope using dual-arm robot based on deep learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6724–6731. IEEE, 2021.

[31] Daniel Seita, Nawid Jamali, Michael Laskey, Ajay Kumar Tanwani, Ron Berenstein, Prakash Baskaran, Soshi Iba, John Canny, and Ken Goldberg. Deep transfer learning of pick points on fabric for robot bed-making. In *The International Symposium of Robotics Research*, pages 275–290. Springer, 2019.

[32] Yulei Qiu, Jihong Zhu, Cosimo Della Santina, Michael Gienger, and Jens Kober. Robotic fabric flattening with wrinkle direction detection. In *International Symposium on Experimental Robotics*, pages 339–350. Springer, 2023.

[33] Robert Lee, Daniel Ward, Vibhavari Dasagi, Akansel Cosgun, Juxi Leitner, and Peter Corke. Learning arbitrary-goal fabric folding with one hour of real robot experience. In *Conference on Robot Learning*, pages 2317–2327. PMLR, 2021.

[34] Marcos García, César Mendoza, Luis Pastor, and Angel Rodríguez. Optimized linear fem for modeling deformable objects. *Computer Animation and Virtual Worlds*, 17(3-4): 393–402, 2006.

[35] Pradeepkumar Suryawanshi and Abhishek Gupta. A novel mass spring model for simulating deformable objects. *Journal of Mechanics of Materials and Structures*, 18(2):143–168, 2023.

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

[37] Niels Saabye Ottosen and Matti Ristinmaa. *The mechanics of constitutive modeling.* Elsevier, 2005.

[38] Singiresu S Rao. *The finite element method in engineering.* Elsevier, 2010.

[39] Mandela Ouafo Fonkoua, François Chaumette, and Alexandre Krupa. Deformation control of a 3d soft object using rgb-d visual servoing and fem-based dynamic model. *IEEE Robotics and Automation Letters*, 9(8):6943–6950, 2024.

[40] Adrien Koessler, N Roca Filella, Belhassen-Chedli Bouzgarrou, Laurent Lequièvre, and J-A Corrales Ramon. An efficient approach to closed-loop shape control of deformable objects using finite element models. In *2021 IEEE International conference on robotics and automation (ICRA)*, pages 1637–1643. IEEE, 2021.

[41] George Celniker and Dave Gossard. Deformable curve and surface finite-elements for free-form shape design. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 257–266, 1991.

[42] Kiattisak Sangpradit, Hongbin Liu, Prokar Dasgupta, Kaspar Althoefer, and Lakmal D Seneviratne. Finite-element modeling of soft tissue rolling indentation. *IEEE Transactions on Biomedical Engineering*, 58(12):3319–3327, 2011.

[43] Sylvester Arnab and Vinesh Raja. Simulating a deformable object using a surface mass spring system. In *2008 3rd international conference on geometric modeling and imaging*, pages 21–26. IEEE, 2008.

[44] Jaruwan Mesit, Ratan K Guha, and Shafaq Chaudhry. 3d soft body simulation using mass-spring system with internal pressure force and simplified implicit integration. *J. Comput.*, 2(8):34–43, 2007.

[45] Zhou Zhang. Soft-body simulation with cuda based on mass-spring model and verlet integration scheme. In *ASME International Mechanical Engineering Congress and Exposition*, volume 84546, page V07AT07A025. American Society of Mechanical Engineers, 2020.

[46] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):1–7, 2013.

[47] Fei Liu, Entong Su, Jingpei Lu, Mingen Li, and Michael C Yip. Robotic manipulation of deformable rope-like objects using differentiable compliant position-based dynamics. *IEEE Robotics and Automation Letters*, 8(7):3964–3971, 2023.

[48] Hongly Va, Min-Hyung Choi, and Min Hong. Efficient simulation of volumetric deformable objects in unity3d: Gpu-accelerated position-based dynamics. *Electronics*, 12(10):2229, 2023.

[49] Fei Liu, Entong Su, Jingpei Lu, Mingen Li, and Michael C Yip. Differentiable robotic manipulation of deformable rope-like objects using compliant position-based dynamics. *arXiv preprint arXiv:2202.09714*, 2022.

[50] Marco Fratarcangeli and Fabio Pellacini. A gpu-based implementation of position based dynamics for interactive deformable bodies. *Journal of Graphics Tools*, 17(3):59–66, 2013.

[51] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[52] Erwin Coumans and Yunfei Bai. Pybullet quickstart guide, 2021.

[53] François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, Stéphanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, et al. Sofa: A multi-model framework for interactive physical simulation. *Soft tissue biomechanical modeling for computer assisted surgery*, pages 283–321, 2012.

[54] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. Softgym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 432–448. PMLR, 2021.

[55] Benjamin Balaguer and Stefano Carpin. Combining imitation and reinforcement learning to fold deformable planar objects. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1405–1412. IEEE, 2011.

[56] Pasu Boonvisut and M Cenk Çavuşoğlu. Estimation of soft tissue mechanical parameters from robotic manipulation data. *IEEE/ASME Transactions on Mechatronics*, 18(5):1602–1611, 2012.

[57] Yinoussa Adagolodjo, Nicolas Golse, Eric Vibert, Michel De Mathelin, Stéphane Cotin, and Hadrien Courtecuisse. Marker-based registration for large deformations-application to open liver surgery. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4007–4012. IEEE, 2018.

[58] Nicolas Padoy and Gregory D Hager. Deformable tracking of textured curvilinear objects. In *BMVC*, pages 1–11, 2012.

[59] Daniel Pizarro and Adrien Bartoli. Feature-based deformable surface detection with self-occlusion reasoning. *International journal of computer vision*, 97:54–70, 2012.

[60] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

[61] Te Tang, Changhao Wang, and Masayoshi Tomizuka. A framework for manipulating deformable linear objects by coherent point drift. *IEEE Robotics and Automation Letters*, 3(4):3426–3433, 2018.

[62] Te Tang, Yongxiang Fan, Hsien-Chung Lin, and Masayoshi Tomizuka. State estimation for deformable objects by point registration and dynamic simulation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2427–2433. IEEE, 2017.

[63] John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking deformable objects with point clouds. In *2013 IEEE International Conference on Robotics and Automation*, pages 1130–1137. IEEE, 2013.

