Seeing the Unseen: Closed-loop Occlusion Reasoning for Cloth Manipulation

Zixuan Huang

CMU-RI-TR-22-32

July 2022

Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Prof. David Held, Chair
Prof. Shubham Tulsiani
Alex LaGrassa

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Keywords: Deformable Object Manipulation, 3D reconstruction, Self-supervised Learning, sim-to-real transfer

For my family, friends, and teachers.

Abstract

Robotic manipulation of cloth remains challenging due to the complex dynamics of cloth, lack of a low-dimensional state representation, and self-occlusions. Particularly, self-occlusion makes it difficult to estimate the full state of the cloth, which poses significant challenges to policy learning and dynamics modeling. Ideally, a robot trying to unfold a crumpled or folded cloth should be able to reason about the cloth's occluded regions. In this thesis, we aim to investigate different strategies of enabling the robot to reason about occlusion in a closed-loop manner.

In the first project, we propose to learn a particle-based dynamics model from a partial point cloud observation. To overcome the challenges of partial observability, we infer which visible points are connected on the underlying cloth mesh. We then learn a dynamics model over this visible connectivity graph. Since the partial point cloud only captures the structure of visible cloth, we design a graph imitation learning method that reason about occlusion implicitly. Compared to previous learning-based approaches, our model poses strong inductive bias with its particle based representation for learning the underlying cloth physics; it can generalize to cloths with novel shapes; it is invariant to visual features; and the predictions can be more easily visualized.

In the second project, we explore explicit occlusion reasoning via cloth reconstruction. We leverage recent advances in pose estimation for cloth to build a system that uses explicit occlusion reasoning to unfold a crumpled cloth. Specifically, we first learn a model to reconstruct the mesh of the cloth. However, the model will likely have errors due to the complexities of the cloth configurations and due to ambiguities from occlusions. Our main insight is that we can further refine the predicted reconstruction by performing test-time finetuning with self-supervised losses. The obtained reconstructed mesh allows us to use a mesh-based dynamics model for planning while reasoning about occlusions. We evaluate MEDOR (MEsh-based Dynamics with Occlusion Reasoning) both on cloth flattening as well as on cloth canonicalization, in which the objective is to manipulate the cloth into a canonical pose. Our experiments show that our method significantly outperforms prior methods that do not explicitly account for occlusions or perform test-time optimization.

Although MEDOR produces impressive reconstruction results in simulation, we observe a huge sim-to-real gap when deploying it to real world. In the third project, we propose a self-supervised method to finetune a mesh reconstruction model in the real world. Since the full mesh of crumpled cloth is difficult to obtain in the real world, we design a special data collection scheme and an action-conditioned model-based cloth tracking method to generate pseudo-labels for self-supervised learning. By finetuning the pretrained mesh reconstruction model on this pseudo-labeled dataset, we show that we can improve the quality of the reconstructed mesh without requiring human annotations.

Acknowledgments

First and foremost, I'm deeply grateful to my advisor, David Held, who has consistently supported me for the past two years. I am fortunate to have Dave's thoughtful guidance throughout the master's study. The weekly meeting is the most enjoyable and rewarding part, where Dave always shares his valuable insights and teach me to conduct research in a systematic way.

I would also like to thank my mentor, Xingyu Lin, who is a role model that I look up to. Whenever I'm stuck with some problems, he always inspired me with wonderful suggestions and ideas. I thank Professor Shubham Tulsiani and Alex LaGrassa for being my committee members and providing helpful feedbacks.

It is also my great fortune to meet and work with a group of nice people at CMU and RPAD - Fan Yang, Carl Qi, Yufei Wang, Bowei Chen, Cindy Wu, Xinjie Yao, Chuer Pan, Tianyuan Zhang, Gaurav Pathak, Sarthak Shetty, Sashank Tirumala, Brian Okorn, Ben Eisner, Wenxuan Zhou and Thomas Weng. Particularly, I would like to thank Harry Zhang for offering ride when we both stayed late before the paper deadline. I give my sincere appreciation for all your kind help during my study here.

At last, I would like to thank my parents for their continuous love and support along every step in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic manipulation of cloth remains difficult due to several reasons. Particularly, state estimation poses a specific challenge due to the deformability, high dimensional state representation, and self-occlusions. In this thesis, we study the problem of occlusion and investigate several strategies to tackle it.

Why do we need to tackle with self-occlusion? Ideally, a robot trying to unfold a crumpled or folded cloth should be able to reason about the cloth's occluded regions. For example, suppose the robot try to unfold a shirt whose sleeve is folded below, if the robot doesn't have that piece of information, it cannot come up with the action to reveal it. Our assumption is that occlusion reasoning provide useful information for downstream cloth manipulation task.

Why is self-occlusion a difficult problem for cloth? For rigid object, the configuration of the occluded region can be easily determined by the visible part. However, since cloth is "deformable", the configuration of occluded part can be arbitrary. Another consequence of the deformability is the high dimensional representation required for specifying the full configuration, which makes it a challenging task for learning-based method.

In this thesis, we investigate two possible solutions: implicit occlusion reasoning and explicit occlusion reasoning by reconstruction.

- In the chapter [2], we introduce VCD (Visible Connectivity Dynamics), a particle-based dynamics model on partial point cloud. The particle-based dynamics model captures the inductive bias of the underlying physics dynamics, thus naturally generalize to clothes of different shapes. Since the partial point cloud only contains information of the visible surface, we propose to use graph imitation learning to implicitly reason the effect of occluded part. Our experiment shows that implicit occlusion reasoning can improve the accuracy of learned dynamics model.
- However, implicit reasoning is insufficient for tasks like cloth unfolding, so we further investigate the role of explicit occlusion reasoning by reconstructing the whole cloth. While prior works [54, 59, 72, 62, 61, 60, 19] also attempt to estimate the state of the occluded regions, they only allow open-loop cloth manipulation. In chapter [3], we propose MEDOR to reconstruct the whole cloth in close-loop manner by self-supervised test-time finetuning. We show the explicit reconstruction reduce the rollout error of dynamics model, and provide more informative reward signal for planning
- One caveat of MEDOR is that it is fully trained in the simulation, thus suffers from a sim-to-

real gap when deployed in real world. Training cloth reconstruction model in real world is challenging due to the difficulty of obtaining ground-truth label. In chapter [4], we proposed a method for self-supervised cloth reconstruction in real world by using cloth tracking.

Chapter 2

Learning Visible Connectivity Dynamics for Cloth Smoothing

2.1 Introduction

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

One approach to cloth manipulation explored by previous work, which we also adopt, is to learn a cloth dynamics model and then use the model for planning to determine the robot actions. However, given that a crumpled cloth has many self-occlusions and complex dynamics, it is unclear how to choose the appropriate state representation. One possible state representation is to use a mesh model of the entire cloth [50]. However, fitting a full mesh model to an arbitrary crumpled cloth configuration is difficult. Recent work have approached fabric manipulation by either compressing the cloth representation into a fixed-size latent vector [127, 126, 73] or directly learning a visual dynamics model in pixel space [38]. However, these representations do not enforce any inductive bias of the cloth physics, leading to suboptimal performance and generalization.

In contrast to a pixel-based or latent dynamics model, particle-based models have recently been shown to be able to learn dynamics for fluid and plastics [63, 97, 88]. A particle-based dynamics representation has the following benefits: first, it captures the inductive bias of the underlying physics, since real-world objects are composed of underlying atoms that can be modeled on the micro-level by particles. Second, we can incorporate inductive bias by directly applying the effect of the robot gripper on the particle being grasped (though the effect on the other particles must still be inferred). Last, particle-based models are invariant to visual features such as object colors or patterns. As such, in this paper we aim to learn a particle-based dynamics model for cloth. However, the challenges in applying the particle-based model to cloth are that we cannot directly observe the underlying particles composing the cloth nor their mesh connections. The problem is made even more challenging due to the partial observability of the cloth from self-occlusions when it is in a crumpled configuration.

Our insight into this problem is that, rather than fitting a mesh model to the observation, we

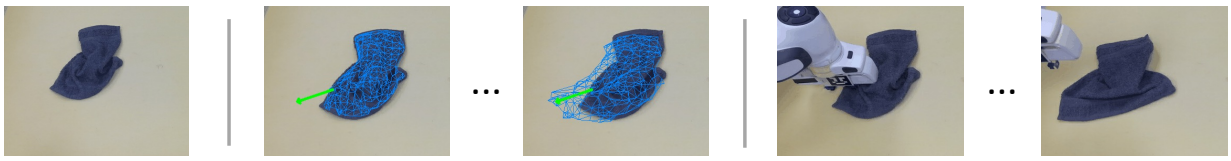


Figure 2.1: Cloth smoothing by planning using a dynamics model with a visible connectivity graph.

should learn the *visible connectivity dynamics (VCD):* a dynamics model based on the connectivity structure of the visible portion of the cloth. To do so, we first learn to estimate the *visible connectivity graph*: we estimate which points in the point cloud observation are connected in the underlying cloth mesh (see Figure 2.1). Estimating the mesh connectivity of the observation is a simplification of the problem of fitting a single full mesh model of the entire cloth to the observation; however, it is significantly easier to learn, since we do not need to find a globally consistent explanation of the observation which requires reasoning about occlusion; to estimate the mesh connectivity of the observation, we only need to consider the visible local cloth structure. While the graph is constructed only based on the visible points, we show that the dynamics model can be trained to be robust to partial observation.

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

2.2 Related Work

Vision-based Cloth Manipulation: Some papers on cloth manipulation assume that the cloth is already lying flat on the table [77, 105, 106]. If the cloth starts in an unknown configuration, then one approach is to perform a sequence of actions that are designed to move the cloth into a set of known configurations from which perception can be performed more easily [25, 71, 119]. For example, the robot might first grasp the cloth by an arbitrary point and raise it into the air; it can then detect the lowest point, either while the cloth is held in the air [25, 85, 52, 53, 54, 72] or after throwing the cloth on the table [119]. By constraining the cloth to this configuration set, the task of perceiving the cloth or fitting a mesh model [50] is greatly simplified. However, these funneling actions are usually scripted and are not generalizable to different cloth shapes or configurations. In contrast, our work aims to enable a robot to interact with cloth from arbitrary configurations and shapes.

Other early works designed vision systems for detecting cloth features that can be used for downstream tasks, such as a Harris Corner Detector [124] or a wrinkle-detector [111]. More examples of such approaches are described in [50]. However, these approaches require a task-

specific manual design of vision features and are typically not robust to different variations of the cloth configuration.

Policy Learning for Cloth Manipulation: Recently, there have been a number of learning based approaches to cloth folding and smoothing. One approach is to learn a policy to achieve a given manipulation task. Some papers approach this using learning from demonstration. The demonstrations can be obtained using a heuristic expert [100] or a scripted sequence of actions based on cloth descriptors [30]. Another approach to policy learning is model-free reinforcement learning (RL), which has been applied to cloth manipulation [73, 126, 57]. However, policy learning approaches often lack the ability to generalize to novel situations; this is especially problematic for cloth manipulation in which the cloth can be in a wide variety of crumpled configurations. We compare our method to a state-of-the-art policy learning approach [126] and show greatly improved performance.

Model-based RL for Cloth Manipulation: Model-based RL methods learn a dynamics model and then use it for planning. Model-based reinforcement learning methods have many benefits such as sample efficiency, interpretability, and generalizability to multiple tasks. Previous works have tried to learn a pixel-based dynamics model that directly predict the future cloth images after an action is applied [28, 38]. However, learning a visual model for image prediction is difficult and the predicted images are usually blurry, unable to capture the details of the cloth. Another approach is to represent the cloth with a fixed-size latent vector representation [127] and to plan in that latent space. However, cloth has an intrinsic high dimension state representation; thus, such compressed representations typically lose the fine-grained details of their environment and are unsuitable for capturing the low-level details of the cloth's shape, such as folds or wrinkles, which can be important for folding or other manipulation tasks. Our method also falls into the model-based RL category; unlike previous works, we learn a particle based dynamics model [63, 97], which can better capture the cloth dynamics due to the inductive bias of the particle representation. Additionally, the particle representation is invariant to visual features and enables easier sim-to-real transfer.

2.3 Method

An overview of our method, *VCD*(Visible Connectivity Dynamics), can be found in Figure 2.2. We represent the cloth using a Visible Connectivity Graph, in which we connect points of a partial point cloud with nearby edges and the inferred mesh edges. Next, we learn a dynamics model over this graph, and finally we use this dynamics model for planning robot actions.

2.3.1 Graph Representation of Cloth Dynamics

We represent the state of a cloth with a graph $\langle V, E \rangle$. The nodes $V = \{v_i\}_{i=1...N}$ represent the particles that compose the cloth, where $v_i = (x_i, \dot{x}_i)$ denotes the particle's current position and velocity, respectively. There are two types of edges E in the graph, representing two types of interactions between the particles: mesh edges and nearby edges. The mesh edges, E^M, represent the connections among the particles on the underlying cloth mesh. The mesh connectivity is determined by the structure of the cloth and does not change throughout time. Each edge $e_{ij} = (v_i, v_j) \in E^M$ connects nodes v_i to v_j and models the mesh connection between them. The other type of edges

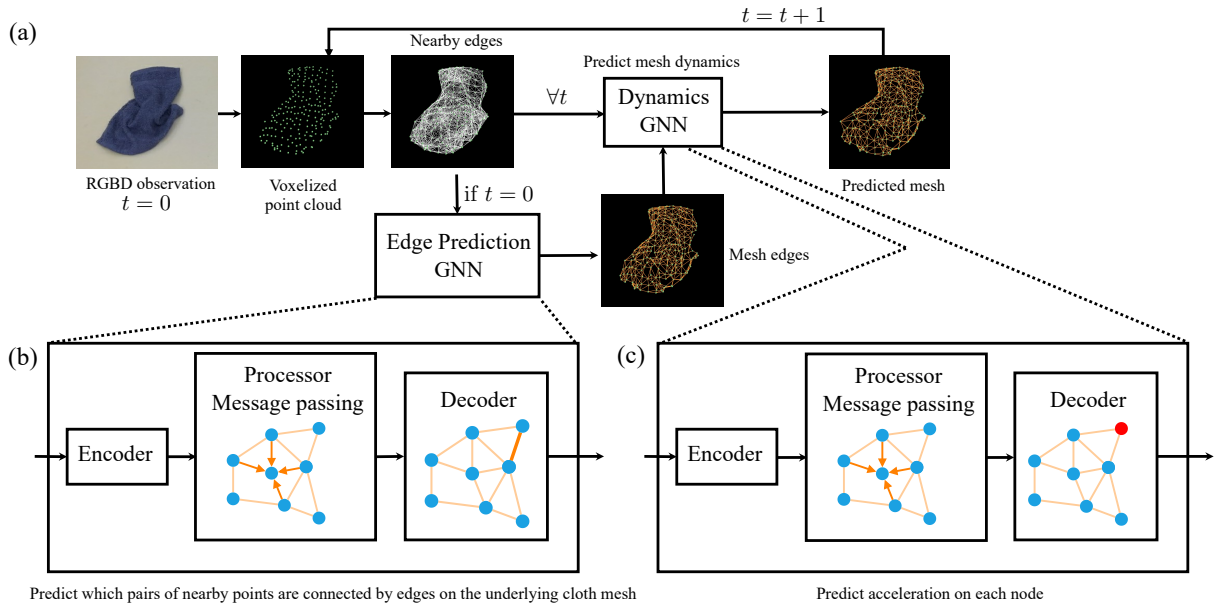


Figure 2.2: (a) Overview of our visible connectivity dynamics model. It takes in the voxelized point cloud, constructs the mesh and predicts the dynamics for the point cloud. (b) Architecture for the edge prediction GNN which takes in the point cloud connected by the nearby edges and predicts for each nearby edge whether it is a mesh edge. (c) Architecture for the dynamics GNN which takes in the point cloud connected by both the nearby edges and the mesh edges and predict the acceleration of each point in the point cloud.

are nearby edges, E^C, which model the collision dynamics among two particles that are nearby in space. These can be different from the mesh edges due to the folded configuration of the cloth, which can bring two particles close to each other even if they are not connected by a mesh edge. Unlike the mesh edges which stay the same throughout time, these nearby edges are dynamically constructed at each time step based on the following criteria:

$$E_t^C = \{e_{ij} | \; ||x_{i,t} - x_{j,t}||_2 < R\}, \tag{2.1}$$

where R is a distance threshold and $x_{i,t}, x_{j,t}$ are the positions of particles i, j at time step t. Throughout the paper, we use the subscript t to denotes the state of a variable at time step t if the variable changes with time. Additionally, we assume that $E^M \subset E^C$, since a mesh edge connects nodes that are close to each other and hence should also satisfy Eqn. 2.1.

2.3.2 Inferring Visible Connectivity from a Partial Point Cloud

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

We create a graph node v_i for each point x_i in the voxelized point cloud P. The nearby edges are then constructed by applying the criterion from Eqn. 2.1. However, inferring the mesh edges is less straightforward, since in the real world we cannot directly perceive the underlying cloth mesh connectivity. To overcome this challenge, we use a graph neural network (GNN) [5] to infer the mesh edges from the voxelized point cloud. Given the positions of the points in P, we first construct a graph $\langle P, E^C \rangle$ with only the nearby edges based on Eqn. 2.1. As we assume $E^M \subset E^C$, we then train a classifier, which is a GNN, to estimate whether each nearby edge $e \in E^C$ is also a mesh edge. We denote this edge GNN as G_{edge}. The edge GNN takes as input the graph $\langle P, E^C \rangle$, propagates information along the graph edges in a latent vector space, and finally decodes the latent vectors into a binary prediction for each edge $e \in E^C$ (predicting whether the edge is also a mesh edge). For the edge GNN, we use the network architecture in previous work [97] (referred to as GNS). See Appendix A.1 for the detailed architecture. The edge GNN is trained in simulation, where we obtain labels for the mesh edges based on the ground-truth mesh of the simulated cloth. After training, it can then be deployed in the real world to infer the mesh edges from the point cloud. We defer the description of how we obtain the ground-truth mesh labels in Sec. 2.3.5.