[64] Cheng Chi and Dmitry Berenson. Occlusion-robust deformable object tracking without physics simulation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6443–6450. IEEE, 2019.

[65] Yixuan Wang, Dale McConachie, and Dmitry Berenson. Tracking partially-occluded deformable objects while enforcing geometric constraints. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14199–14205. IEEE, 2021.

[66] Cheng Chi and Shuran Song. Garmentnets: Category-level pose estimation for garments via canonical space shape completion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3324–3333, 2021.

[67] Han Xue, Wenqiang Xu, Jieyi Zhang, Tutian Tang, Yutong Li, Wenxin Du, Ruolin Ye, and Cewu Lu. Garmenttracking: Category-level garment pose tracking. In *Proceedings of*

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 21233–21242, 2023.

[68] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[69] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77, 2014.

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

[71] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[72] Rafael Papallas, Anthony G Cohn, and Mehmet R Dogar. Online replanning with human-in-the-loop for non-prehensile manipulation in clutter—a trajectory optimization based approach. *IEEE Robotics and Automation Letters*, 5(4):5377–5384, 2020.

[73] Rafael Papallas and Mehmet R Dogar. Non-prehensile manipulation in clutter with human-in-the-loop. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6723–6729. IEEE, 2020.

[74] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[75] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[76] Giray Havur, Guchan Ozbilgin, Esra Erdem, and Volkan Patoglu. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 445–452. IEEE, 2014.

[77] Muhammad Suhail Saleem and Maxim Likhachev. Planning with selective physics-based simulation for manipulation among movable objects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6752–6758. IEEE, 2020.

[78] Mehmet Remzi Dogar, Kaijen Hsiao, Matei T Ciocarlie, and Siddhartha S Srinivasa. Physics-based grasp planning through clutter. In *Robotics: Science and systems*, volume 8, pages 57–64, 2012.

[79] Wissam Bejjani, Mehmet R Dogar, and Matteo Leonetti. Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning.

In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6562–6569. IEEE, 2019.

[80] Nikita Kitaev, Igor Mordatch, Sachin Patil, and Pieter Abbeel. Physics-based trajectory optimization for grasping in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3102–3109. IEEE, 2015.

[81] Sachin Chitta. Moveit!: an introduction. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 3–27, 2016.

[82] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[83] Olzhas Adiyatov and Huseyin Atakan Varol. Rapidly-exploring random tree based memory efficient motion planning. In *2013 IEEE international conference on mechatronics and automation*, pages 354–359. IEEE, 2013.

[84] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134:19–67, 2005.

[85] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[86] Probabilistic Roadmap Planning. On the probabilistic foundations of probabilistic roadmap planning. In *Robotics Research: Results of the 12th International Symposium ISRR*, volume 28, page 83. Springer, 2007.

[87] Nancy M Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of IEEE international conference on robotics and automation*, volume 1, pages 113–120. IEEE, 1996.

[88] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. *International Journal of Computer Science and Network Security (IJCSNS)*, 16(10):20, 2016.

[89] Sebastian Klemm, Jan Oberländer, Andreas Hermann, Arne Roennau, Thomas Schamm, J Marius Zollner, and Rüdiger Dillmann. Rrt*-connect: Faster, asymptotically optimal motion planning. In *2015 IEEE international conference on robotics and biomimetics (ROBIO)*, pages 1670–1677. IEEE, 2015.

[90] Jennifer E King, Matthew Klingensmith, Christopher M Dellin, Mehmet Remzi Dogar, Prasanna Velagapudi, Nancy S Pollard, and Siddhartha S Srinivasa. Pregrasp manipulation as trajectory optimization. In *Robotics: Science and Systems*. Berlin, 2013.

[91] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International journal of robotics research*, 32(9-10):1164–1193, 2013.

[92] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *First International Conference on Informatics in Control, Automation and Robotics*, volume 2, pages 222–229. SciTePress, 2004.

[93] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[94] Francis Celeste, Frédéric Dambreville, and J-P Le Cadre. Optimal path planning using cross-entropy method. In *2006 9th International Conference on Information Fusion*, pages 1–8. IEEE, 2006.

[95] Safiuddin Che Suhaimin, Nur Liyana Azmi, Md Mozasser Rahman, and Hazlina Md Yusof. Analysis of point-to-point robotic arm control using pid controller. In *2019 7th International Conference on Mechatronics Engineering (ICOM)*, pages 1–6. IEEE, 2019.

[96] Florian Petit and Alin Albu-Schäffer. Cartesian impedance control for a variable stiffness robot arm. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4180–4186. IEEE, 2011.

[97] Mark Moll and Lydia E Kavraki. Path planning for minimal energy curves of constant length. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2826–2831. IEEE, 2004.

[98] Ixchel G Ramirez-Alpizar, Kensuke Harada, and Eiichi Yoshida. Motion planning for dual-arm assembly of ring-shaped elastic objects. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 594–600. IEEE, 2014.

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

[100] Kyoko Hamajima and Masayoshi Kakikura. Planning strategy for task of unfolding clothes (classification of clothes). *Journal of Robotics and Mechatronics*, 12(5):577–584, 2000.

[101] Dale McConachie, Andrew Dobson, Mengyao Ruan, and Dmitry Berenson. Manipulating deformable objects by interleaving prediction, planning, and control. *The International Journal of Robotics Research*, 39(8):957–982, 2020.

[102] Mengyao Ruan, Dale McConachie, and Dmitry Berenson. Accounting for directional rigidity and constraints in control for manipulation of deformable objects without physical simulation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 512–519. IEEE, 2018.

[103] Dmitry Berenson. Manipulation of deformable objects without modeling and simulating deformation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4525–4532. IEEE, 2013.

[104] Li Sun, Gerarado Aragon-Camarasa, Paul Cockshott, Simon Rogers, and J Paul Siebert. A heuristic-based approach for flattening wrinkled clothes. In *Towards Autonomous Robotic Systems: 14th Annual Conference, TAROS 2013, Oxford, UK, August 28–30, 2013, Revised Selected Papers 14*, pages 148–160. Springer Berlin Heidelberg, 2014.

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

[106] Stephen Miller, Jur Van Den Berg, Mario Fritz, Trevor Darrell, Ken Goldberg, and Pieter Abbeel. A geometric approach to robotic laundry folding. *The International Journal of Robotics Research*, 31(2):249–267, 2012.