2.3.3 Modeling Visible Connectivity Dynamics with a GNN

In order to predict the effect of a robot's action on the cloth, we must model the cloth dynamics. While there exists various physics simulators that support simulation of cloth dynamics [24, 64, 82], applying these simulators for a real cloth is still challenging due to two difficulties: first, only a partial point cloud of a crumpled cloth is observed in the real world, usually with many self-occlusions. Second, the estimated mesh edges from Sec. 2.3.2 may not all be accurate. To handle these challenges, we learn a dynamics model based on the voxelized partial point cloud and its inferred visible connectivity (Sec. 2.3.2). Formally, given the cloth graph $G_t = \langle V, E \rangle$, a dynamics GNN G_{dyn} predicts the particle accelerations in the next time step, which can then be integrated to update the particle positions and velocities. Here, V refers to the voxelized point cloud, and E refers to inferred visible connectivity that includes both the predicted mesh edges E^M as well as the nearby edges. Our dynamics GNN G_{dyn} uses the similar GNS architecture as the G_{edge}. It takes a cloth mesh as input with state information on each node, propagates the information along the graph edges in a latent vector space, and finally decodes the latent vectors into the predicted acceleration on each node. See Appendix A.1 for the detailed architecture of the GNN.

2.3.4 Planning with Pick-and-place Actions

We plan in a high-level, pick-and-place action space over the VCD model. For each action $a = \{x_{pick}, x_{place}\}$, the gripper grasps the cloth at x_{pick}, moves to x_{place}, and then drops the cloth. As the GNN dynamics model is only trained to predict the changes of the particle states in small time intervals in order to accurately model the interactions among particles, we decompose each high-level action into a sequence of low-level movements, where each low-level movement is a small delta movement of the gripper and can be achieved in a short time. Specifically, we generate a sequence of small delta movements $\Delta x_1, ..., \Delta x_H$ from the high-level action, where $x_{pick} + \sum_{i=1}^{H} \Delta x_i = x_{place}$. Each delta movement Δx_i moves the gripper a small distance along the pick-and-place direction and the motion can be predicted by the dynamics GNN in a single step.

When the gripper is grasping the cloth, we denote the picked point as u. We assume that the picked point is rigidly attached to the gripper; thus, when considering the effect of the t^{th} low-level movement of the robot gripper, we modify the graph by directly setting the picked point u's position $x_{u,t} = x_{pick} + \sum_{i=1}^{t} \Delta x_i$ and velocity $\dot{x}_{u,t} = \Delta x_i / \Delta t$, where Δt is the time for one low-level movement step. The dynamics GNN will then propagate the effect of the action along the graph when predicting future states. For the initial steps where the historic velocities are not available, we pad them with zeros for input to the dynamics GNN. If no point is picked, e.g., after the gripper releases the picked point, then the dynamics model is rolled out without manually setting any particle state.

The objective of the task to smooth a piece of cloth from a crumpled configuration. To compute the reward r based on either the observed or the predicted point cloud, we treat each point in the point cloud as a sphere with radius R and compute the covered area of these spheres when projected onto the ground plane. Due to computational limitations, we greedily optimize this reward over the predicted states of the point cloud after a one-step high-level pick-and-place action rather than optimizing over a sequence of pick-and-place actions. Given the current voxelized point cloud of a crumpled cloth P, we first estimate the mesh edges using the edge predictor $E^M = G_{edge}(\langle P, E^C \rangle)$. We keep the mesh edges fixed throughout the rollout of a pick-and-place trajectory since the structure of the cloth is fixed. In theory, it could be helpful to update the mesh edges based on the newly observed point cloud at each low-level step, but this is challenging due to the heavy occlusion from the robot's arm during the execution of a pick-and-place action. After the execution of each pick-and-place action, new particles may be revealed and we update the mesh edges when re-planning the next action. The pseudocode of the planning procedure can be found in the appendix.

2.3.5 Training in Simulation

The simulator we use for training is Nvidia Flex, a particle-based simulator with position-based dynamics [78, 70], wrapped in SoftGym [64]. In Flex, a cloth is modeled as a grid of particles, with spring connections between particles to model the bending and stretching constraints.

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

=

2.3.6 Graph-imitation Learning for Occlusion Reasoning

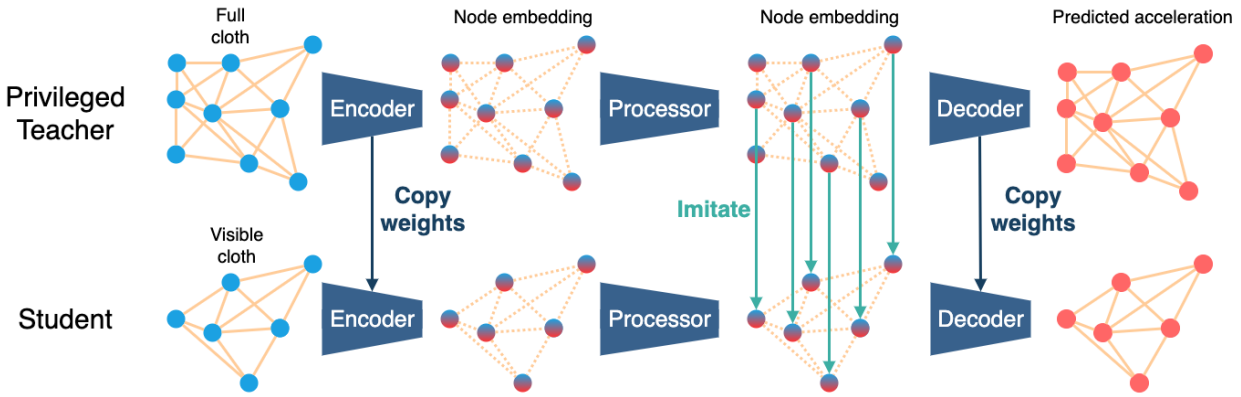


Figure 2.3: A graphical illustration of privileged graph imitation learning. The privileged teacher has the same model architecture as student, but takes full cloth as input. Following [56], we initialize the encoder and decoder of student model by weights of pretrained teacher. Then we freeze the teacher and transfer the privileged information by matching the node embedding and global embedding of two models. The target nodes to imitate are obtained by bi-partite matching as described in A.1.2.

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

Therefore, we introduce graph-imitation learning to inject prior knowledge of the full cloth into the dynamics model. The prior encodes structure of the full cloth and incentivizes the model to reason about occlusion implicitly.

Privileged Graph-imitation Learning

The main spirit of privileged graph-imitation learning is to train a student model which takes partial point cloud as input, to imitate a privileged agent which has access to privileged information. We hope the student to learn a recover function that recovers true states from partial information. A visual illustration is shown in Figure 2.3.

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

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

Once we have the node correspondence, we retrieve the intermediate node features of both teacher and student model, h_T^L and h_S^L, and force the node feature of student h_S^L to be similar to h_T^L. The final output is still supervised by groundtruth acceleration. We copy the weights of encoder and

decoder from teacher model to initialize student since we find it accelerate training. The weights of the teacher model is frozen during imitation learning. By imitating the intermediate node features, we provide high capacity training signal to the student to recover groundtruth acceleration by proper message passing. To successfully imitate the teacher, the student have to conduct occlusion reasoning, and take the effects of occluded particles and erroneous mesh edges into account. In addition to node features, the student model also mimic the global embedding of teacher model to make more accurate reward prediction. We use mean square error for imitation learning.

Auxiliary Reward Prediction

Following [44], we additionally train our dynamics model to predict reward in order to regularize the model. The groundtruth reward, which is the coverage of cloth after one time step, is calculated by approximation as described in A.1.3. The coverage is calculated over all particles, thus it provides information from a global view to the model. The reward model is a three-layer MLP which takes global embedding c^L as input. We use mean square error to train the model.