[107] Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. In *Conference on Robot Learning*, pages 564–574. PMLR, 2021.

[108] Thomas Lips, Victor-Louis De Gusseme, et al. Learning keypoints for robotic cloth manipulation using synthetic data. *IEEE Robotics and Automation Letters*, 2024.

[109] Peng Zhou, Jihong Zhu, Shengzeng Huo, and David Navarro-Alarcon. Lasesom: A latent and semantic representation framework for soft object manipulation. *IEEE Robotics and Automation Letters*, 6(3):5381–5388, 2021.

[110] Solvi Arnold, Daisuke Tanaka, and Kimitoshi Yamazaki. Cloth manipulation planning on basis of mesh representations with incomplete domain knowledge and voxel-to-mesh estimation. *Frontiers in Neurorobotics*, 16:1045747, 2023.

[111] Xiao Ma, David Hsu, and Wee Sun Lee. Learning latent graph dynamics for deformable object manipulation. *arXiv preprint arXiv:2104.12149*, 2, 2021.

[112] Chenchang Li, Zihao Ai, Tong Wu, Xiaosa Li, Wenbo Ding, and Huazhe Xu. Deformnet: Latent space modeling and dynamics prediction for deformable object manipulation. *arXiv preprint arXiv:2402.07648*, 2024.

[113] Naijing Lv, Jianhua Liu, and Yunyi Jia. Dynamic modeling and control of deformable linear objects for single-arm and dual-arm robot manipulations. *IEEE Transactions on Robotics*, 38(4):2341–2353, 2022.

[114] Dale McConachie, Thomas Power, Peter Mitrano, and Dmitry Berenson. Learning when to trust a dynamics model for planning in reduced state spaces. *IEEE Robotics and Automation Letters*, 5(2):3540–3547, 2020.

[115] Dale McConachie and Dmitry Berenson. Bandit-based model selection for deformable object manipulation. In *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, pages 704–719. Springer International Publishing, 2020.

[116] Thomas Power and Dmitry Berenson. Keep it simple: Data-efficient learning for controlling complex systems with simple models. *IEEE Robotics and Automation Letters*, 6(2): 1184–1191, 2021.

[117] Peng Zhou, Pai Zheng, Jiaming Qi, Chenxi Li, Chenguang Yang, David Navarro-Alarcon, and Jia Pan. Bimanual deformable bag manipulation using a structure-of-interest based latent dynamics model. *arXiv preprint arXiv:2401.11432*, 2024.

[118] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Kumar Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. Visuospatial foresight for multi-step, multi-task fabric manipulation. *arXiv preprint arXiv:2003.09044*, 2020.

[119] Zehang Weng, Fabian Paus, Anastasiia Varava, Hang Yin, Tamim Asfour, and Danica Kragic. Graph-based task-specific prediction models for interactions between deformable and rigid objects. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5741–5748. IEEE, 2021.

[120] Peter Mitrano, Alex LaGrassa, Oliver Kroemer, and Dmitry Berenson. Focused adaptation of dynamics models for deformable object manipulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5931–5937. IEEE, 2023.

[121] Tae Min Lee, Young Jin Oh, and In-Kwon Lee. Efficient cloth simulation using miniature cloth and upscaling deep neural networks. *arXiv preprint arXiv:1907.03953*, 2019.

[122] Yu-Ming Chen and Michael Posa. Optimal reduced-order modeling of bipedal locomotion. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8753–8760. IEEE, 2020.

[123] Sander Tonkens, Joseph Lorenzetti, and Marco Pavone. Soft robot optimal control via reduced order finite element models. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12010–12016. IEEE, 2021.

[124] Andreas Doumanoglou, Jan Stria, Georgia Peleka, Ioannis Mariolis, Vladimir Petrik, Andreas Kargakos, Libor Wagner, Václav Hlaváč, Tae-Kyun Kim, and Sotiris Malassiotis. Folding clothes autonomously: A complete pipeline. *IEEE Transactions on Robotics*, 32 (6):1461–1478, 2016.

[125] Julia Borras, Guillem Alenya, and Carme Torras. A grasping-centered analysis for cloth manipulation. *IEEE Transactions on Robotics*, 36(3):924–936, 2020.

[126] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

[127] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33 (4):1–12, 2014.

[128] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[129] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, Guido Ranzuglia, et al. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136. Salerno, Italy, 2008.

[130] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings. IEEE international conference on multimedia and expo*, volume 1, pages 705–708. IEEE, 2002.

[131] Devin J Balkcom and Matthew T Mason. Robotic origami folding. *The International Journal of Robotics Research*, 27(5):613–627, 2008.

[132] Iram Noreen, Amna Khan, and Zulfiqar Habib. Optimal path planning using rrt* based approaches: a survey and future directions. *International Journal of Advanced Computer Science and Applications*, 7(11), 2016.

[133] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010.

[134] Joo Pedro Pedroso. Optimization with gurobi and python. *INESC Porto and Universidade do Porto,, Porto, Portugal*, 1, 2011.

[135] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3. Kobe, Japan, 2009.

[136] 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–eabd8170, 2021.

[137] Zixuan Huang, Xingyu Lin, and David Held. Mesh-based dynamics with occlusion reasoning for cloth manipulation. *arXiv preprint arXiv:2206.02881*, 2022.

[138] Zisong Xu, Rafael Papallas, and Mehmet R Dogar. Physics-based object 6d-pose estimation during non-prehensile manipulation. In *International Symposium on Experimental Robotics*, pages 181–191. Springer, 2023.

[139] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33. PMLR, 2022.

[140] Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Iterative residual policy: for goal-conditioned dynamic manipulation of deformable objects. *The International Journal of Robotics Research*, 43(4):389–404, 2024.

[141] Yahav Avigal, Lars Berscheid, Tamim Asfour, Torsten Kröger, and Ken Goldberg. Speedfolding: Learning efficient bimanual folding of garments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2022.

[142] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi: 10.1007/s12532-017-0130-5.

[143] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[144] Davide Chicco. Siamese neural networks: An overview. *Artificial neural networks*, pages 73–94, 2021.

[145] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[146] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Appendices

A1 The Teenager's Problem

This project focuses on enabling robots to efficiently declutter piles of cloth—a task closely related to deformable object manipulation and multi-object manipulation, such as collecting laundry before washing. My contribution involves implementing and solving this problem as a probabilistic set cover problem, an idea initially proposed by collaborators. Additionally, I designed and trained a neural network to act as a predictor, determining whether a specific item can be removed based on a candidate grasping position and orientation.

In the following sections, I provide an overview of the project and describe the formulation of the probabilistic set cover problem. I then introduce the neural network-based predictor, covering how I generate data for training and validation, the structure of the neural network, and the testing results.

A1.1 Overview

In this project, we aim to develop a robotic system capable of efficiently grasping and transporting multiple cloth items in a single grasp, rather than moving them one by one. The goal is to minimize the number of grasps required to declutter a pile of clothes, thus speeding up the entire process. To achieve this, the system focuses on grasping as many items as possible in a single attempt, reducing the total number of grasps necessary to clear the table.

We set up an experimental system consisting of a UR5 robotic arm, a RealSense depth camera, and a flat table covered with various cloth items. A digital scale is used to measure the weight of the items deposited into the basket, providing feedback on the progress of the decluttering process and helping determine if the task has been successfully completed. An example scenario of this setup is depicted in Figure 1.

To solve this problem, the system first captures an image of the clothes spread out on the table, selects a sequence of picking poses, then grasps the items and transports them to the basket. The primary performance metric we use to evaluate the method is the average number of objects picked per transport (OpT). This metric allows us to measure how efficiently the robotic system declutters the table in each transport cycle.

A1.2 Set Cover Problem Solving

The Set Cover Problem is a fundamental NP-hard problem in combinatorial optimization, widely studied in theoretical computer science and operations research. Given a universal set U of elements and a collection of subsets S_1,S_2,. . . ,S_n, the objective is to determine the smallest number of subsets whose union equals the entire set U. Formally, the goal is to find a subcollection of the given subsets that covers every element in U while minimizing the number of subsets selected.

The decluttering problem for the set of clothes on the table can be formulated similarly as a mixed integer linear programming (MILP) problem, where the objective is to find the minimum

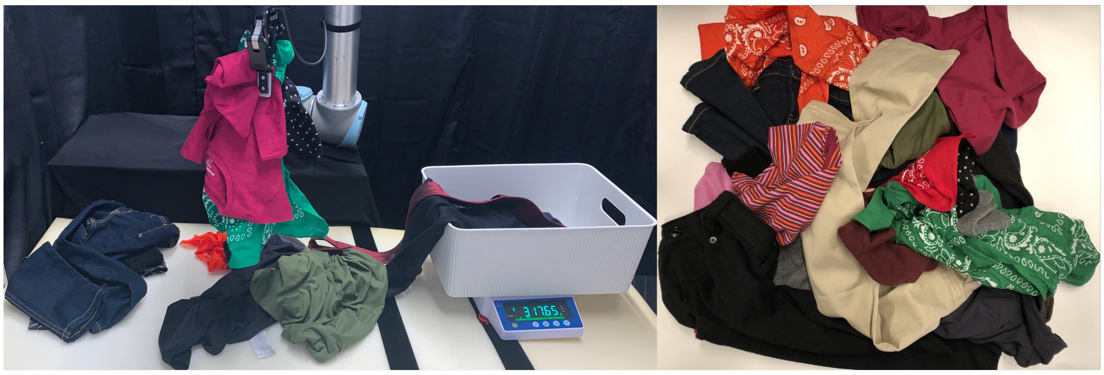


Figure 1: Setup for robot decluttering cloth from table to basket (left) and the set of clothes used for neural network training data collection (right). The above images are taken from Adler et al. [1, 2] and show experiments conducted by our collaborators at UC Berkeley.

number of grasps $\boldsymbol{x}$ that can remove all the clothes while satisfying a set of constraints. This MILP formulation of the problem has been proposed by our collaborators at UC Berkeley [1]. The MILP formulation can be written as follows:

$$\begin{aligned} \min \quad & \mathbf{1}^\top \boldsymbol{x} \\ & \boldsymbol{A}^\top \boldsymbol{x} \leq \boldsymbol{b} \\ s.t. \quad & x_i + x_{i'} \leq 1 \texttt{ for all } i,\ i' \texttt{ which overlap} \\ & x_i \in \{0,1\} \texttt{ for all } i\,. \end{aligned} \tag{1}$$

where $\boldsymbol{x}$ is a binary vector, and $x_i = 1$ if grasp i is selected and $x_i = 0$ otherwise; $\boldsymbol{A}^\top$ represents the probabilities associated with each item being removed by each grasp; $\boldsymbol{b}$ is a vector of thresholds for the probability of successfully removing each item; the constraint $x_i + x_{i'} \leq 1$ ensures that no two grasp positions overlap, meaning that only one grasp can be selected if two positions conflict; the objective function min $\mathbf{1}^\top \boldsymbol{x}$ minimizes the total number of grasps required to remove all the clothes.

This MILP formulation allows for optimizing the decluttering process by selecting the minimal number of grasps necessary to successfully clear the table of all items while avoiding overlap and maintaining a high probability of success. Experiments demonstrate that solving this decluttering problem as a set cover formulation increases OpT by 50% over a random baseline. Other results can be found in the published papers [1, 2].

I implemented the MILP formulation of this problem and used the MILP solver [142] within the SciPy library [143] to solve it. Besides, the first constraint involves calculating the possibilities of each item being removed given a candidate grasp. To compute these probabilities, I trained a neural network as a predictor to estimate the likelihood of removing each item based

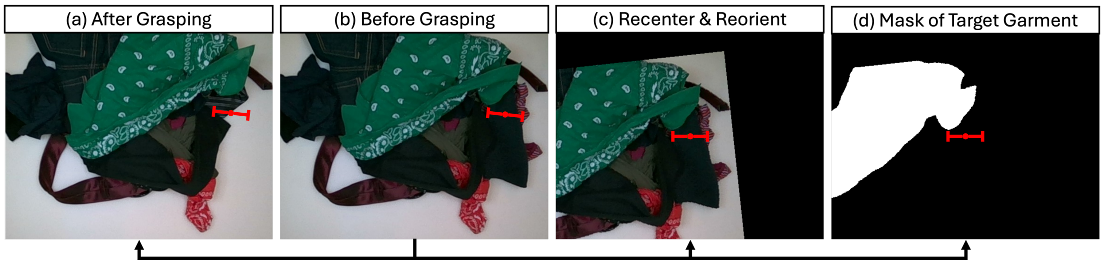


Figure 2: Example of Training Data Generation: The first two frames display the images before and after a grasp, with the red marker indicating the grasping position and orientation. The third frame shows the pre-grasp image recentered and reoriented according to the grasping position and orientation. The fourth frame presents the mask of the target garment, which is labeled as "removed" or "unremoved". In this case, based on the post-grasp image in (a), the item is determined to be removed. The last two frames—comprising the reoriented image and the mask—serve as inputs to the neural network, with the label set to 0, representing "unremoved".

on the selected grasp position and orientation, which is detailed in the next section.