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

2.4 Experiments

2.4.1 Experimental Setup

Simulation Setup As mentioned, we use the Nvidia Flex simulator wrapped in SoftGym [64] for training. The robot gripper is modeled as a spherical picker that can move freely in 3D space and can be activated so the nearest particle will be attached to it. For training, we generate random pick-and-place trajectories on a square cloth. The side length of the cloth varies from 25 to 28 cm. For evaluation, we consider three different shapes: 1) the same type of square cloth as used in training; 2) Rectangular cloth, with its length and width sampled from $[19, 21] \times [31, 34]$ cm. 3) Two layered T-shirt (the square cloth used for training was single-layered). For each shape, the experiment was run 40 times, each time with a different initial configuration of the fabric. We report the 25%, 50% and 75% (Q_{25}, Q_{50}, Q_{75}) percentiles of the performance. For all our quantitative results, numbers after $\pm$ denotes $\max(|Q_{50} - Q_{25}|, |Q_{75}, -Q_{50}|)$.

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

We evaluates two variants of our method: Visible Connectivity Dynamics (VCD) and VCD with graph imitation learning. We compare with previous state-of-the art methods for cloth smoothing: VisuoSpatial Foresight (VSF) [38], which learns a visual dynamics model using RGBD data;

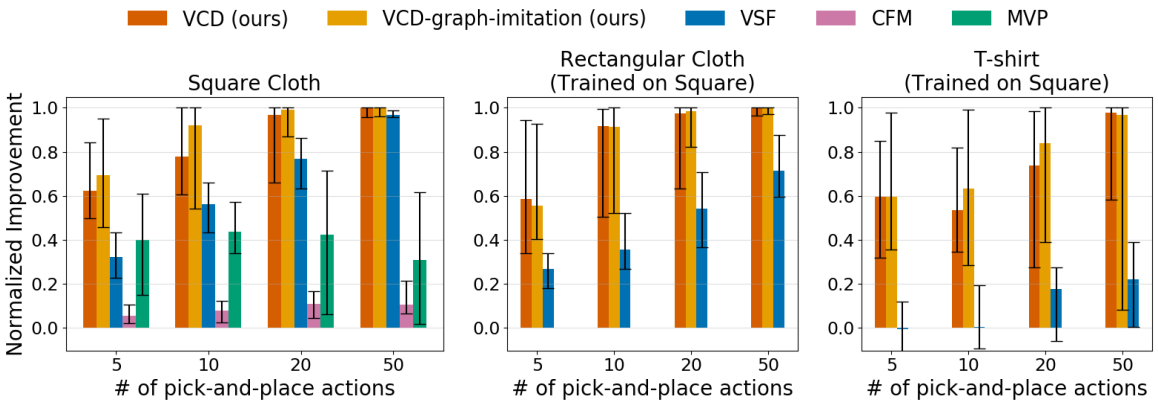


Figure 2.4: Normalized improvement on square cloth (left), rectangular cloth (middle), and t-shirt (right) for varying number of pick-and-place actions. The height of the bars show the median while the error bars show the 25 and 75 percentile. For detailed numbers, see the appendix.

Contrastive forward model (CFM) [127], which learns a latent dynamics model via contrastive learning; Maximal Value under Placing (MVP) [126], which uses model-free reinforcement learning with a specially designed action space. More implementation details can be found in the appendix.

Real World Setup We use our dynamics model trained in simulation to smooth cloth in the real world with a Franka Emika Panda robot arm and a standard panda gripper, with FrankaInterface library [129]. We obtain RGBD images from a side view Azure Kinect camera. We use color thresholding for segmenting the cloth and obtain the cloth point cloud. We evaluate on three pieces of cloth: Two square towels made of cotton and silk respectively, and one t-shirt made of cotton. We use our dynamics model trained in simulation without any fine-tuning. More details are in the appendix.

2.4.2 Simulation Results

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

We first test all methods on the same type of square cloth used in training. The results are shown in Figure 2.4 (left). Under any given number of pick-and-place actions, *VCD* greatly outperforms all of the baselines. The graph imitation learning approach described in Section **??** further improves the performance. To test the generalization of these methods to novel cloth shapes that are not seen during training, we further evaluate on a rectangular cloth and a t-shirt. For this experiment we only compare VCD to VSF, since VSF achieves the best performance on the square cloth among all the baselines. The results are summarized in Figure 2.4 (middle and right). VCD shows a larger improvement over VSF on the rectangular cloth. T-shirt is more different from the training square cloth and VSF completely fails, while VCD still shows good generalization. The graph imitation learning still leads to marginal improvement and better stability on rectangular since it has a similar shape to the square cloth. However, as the t-shirt has very different shape compared to the square

cloth, VCD-graph-imitation does not lead to much improvement and has larger variance on it.

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

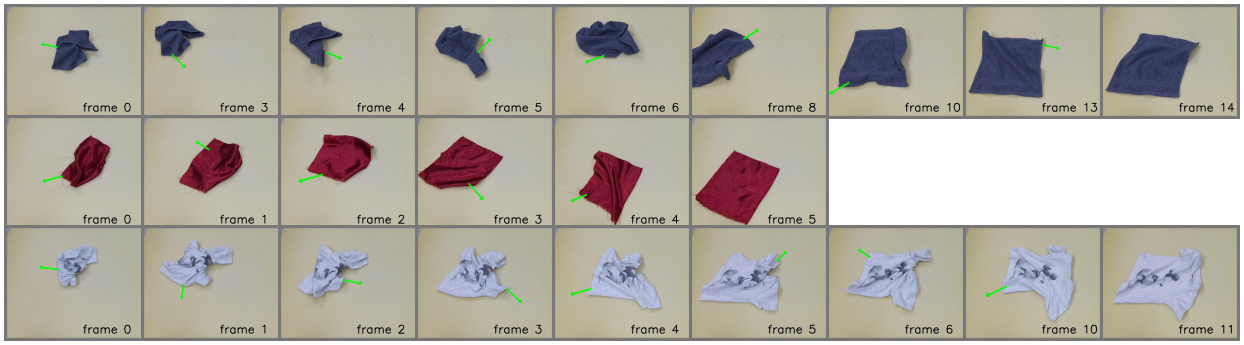


Figure 2.5: Smoothing cloths of different colors, materials and shapes with our method on a Franka robot: square cotton (top), square silk (middle), cotton t-shirt (bottom). Each row shows one trajectory. Frame 0 shows the initial configuration of the cloth, and each frame after shows the observation after some number of pick-and-place actions, with the number labeled on the frame. The green arrow shows the 2D projection of the pick-and-place action executed.

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

Table 2.1: Normalized improvement of VCD in the real world.

2.4.3 Real-world Results

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

We evaluate 12 trajectories for each cloth. The quantitative results are in Table 2.1 and a visualization of smoothing sequences is shown in Figure 2.5. Despite the drastic differences of the cotton and silk cloths in visual appearances, shapes, as well as the different dynamics, our model is able to smooth the cotton and silk cloths and generalize well to t-shirt. We also report the performance if our method is able to terminate optimally in hindsight and choose the frame with the

highest performance in each trajectory; the result is shown in the last column of Table 2.1. Videos of complete trajectories and the model predicted rollouts can be found on our project website.

2.4.4 Ablation Studies

We perform the following ablations to study the contribution of each component of our method. The first ablation replaces the learned GNN dynamics model with the Flex simulator to test whether a learned dynamics model performs better for our task than the physical simulator. In more detail, after we use the edge GNN to infer the mesh edges on the point cloud, we create a cloth using Flex where a particle is created at each location of the voxelized points and a spring connection is added for each inferred mesh edge. The results is shown in Table 2.2, row 2. We see that using the Flex simulator instead of the dynamics GNN produces worse performance. The main reason is that the cloth created from the partial point cloud with the inferred mesh edges deviates from the common cloth mesh structure used in Flex; thus, using the Flex simulator under this condition does not create realistic dynamics. Besides, planning with FleX is much slower than planning with the learned GNN dynamics model ($\sim$330s for 1 pick-and-place with FleX and $\sim$40s with VCD). On the other hand, the dynamics GNN is trained directly on the partial point cloud; therefore it can learn to compensate for the partial observability when predicting the cloth dynamics. This ablation validates the importance of using a dynamics GNN to learn the dynamics of the partially observable point cloud.

Algorithm	Normalized Improvement
VCD (Our method)	$\mathbf{0.778 \pm 0.206}$
Replace dynamics GNN with Flex	0.616 ± 0.143
No edge GNN (dynamic nearby edges)	0.531 ± 0.298
No edge GNN (fixed nearby edges)	0.599 ± 0.327
Remove edge GNN at test time	0.259 ± 0.118

Table 2.2: Normalized improvement of all ablations in simulation after 10 pick-and-place actions.

The next set of ablations aims to test whether using an edge GNN to infer the mesh edges as described in Section 2.3.1 is necessary for learning a good dynamics model. First, we train a dynamics GNN without using the edge GNN, where the edges are constructed solely based on distance by Eqn. (2.1). Since this ablation does not use an edge GNN, it cannot have two different edge types (nearby edges vs mesh edges). Thus at test time, all edges can either be kept fixed throughout the trajectory (similar to the mesh edges in our model), or dynamically reconstructed using Eqn. (2.1) at each time step (similar to the nearby edges in our model). The results of these two ablations are shown in Table 2.2, rows 3 and 4. As can be seen, the performance is worse without the edge GNN.

Additionally, we perform another ablation where we train with both nearby edges and mesh edges, but at test time, we do not use an edge GNN to infer the edge type; instead we consider the edges that satisfy the criteria of Eqn. (2.1) in the first time step as the mesh edges. The result of this ablation is shown in Table 2.2, row 5. The performance is again much worse. All these ablations validate the importance of using an edge GNN to infer the mesh edges.

Chapter 3

Mesh-based Dynamics with Occlusion Reasoning for Cloth Manipulation

3.1 Introduction

Manipulation of clothing has wide applications but remains a challenge in robotics. Cloth has nearly infinite degrees of freedom (DoF), making state estimation difficult and resulting in complex dynamics. Furthermore, cloth is often subject to severe self-occlusions, especially when it is crumpled.

Due to self-occlusions, prior methods typically rely on visible features for manipulation, such as wrinkles [111] or corners [71]. Data-driven methods have been proposed that can learn a dynamics model in the pixel space [38] or in a latent space [127]. Recently, Lin, Wang, *et al.* [65] proposed to learn a mesh dynamics model on the observed partial point cloud of a crumpled cloth (VCD). However, VCD only considers a graph over the visible points and does not reason explicitly about the occluded regions of the cloth. This simplification sometimes prevents the planner from finding optimal cloth unfolding actions.

Some prior work [54, 59, 72, 62, 61, 60, 19] attempt to reconstruct the full cloth structure. To simplify the task, these approaches typically first grasp and lift the cloth in order to limit the diversity of possible cloth configurations. However, this approach prevents continual state estimation throughout a manipulation task, since the pose can only be estimated while being held in the air by the gripper at a single point. In contrast, we aim to estimate the cloth pose from a more diverse set of poses, such as while it is crumpled on a table in arbitrary configurations.

In chapter 3, we propose MEDOR (MEsh-based Dynamics with Occlusion Reasoning) which

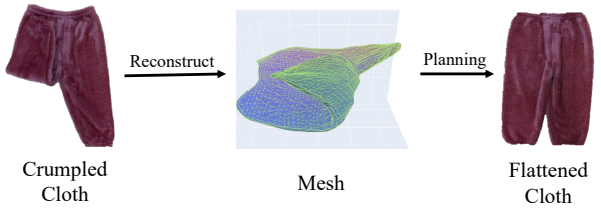


Figure 3.1: Our method, MEDOR (MEsh-based Dynamics with Occlusion Reasoning), explicitly reasons about occlusions by reconstructing the full mesh of a cloth; we then use a learned mesh-based dynamics model to plan the robot actions.

is built on top of previous methods for mesh-based dynamics learning [65, 88, 97]. MEDOR explicitly reasons about occlusions by reconstructing the full cloth mesh from a single depth image. While previous work such as GarmentNets [19] has demonstrated promising results for mesh reconstruction, we show that, by itself, it is not robust enough for a robot cloth manipulation system. Due to the inherent ambiguity induced by self- occlusions from the crumpled cloth, the reconstructed mesh will likely have many errors, which will create difficulties for downstream planning. Our insight is that we can improve the mesh reconstruction using test-time optimization with self-supervised losses, which can be easily computed without access to the ground-truth mesh and can be directly optimized on real data.

We evaluate our method on two tasks: cloth smoothing and cloth canonicalization, which requires aligning the cloth with a canonical flattened pose. We show that our method of mesh reconstruction, with test-time finetuning, enables a robot to smooth or canonicalize the cloth from crumpled configurations more accurately than previous methods. Our contributions include:

1. a novel perception model that can better estimate the full cloth structure from crumpled configurations, including a self-supervised test-time optimization procedure.
2. a cloth manipulation system that plans over the reconstructed cloth mesh and can perform both cloth flattening and cloth canonicalization.

3.2 Related works

3.2.1 Perception for Cloth Manipulation

There has been a long history of work on cloth perception for manipulation. We refer to Jimenez *et al.* [49] for a comprehensive overview.

Earlier works usually estimate specific visual features of the cloth for manipulation. These include detecting edges and corners for re-grasping [71] or grasping and un-folding [84, 124, 92], or detecting wrinkles for smoothing [111]. If the cloth is loosely extended, simplified models such as the parameterized shape model [77] or the polygonal model [105] can also be used for folding. Other works also detect category-specific features such as collars and hemlines [93]. These features have been used with hand-designed controllers and strategies; in this work, we aim to learn mesh reconstruction and a cloth dynamics model that we can use to plan a cloth manipulation action sequence.

There are also prior works on more generic pose estimation for clothes. One line of works tries to estimate the pose of on-body clothing from video by leveraging a human body shape prior [48, 26, 87, 96, 89, 37, 107]. Another line of works assumes the initial configuration of the cloth is known and uses tracking to estimate the full configuration of the cloth under occlusion [18, 121, 114]. In contrast, we assume that the cloth may be initially crumpled on a table in an unknown initial configuration.

Some other works [54, 59, 72, 62, 61, 60, 19] simplify the problem by lifting up the cloth using the robot gripper for the purpose of easier pose estimation; lifting up the cloth significantly reduces the set of possible poses and simplifies the reconstruction task. After lifting the cloth, Kita *et al.* [54] deform a set of predefined representative shapes to fit the observed data. Other works create a dataset of clothes grasped at different locations and retrieve the observed pose at test-time by classification [59, 72, 61, 60] or using nearest neighbor [62]. In contrast, our method

can estimate the cloth directly from a crumpled state on the table, which enables our method to continually re-estimate the cloth state throughout a manipulation sequence.

Recently, GarmentNets [19] performed categorical cloth 3D reconstruction by mapping the cloth point cloud into a normalized canonical space (NOCS) defined for each cloth category [120]. However, like the above methods, GarmentNets also requires grasping and raising the cloth into the air and obtaining four different camera views to reduce the amount of occlusion. In contrast, our perception module only requires a single view of the cloth crumpled on the table. Further, we find that the reconstructions produced by GarmentNets are not sufficient for accurate cloth manipulation. Also, unlike GarmentNets that only considers the perception task, we demonstrate a full cloth manipulation system and show the effectiveness of our perception method for manipulation.

3.2.2 Data-driven Methods for Cloth manipulation

Prior works in data-driven cloth manipulation can be categorized as model-free or model-based. Model-based methods train a policy that outputs actions for cloth manipulation. The policies are trained by either reinforcement learning [73, 126], imitation learning [100], or by learning a value function [35]. Alternatively, the policies can be learned as a one-step inverse dynamics model [123, 81].

The second approach is to learn a dynamics model and then plan over the model to find the robot actions. Hoque *et al.* directly learn a video prediction dynamics model [38] and Wilson *et al.* learn a latent dynamics model with contrastive losses [127]. These models do not explicitly reason about the cloth structure, making generalization difficult. Recent work (VCD) trains a mesh dynamics model [65] over the visible points. However, without reasoning about the occluded regions, the planner will often fail to find the optimal smoothing actions. In contrast, our approach plans over a full reconstructed mesh dynamics model.

3.2.3 Test-time Optimization

Test-time optimization has been widely used for view-synthesis [76, 128], 3D particle reconstruction [16] and 3D scene flow [90]. For example, Chen et al. [16] trained a network to predict an object point cloud that is consistent with a set of object masks in different camera views. However, these approaches have not been applied to cloth reconstruction or robot manipulation tasks.

3.3 Background

3.3.1 Problem Formulation

We consider the task of manipulating clothes in a planar workspace with a single robot arm. A cloth at time t is represented by a mesh $M^t = (V^t, E^t)$ with vertices $V^t = \{v_i\}_{i=1...N}$ and mesh edges E^t. Each vertex consists of a position x_i and velocity $\dot{x}_i$ that will change with the cloth configuration. The ground-truth configuration of the cloth M^t is unknown, and the robot only observes the RGB-D image I^t, which includes severe self-occlusions for crumpled garments (though our method only uses the color to segment the cloth from the background). Given the camera intrinsics and the segmentation mask, we can also back-project I^t to a partial point cloud of the