A1.3 Neural Network based Predictor

Deformable objects such as garments are difficult to model in simulation. Therefore, I train the predictor entirely on real-world data through self-supervision.

I conduct segmentation first, and then, given a proposed grasp, I aim to determine which items are most likely to be picked up and which are not, thus assisting the choice of the grasping pose to maximize the number of items being transferred. To achieve this, I train a neural network model using data collected from real robot experiments and a segmentation-based autolabeler.

A1.3.1 Data Generation

The original data for this project comes from previous robot decluttering experiments. In this setup, a laundry basket is placed on a digital scale to measure the weight of the garments grasped by the robot. The procedure begins with the robot resetting the scene by dumping garments from the laundry basket onto the working surface and shuffling them. Then, a random pick point is sampled from the garment points in the RGB image, along with a grasping orientation. The robot executes the grasp and places the grasped garments into the laundry basket. After each manipulation, RGB images are recorded both before and after the grasp.

To build a labeled dataset, we preprocess the data using the Segment Anything Model (SAM) to automatically generate segmentation masks. This allows us to label each instance by checking whether a grasped item has been removed. For example, as shown in Figure 2, if a masked sock disappears from the post-manipulation RGB image, it is labeled as "removed"; if the item remains, it is labeled as "unremoved."

We encode the grasping position and orientation into the dataset by transforming the images so that the grasping point is centered and the image is rotated according to the grasp orientation, as shown in Figure 2. This approach ensures that the neural network is trained to recognize the grasp position and direction effectively. In total, 236 decluttering scenarios are sampled, resulting in a dataset of 3,304 RGB image and grasp pairs.

To enhance the ability of the predictor to generalize across different environments, several data augmentation techniques were applied: the image is flipped both horizontally and vertically without changing the original label due to the symmetry of grasping; the brightness, contrast, saturation, and hue of the original image are randomly changed accounting for different scenarios; the size of the original image is randomly cut around the center in case the position of the camera may be changed.

A1.3.2 Neural Network Model

The neural network model, as shown in Figure 3, is built upon the Siamese [144] framework and the widely used Resnet [145] pre-trained on the large ImageNet dataset [146] for classification. The inputs of the model include the original RGB image and the mask image of the item that we are interested in. The weights of Resnet are fixed and shared during feature extraction for both images. After that, the features are concatenated and fed into a multi-layer perceptron model for comparison and predicting the probability of the masked item being grasped.

We use the RGB image and the mask image as inputs because the mask image identifies which object we are interested in while the original RGB image accounts for the impacts of neighboring items around the mask. This is particularly suitable for assessing the effectiveness of grasping actions in intricate and crowded environments where the number of garments remaining on the working surface varies from scene to scene.

A1.3.3 Training and Testing

Based on the collected image pairs and the labels, I train the neural network model using binary entropy loss and the Adam optimizer with a learning rate of 10^{-5} for 500 epochs. In this way, the possibility of each item on the platform being removed or not can be predicted given an overhead RGB image, the mask of the interested item, and a proposed grasping pose.

During training, the loss curves for both the training and testing sets are shown in Figure 4, and the prediction accuracy is displayed in Figure 5. The model achieves an accuracy of around 76%, which, while reasonable, indicates that the task is quite challenging. One potential reason for the suboptimal accuracy could be the relatively small dataset, as well as noisy data generated by the imperfection of the autolabeller.

In a real robot experiment using this neural network predictor, we achieved an OpT of 1.70.

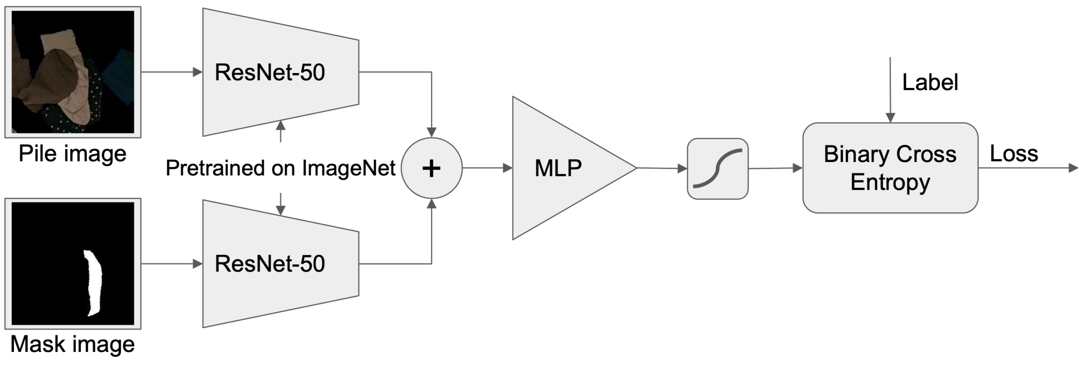


Figure 3: Neural network structure for learning the probability predictor: the network consists of two pre-trained ResNet-50 models and a multi-layer perceptron. The input includes an RGB image and a mask, while the output predicts the probability of the masked item being removed given the current candidate grasp.

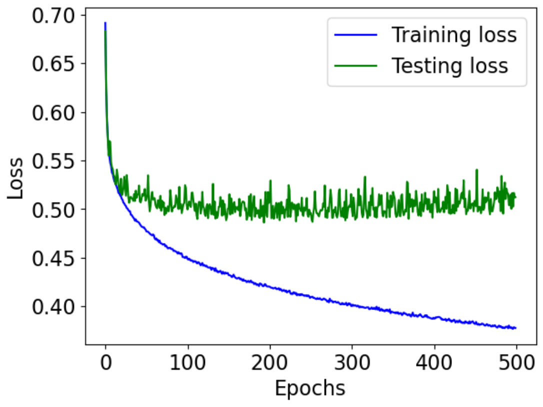


Figure 4: The training and testing loss for the neural network based predictor.

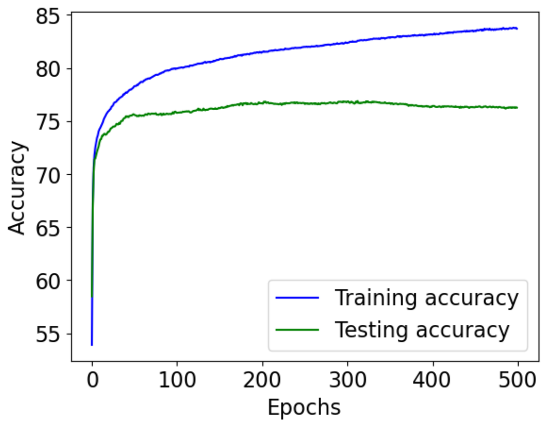


Figure 5: The training and testing prediction accuracy.

A2 Neural Network based Cloth Folding

I also conduct some preliminary experiments of bimanual cloth folding using learning-based methods. One key challenge with such methods is the vast amount of data required to train the neural network due to the high dimensionality of the configuration space. To mitigate this, I utilize the proposed model simplification method to extract a succinct representation of the original high-dimensional model.

I employ the reduced action space and design a hand-crafted policy using two pickers to generate training data. This approach allows me to explore whether the simplified representation of the deformable object provides sufficient information for the neural network to effectively learn the desired manipulation strategies.

The cloth folding task is used as an illustrative example. I provide details on the hand-crafted folding policy, the data generation process for policy learning, and the neural network training procedure. Finally, I present the testing results of the learned neural network-based manipulation policy on a novel cloth folding task, which involves folding cloths of different sizes.

A2.1 Hand-Crafted Cloth Folding Policy

I use the cloth side folding task in SoftGym as an exemplary task. Since I do not have the necessary permissions from Nvidia to deploy robot models in the environment, I use two floating spheres moving within a constrained space as substitutes for end effectors. The assumption is that when the distance between a sphere and any particle on the cloth mesh falls below a predefined threshold, the nearest particle can be “grasped” or anchored by the sphere upon command.

Based on this simplified “end-effector” model, I use speed vectors as control signals to move the spheres to desired positions. The cloth model in the environment consists of particles and is initially placed on a flat surface in its natural, spread-out state.

Inspired by the simplest way a human might fold a towel with two hands, I design a co-operative manipulation policy for the two sphere-shaped grippers, consisting of four steps, as shown in Figure 6.

– Approaching Phase: The two pickers move toward positions directly above the left two corners of the cloth. During this phase, the distance discrepancy between the two pickers is taken into account to ensure that they arrive at their target positions simultaneously.

– Grasping Phase: The pickers lower themselves until the distance between them and the cloth corners is below a predefined threshold, signaling that the particles at the corners can be grasped by the floating spheres.

– Transporting Phase: Once the particles are grasped, the pickers move along predefined trajectories to bring the cloth corners to target positions above the opposite corners.

– Releasing Phase: Finally, the grippers slowly descend and release the cloth corners, completing the folding task.

The action at each time step t consists of a velocity vector in 3D space and an indicator for grasping or releasing for each gripper. The velocity is constrained to ensure that the pickers do not move too fast, and the velocities of the two pickers are synchronized so that they reach the target positions simultaneously. Thus, the action control signal at each time step is represented as an eight-dimensional vector:

$$a_t = [v_{x,1}, v_{y,1}, v_{z,1}, g_1, v_{x,2}, v_{y,2}, v_{z,2}, g_2] \tag{2}$$

where $v_{x,i}$,$v_{y,i}$,$v_{z,i}$ denote the velocities in the x, y, and z directions for picker i, and g_i indicates whether picker i is grasping ($g_i = 1$) or releasing ($g_i = 0$).

Following the folding procedure and control rule proposed above, a piece of cloth can be successfully folded in half. An illustration of the full process is shown in Figure 7, moving from top left to bottom right. The sequence includes the following stages: the initial state, where the cloth is fully spread out on the flat surface in its natural, unfolded state; the approaching state, where the two pickers move to positions above the left two corners of the cloth, preparing to grasp; the grasped and nearly folded state, where the pickers have grasped the cloth corners and are midway through the folding process, bringing the corners toward their target positions; and the final state, where the cloth is successfully folded in half, with the two corners placed near their new positions. This sequence effectively demonstrates how the hand-crafted policy can manipulate and fold the cloth using the floating spheres as grippers.

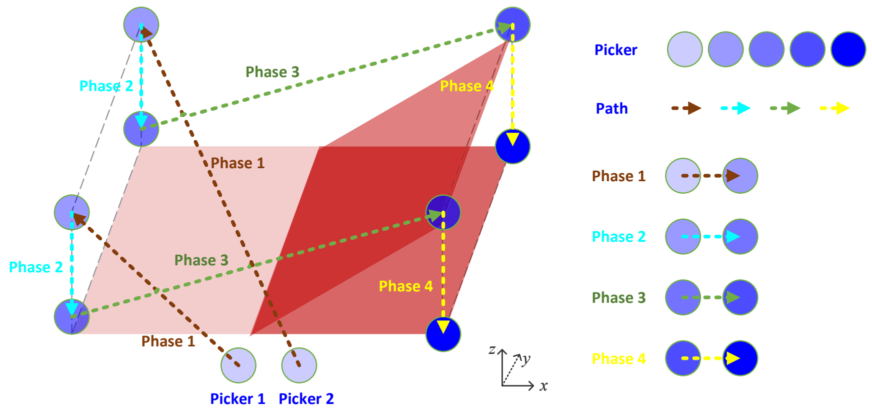


Figure 6: Illustration of the hand-crafted bimanual cloth folding policy.

A2.2 Data Collection for Policy Learning

Based on the hand-crafted folding policy described above, I generate data for training, validation, and testing by randomly creating a variety of cloth folding scenarios. The size of the cloth is randomly selected, with the width ranging from 0.5m to 0.8m and the length from 0.5m to 0.8m. The cloth is placed flat in the center of the workspace at a random angle in each scenario.