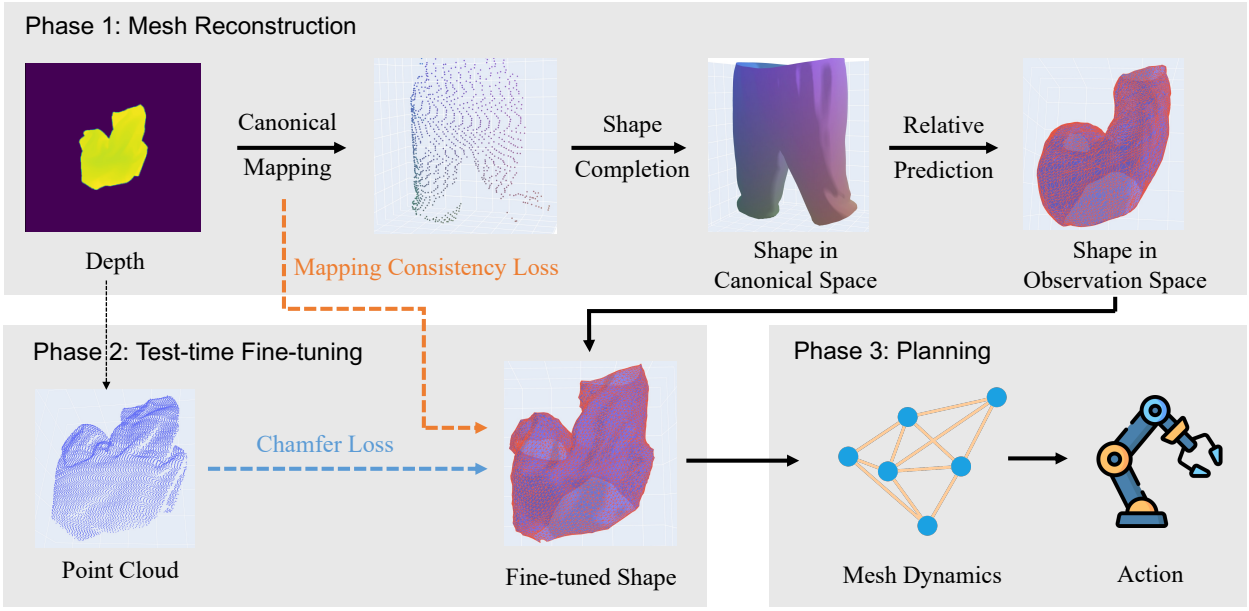


Figure 3.2: System overview: First, we obtain an initial estimate of the full shape of an observed instance of a cloth from a depth image. In this phase, we obtain an estimate of the cloth shape in both canonical space and observation space. Then, we conduct test-time finetuning to improve the prediction to better match the observation. Last, we plan with the predicted mesh using a learned mesh-based dynamics model.

cloth. The observation is captured by a top-down camera mounted on the robot end-effector in our setting. As in prior works [126, 38, 65], we use pick-and-place action primitives for performing cloth manipulation.

3.3.2 GarmentNets

GarmentNets [19] is a previous work that performs categorical cloth reconstruction from a partial point cloud. It contains three steps: first, a normalized canonical space (NOCS [120]) is defined for each cloth instance by simulating the cloth worn by a human in a T-pose. A canonicalization network (using a PointNet++ architecture [91]) is trained to map an observed partial point cloud to the canonical space (see Figure 3.2, top). Second, a 3D CNN performs volumetric feature completion to obtain a dense feature grid. An MLP is then used to predict the winding number [43] for each point, which is used for surface extraction. In the last step, GarmentNets samples points on the extracted surface and trains a warp field network to map each point in the canonical space back to its location in the observation space.

GarmentNets requires a multi-view depth image to reduce occlusions; it also requires the cloth to be grasped in the air to limit the set of possible poses of the cloth. In contrast, we consider the more general setup where the cloth is on the table in an arbitrary configuration.

3.4 Approach

Our goal is to build a general cloth manipulation system that can reason about the occluded cloth regions of a cloth from a partial observation and plan robot actions based on these estimates. To do

so, we propose a three-part approach: (1) First, given only a partial observation of a crumpled cloth, we train a model to generate a complete mesh of the cloth; (2) Due to the difficulty of occlusion reasoning, a neural network alone often fails to accurately reconstruct the cloth mesh. Therefore, we design a test-time finetuning scheme to adapt the predicted mesh to match the observation. (3) Last, we plan with the predicted mesh using a learned dynamics model to find optimal actions for the manipulation task.

3.4.1 Estimating the pose of a cloth

Reconstruction of the full cloth structure is fundamentally challenging due to the high-dimensional state space and the ambiguity induced by self-occlusion. As discussed in Section 3.3.2, GarmentNets [19] simplifies the problem by using a robot to grasp and lift the cloth and capture 4 observations from different views. In contrast, we tackle the harder problem of estimating the cloth state from a crumpled configuration on the table, so that the cloth state can be re-estimated throughout a manipulation sequence.

To tackle this challenge of cloth pose estimation from crumpled configurations, we make several modifications to GarmentNets that greatly improve its performance:

HRNet: First, since the cloth is not lifted up, we represent the cloth as a depth image captured by a single top-down camera instead of as a point cloud merged from 4 observations. In this case, the Pointnet++ [91] architecture is no longer able to provide reasonable estimates of the cloth configuration, as we show in the ablation 2.2. Instead, we use an architecture designed for depth images. In particular, we find that PointNet++ is not able to distinguish whether a piece of cloth is folded above or folded below the rest of the cloth. Differentiating these two cases requires capturing the subtle depth changes at the boundary where two layers meet. Therefore, we replace Pointnet++ with a High Resolution Network (HRNet) [110] which is a convolutional architecture that specializes in producing a high-resolution and spatially precise representation.

Relative predictions: We found that the reconstruction model sometimes inaccurately estimates the configuration of the cloth in observation space, but it often predicts the canonical shape reasonably well (see Figure 3.2 for a visualization of these two spaces). Thus, we learn to predict the delta between the position of a point in canonical space and its corresponding location in observation space. Specifically, given the predicted mesh in canonical space, $\tilde{M}^c = (\tilde{V}^c, \tilde{E}^c), \tilde{V}^c = \{\tilde{v}_i\}_{i=1...n}$, where the coordinate of each vertex in canonical space is $\tilde{x}_i^c$, we predict a 3-dimensional residual vector $\tilde{f}_i$ for each point i. The predicted coordinates in the observation space $\tilde{x}_i^o$ are obtained by

$$\tilde{x}_i^o = \tilde{x}_i^c + \tilde{f}_i \tag{3.1}$$

As shown in the ablation experiments in Table 3.1, both modifications described above are important for the performance of the method.

3.4.2 Test-time finetuning

Estimating the complete structure of cloth is inherently challenging due to the high degrees of freedom and the ambiguity induced by occlusion. As a result, the network described above still has significant prediction errors, as shown in Fig. 3.3.

To tackle this issue, we design a test-time finetuning scheme that further optimizes the predicted mesh using self-supervised losses that can be computed without knowledge of the ground-truth

cloth state. We deform the predicted mesh by optimizing the location of each vertex to optimize an objective consisting of the sum of two loss terms: unidirectional Chamfer loss and mapping consistency loss.

Unidirectional Chamfer loss. The first self-supervised loss term penalizes any deviations between the predicted mesh and the observed cloth surface (depth image) so that geometric details are preserved. Since the observation only contains information about the visible surface, optimizing the mesh with a standard bi-directional Chamfer loss will result in undesirable results, i.e., the predicted mesh will move entirely to the visible surface and no part of the mesh will remain in the occluded region. Therefore, we use a unidirectional Chamfer loss as described below.

Suppose at timestep t, the point cloud observation of the cloth is $P^t = \{p_i\}_{i=1..L}$ and the predicted mesh in observation space is $\tilde{M}^t = (\tilde{V}^t, \tilde{E}^t)$. The coordinate of each vertex is specified by a 3-dimensional vector. Then the loss term is formulated as:

$$\mathcal{L}_{\mathcal{C}}(\tilde{V}^t; P^t) = \frac{1}{|P^t|} \sum_{p_i \in P^t} \min_{\tilde{v}_j \in \tilde{V}^t} d(p_i, \tilde{v}_j) \tag{3.2}$$

where $d(\cdot, \cdot)$ can be any distance metric, and we use Euclidean distance. In other words, for each point in the observed point cloud, we find the distance to the nearest point in the predicted mesh and minimize the sum of such distances.

Mapping consistency loss. Since the exact correspondence between the predicted mesh and the partial point cloud is not available, directly optimizing the uni-directional Chamfer loss above may lead to a local minimum. To alleviate this issue, we observe that the mapping from the observation space to canonical space and then back to observation space (see Figure 3.2, top) creates a cycle; thus we add a loss that, for each visible point, this cycle should end at the location where it started.

Let P^t be the point cloud observation of the cloth; let f be the learned mapping from each observation point to a location in the canonical space; and let g be a learned mapping from each location in the canonical space to a position back in the observation space (shown in Figure 3.2, top). Note that g operates on the predicted completed cloth surface which includes both observed and occluded points. The mapping consistency loss term can be expressed as:

$$\mathcal{L}_{\mathcal{M}}(P^t) = \frac{1}{|P^t|} \sum_{p_i \in P^t} d(g(f(p_i)), p_i) \tag{3.3}$$

In other words, this loss penalizes the distance between the original location of each observed point p_i and its predicted location $g(f(p_i))$.