To reduce the complexity of training in the image space, the state of the cloth is represented as the concatenation of the positions of selected particles, following the geometric model simplification proposed earlier. Additionally, the positions of the two grippers are included to represent the state of the environment. This provided a concise, 24-dimensional vector representing the state at each time step.

I generate 100 cloth folding scenarios, using 80% for training and validation and the remaining 20% as the test set. Solving these tasks with the hand-crafted policy results in a collection of 2,000 state-action pairs, which are used to train the neural network model.

A2.3 Neural Network Model

To learn the folding policy, I build a five-layer feed-forward neural network, as illustrated in Figure 9. The network starts with an input layer consisting of 24 neurons, which represent the state vector of the cloth and the positions of the two grippers. The following three layers are hidden layers, with 300, 200, and 100 neurons, respectively. For the hidden layers, I use the sigmoid activation function for all neurons. The final layer is the output layer, which contains 8 neurons corresponding to the control signals for the two grippers. The activation function for the output layer is linear, as the control signals require continuous values to guide the movements of the grippers.

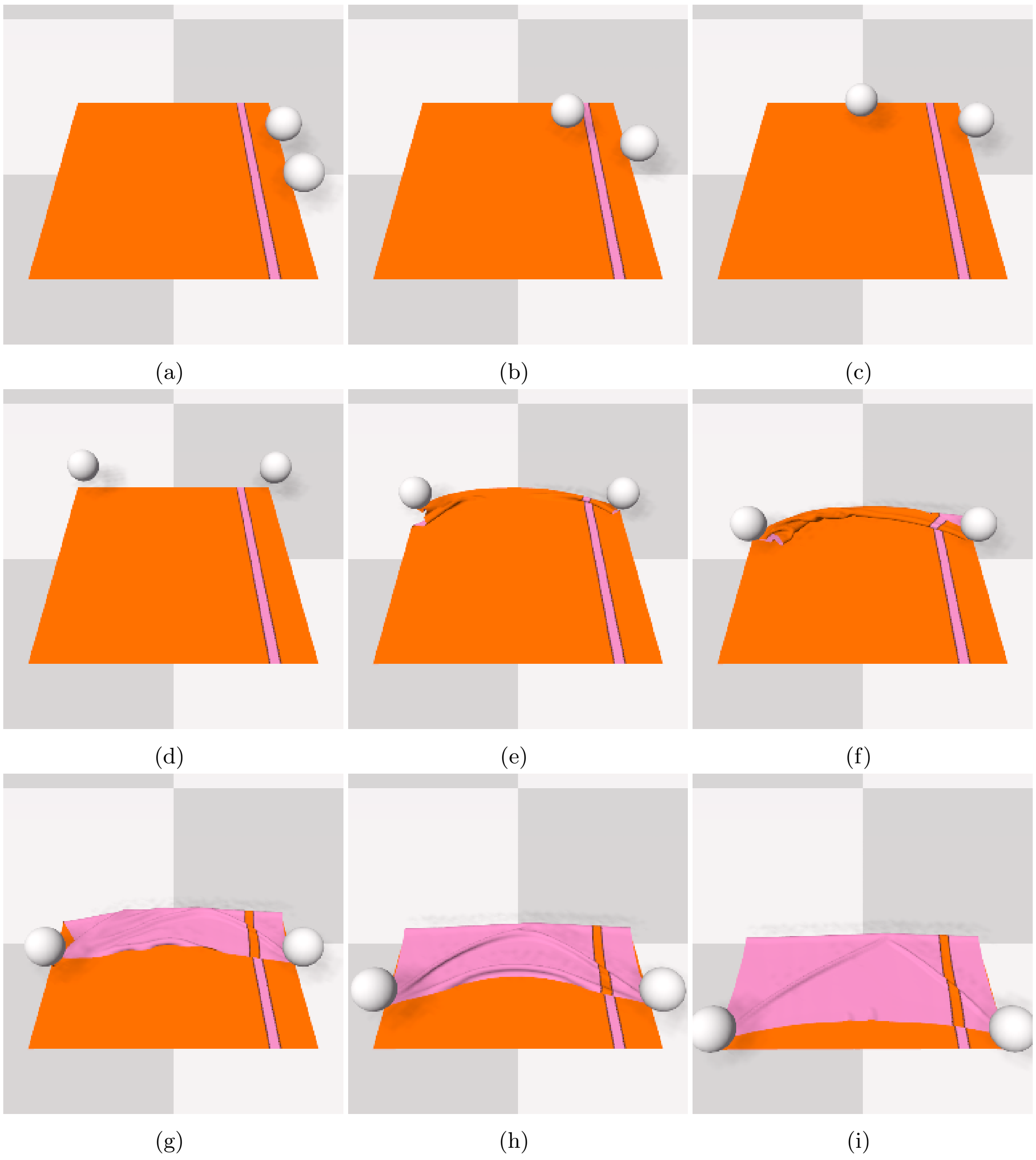


Figure 7: Cloth folding process using the hand-coded bimanual manipulation policy