Optimization. We use gradient descent with the Adam optimizer [51] to optimize the losses above. The optimization is divided into two phases. In the first 50 steps, we optimize the mesh using the Chamfer loss together with mapping consistency loss; then we optimize the mesh using the Chamfer loss alone for another 50 iterations.

We do not perform a joint optimization throughout the optimization process because multiple pixels in the depth image might be mapped to the same voxel in the canonical space. Enforcing mapping consistency throughout the optimization process will create implausible meshes whose vertices converge to a set of clusters. Instead, we use the mapping consistency loss to provide a good initialization (beyond the initial network prediction), and then we use the uni-directional Chamfer loss to refine the geometric details. As shown in the ablation in Table 3.1, the two-stage optimization is critical for the performance.

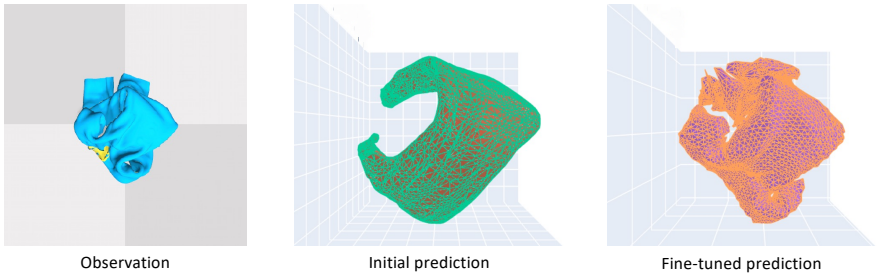


Figure 3.3: An example showing the benefits of test-time finetuning: The network is able to correctly estimate the main cloth structure, but the initial predicted mesh may be overly smooth (center). Test-time finetuning significantly improves the quality of predicted mesh (right). For more examples, see our project website.

3.4.3 Planning with GNN-based dynamics model

Once we have reconstructed the cloth mesh, we use the reconstructed mesh to plan the robot actions using a dynamics model. One option would be to use a physical simulator as a dynamics model. However, our experiments show that using a learned dynamics model can be more accurate and faster than a simulator when planning with a reconstructed mesh; these results (shown in Sec. 3.5.3) are consistent with previous papers [65]. We suspect that a learned dynamics model can be more robust to errors in the mesh reconstruction compared to a physics-based simulator.

As a first step towards learning a dynamics model, we downsample the mesh by vertex clustering [68] for faster computation. We apply vertex clustering in the canonical space (defined as the pose of the cloth when worn by a human in a T-pose) because the cloth surfaces are well separated and thus will avoid undesirable artifacts such as merging different layers.

Given a down-sampled mesh from the simulator $\bar{M}^t = (\bar{V}^t, \bar{E}^t)$, we train a dynamics Graph Neural Network (GNN) [97, 88]. A GNN encodes the input feature on the nodes and on the edges and then conducts multiple message passing steps between the nodes and edges [98]. The decoder will decode the latent features of each node into the predicted acceleration for that node. The GNN dynamics model that we use is the same as the dynamics model in VCD [65]: the input feature on each node is the historic particle velocities and an indicator of whether the node is picked by the gripper or not. The edge features includes the distance vector of connected vertices $(x_j - x_k)$, its norm $||x_j - x_k||$, and the current displacement from the rest position $||x_j - x_k|| - r_{jk}$. We use Euler integration to obtain the states of the cloth in the next time step. The action is encoded to the dynamics model by directly modifying the position and velocity of the grasped point on the cloth. We refer the reader to previous work [65] for details on the graph dynamics model. Once we have the dynamics model, we use random shooting to plan over different pick-and-place actions and pick the action with the highest predicted reward. At each time step, we plan over a horizon of one, i.e., one pick-and-place action.

3.4.4 Implementation details

All models are trained in simulation and data are generated by Nvidia Flex wrapped in Softgym [64]. To obtain a diverse dataset with garments of different sizes and shapes, we port the CLOTH3D dataset [6] into Softgym. We choose five categories of garments from CLOTH3D: Trousers, T-shirts, Dress, Skirt and Jumpsuit. Each category contains 400-2000 different meshes and covers a wide range of variations, such as shirts with short and long sleeves, with and without

an opening in front. We divide the CLOTH3D dataset into a train set and test set in a 9:1 ratio.

Mesh reconstruction model Initial crumpled cloth configurations are generated by a random drop or random pick-and-place actions from flattened states, with a ratio of 1:1. The mesh reconstruction model is trained in a category-dependent manner, i.e. one mesh reconstruction model per category, similar to GarmentNets [19]. For each category, the training set contains 20,000 observations.

Dynamics model The GNN dynamics model is trained only on Trousers and is evaluated across all object categories. The dynamics model is trained on a dataset with 5,000 pick-and-place actions, which equals 500,000 intermediate timesteps. The training of the mesh reconstruction model and the dynamics model each take around 3 days.

Runtime At test-time, the mesh reconstruction model and test-time finetuning take around 2.5 and 3 seconds per mesh respectively. For planning, we randomly sample 500 pick-and-place actions and rollout each action with the GNN dynamics model, which takes around 100 seconds. For additional implementation details, please refer to the supplementary materials.

3.5 Experiments

To evaluate the effectiveness of our method, we conduct comprehensive experiments on the tasks of cloth flattening and canonicalization (described below). To demonstrate the generalizability and robustness of our method, we evaluate in both the real world and in simulation on 5 different categories of garments. Through the experiments, we would like to answer the following questions:

1. Does explicit occlusion reasoning improve the performance of cloth manipulation? How does it compare to methods that operate only on the visible points?
2. Does test-time finetuning improve the quality of predicted mesh as well as the performance in cloth manipulation tasks?
3. Can our method work on a physical robot?

3.5.1 Tasks

Flattening. Our goal is to flatten a crumpled cloth, that is, spreading it on the table. Following prior works [65], we compute the coverage of the cloth as the objective for planning and evaluation.
Canonicalization. Usually, flattening is the first step of a cloth manipulation pipeline [62, 61, 60, 72, 30], which makes the subsequent tasks such as folding easier. However, for certain types of clothing, such as skirts or unbuttoned shirts, the flattening objective can produce undesirable results, as shown in Figure 3.4a, in which the robot maximizes the area covered by the cloth in a manner that is not conducive for downstream folding.

Therefore, we also evaluate our method on a task that we call "cloth canonicalization," where the goal is to manipulate the cloth and align it with the flattened canonical pose, as shown in Figure 3.4b. To account for the ambiguity due to rotation and reflection symmetries, we define a set of symmetries for each type of cloth. Using these symmetries, we define a canonical goal

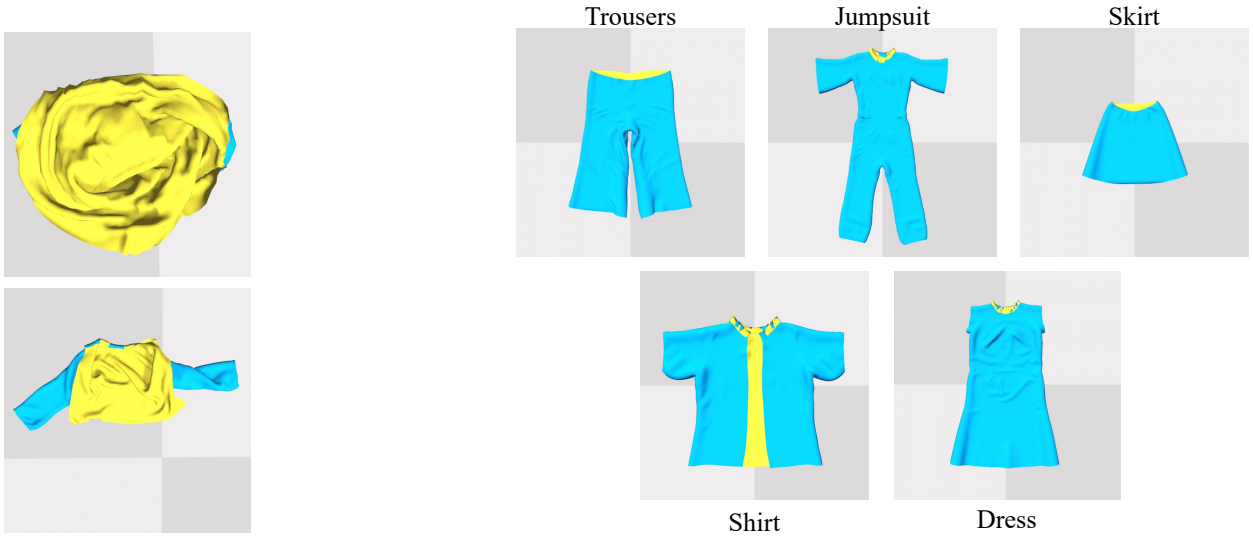


Figure 3.4: Flattening (i.e. maximizing the covered area of the cloth) may not always give us a good starting point for folding. Left: Undesirable results of actions that optimize for the flattening task (maximizing the coverage). Right: Examplar goal poses for canonicalization of each category.

set of flattened poses $\mathcal{G} = \{G_i^{N\times 3}\}_{i=1...A}$ for each cloth instance, where A is the number of valid canonical poses. For example, for Trousers, we can rotate the canonical pose by 180 degrees to obtain another valid goal. The cost is computed as the minimum of the average pairwise distance to each of the possible canonical poses. Suppose that the current configuration of the cloth in the simulator is $V \in \mathbb{R}^{N\times 3}$, where N is the number of vertices. Then the cost is computed as

$$Cost_{canon} = \min_{G_i \in \mathcal{G}} \frac{1}{N} \sum_{j=0}^{N-1} (g_j - v_j)^2 \tag{3.4}$$

Note that G_i and V have the same number of vertices because both refer to the simulated cloth in different configurations. In this work, we allow for our method to canonicalize the cloth without penalizing for errors with respect to a rigid transformations; this is because, for the task of cloth folding, the rotation and translation of the cloth is of lesser importance. To evaluate this, we first align the goal with the current state by computing an optimal rigid transformation using the Kabsch algorithm [3], and then we compute the cost by Equation B.1.