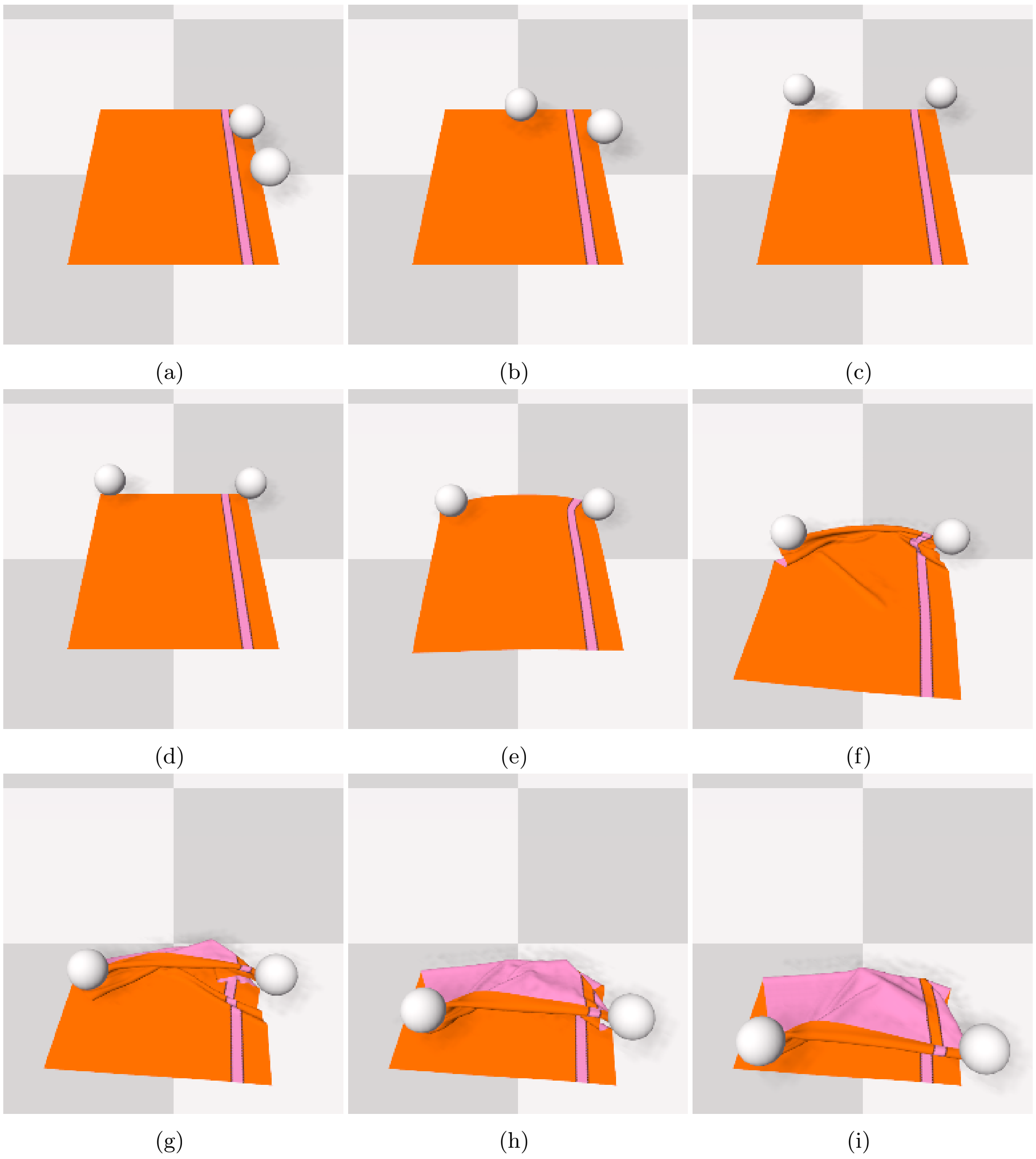


Figure 8: Cloth folding process using the trained bimanual manipulation policy.

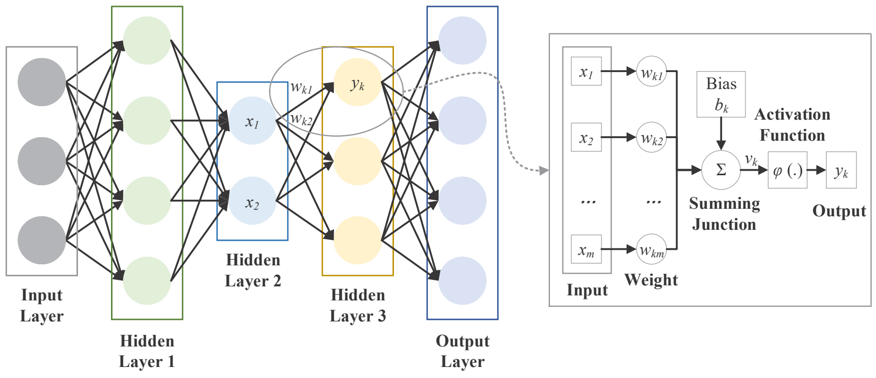


Figure 9: The neural network architecture for bimanual cloth folding policy learning.

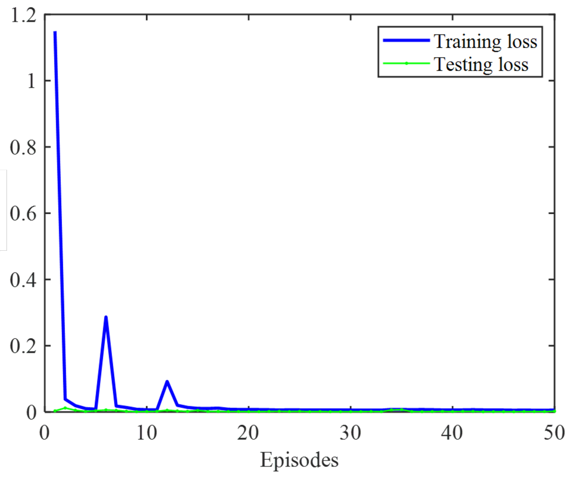


Figure 10: The loss curves of training and testing

A2.4 Neural Network Training and Testing

After constructing the neural network as a regression model to learn the bimanual folding policy for the cloth folding task, I used the Mean Squared Error (MSE) as the loss function, which calculates the error between the ground truth action values and the predicted action values. To optimize the model, I selected RMSProp, an adaptive learning rate optimization method, with a learning rate l_r of 0.001. The batch size for each training iteration is set to 128. After training the network using the collected state-action pairs for 50 epochs, the model converges, providing a neural network-based folding policy. The training and testing loss curves throughout the training process are shown in Figure 10.

Once the neural network is trained and has converged, I deploy it to solve the cloth folding tasks in the testing set. To illustrate the performance of the learned policy, I randomly select a cloth from the testing set and show the folding process in Figure 8. The figure presents the sequence of steps taken by the neural network-based bimanual folding policy to manipulate the cloth into the folded configuration.

From the testing results, we can see that the trained neural network successfully folds the cloth in half, demonstrating the effectiveness of the data-driven cloth folding framework. The neural network is able to represent a feasible policy for manipulating deformable objects, suggesting that similar network structures could be applied in deep reinforcement learning for future research. However, the learned neural network policy tends to move the pickers close to the cloth surface, which occasionally causes wrinkles. This behavior arises because the demonstrations provided in the training set are highly skilled, making the cloth's state easily disturbed by small deviations. Improving the robustness of the policy to handle more varied states and disturbances could be an area for future improvement.