For each task, we show the normalized improvement (NI) of each method, where 0 indicates no change from the initial state and 1 is the best possible performance. For flattening, we use the normalized improvement metric defined in previous work [65]; for canonicalization, we compute the normalized improvement as $\text{NI}_{canon} = \frac{cost_{init} - cost_{cur}}{cost_{init}}$ where $cost_{init}$, $cost_{cur}$ are the costs of initial and current cloth configurations computed by Eq. B.1.

3.5.2 Simulation Experiments

Baselines

We compare our method to 4 baselines, including two state-of-the-art cloth manipulation methods:

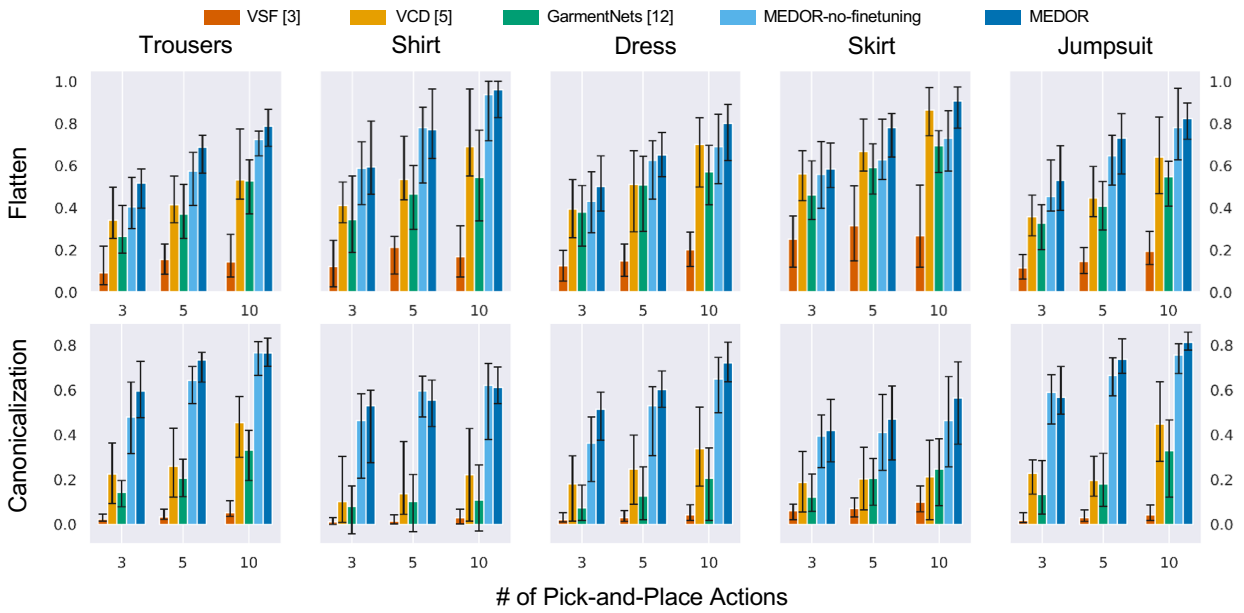


Figure 3.5: Normalized improvements on 2 tasks and 5 different categories of cloths. The height of the bar represents the median and the error bars show the 25 and 75 percentile of the performance.

- **VisuoSpatial Foresight (VSF)** [38]. This baseline learns a visual dynamics model in the RGB-D observation space. It is trained on each category separately.
- **Visible Connectivity Graph (VCD) [65]**. Similar to our method, VCD learns a particle-based dynamics model. However, unlike our method, VCD only operates on the visible points on the cloth, without explicitly reasoning about occlusions. An edge GNN is used to infer the mesh structure on the partial point cloud. To make a fair comparison with our method, the edge GNN is trained in a category-specific manner, while the dynamics model is only trained on Trousers (similar to our approach).
- **GarmentNets [19]**. In this baseline, we use the original implementation of GarmentNets for mesh reconstruction, which processes a partial point cloud with PointNet++ [91] and doesn't use relative prediction. For planning, we evaluate this baseline with the same mesh-based dynamics model as our method.
- **MEDOR-no-finetuning**. This is a variant of our method that removes the test-time finetuning step (Section 3.4.2).
- **MEDOR**. This is our full method, which is essentially a modified version of GarmentNets with test-time finetuning.

For VSF and VCD, we provide the ground-truth RGB-D image and the ground-truth mesh (respectively) in the canonical pose to use for the reward computation. Note that our method does not have access to this information. More details on the baselines can be found in the Supplement.

Results

A trajectory is terminated after ten picks or when NI>0.95.

For each category, we generate 40 initial configurations by random drops. As shown in Fig-

ure 3.5, compared to methods without explicit occlusion reasoning (*VCD* [65] or *VSF* [38]), our methods MEDOR and MEDOR-no-finetuning both achieve competitive performance in all categories and both tasks. This shows the benefits of occlusion reasoning in cloth manipulation and the benefits of recovering the full configurations explicitly. Comparing *MEDOR* against *MEDOR-no-finetuning*, we can see the importance of test-time finetuning to adapt the mesh to better fit the observation. Qualititative examples are shown in Figure 3.3 as well as on the website, which show the differences in the mesh prediction before and after test-time finetuning.

For *VSF* (orange) and *VCD* (yellow), we see that the canonicalization task remains challenging even after being provided with the ground-truth RGB-D image or mesh for reward computation. This demonstrates the importance of planning with the completed cloth shape instead of planning only over the visible parts of the cloth.

We can also see that without our modifications, the original GarmentsNets [19] (green) has poor performance. The variant of our method *MEDOR-no-finetuning* includes the modifications to GarmentNets described in Section 3.4.1: HRNet and Relative Prediction. The huge performance gap between *GarmentNets* and *MEDOR-no-finetuning* demonstrates that these modifications lead to large performance benefits.

3.5.3 Ablations

To further examine each component and design choice in the paper, we conduct the following ablations on both the flattening and canonicalization tasks. Table 3.1 shows the normalized improvements averaged over 3, 5, and 10 picks, averaged over the flattening and canonlicalization tasks, and averaged over all 5 garment categories.

Why is occlusion reasoning beneficial to cloth manipulation? Explicit occlusion reasoning improves the performance of our framework on cloth manipulation in two aspects: (1) Using the reconstructed mesh helps with the reward computation, and (2) Using the reconstructed mesh helps with the dynamics model. We differentiate these benefits in the following experiments:

(1) In the 8th row *Ours w/ Partial Reward*, we use only the visible portion of the cloth for reward computation, while the dynamics uses the full reconstruction (visible + occluded regions). Compared to our full method, the performance drops by 29%, showing the benefits of occlusion reasoning for the reward computation.

(2) Using the reconstructed mesh improves the accuracy of the dynamics model: In VCD, the GNN dynamics model is trained and tested on the visible portion of the mesh. We compute the open loop rollout error of the VCD dynamics model on the visible mesh, and we likewise compute the rollout error of our method using the reconstructed mesh; we find that the rollout error of the VCD dynamics model (on the partial mesh) is 64.2% higher than the rollout error on the reconstructed mesh. This demonstrates the importance of using the reconstructed mesh for accurate cloth dynamics.

To further this analysis, we also perform an experiment in which we use the full reconstructed mesh for the dynamics model but use only the partial mesh for the reward computation (*Ours w/ Partial Reward*). If we compare the performance of this version to the 2nd row *(No Mesh Reconstruction (VCD [65]))* we see the benefits of using the full reconstructed mesh for the dynamics model instead of the partial mesh (0.462 vs 0.391).

How much does test-time finetuning help, and are both losses necessary? When other components remain unchanged, we see that finetuning with only the Chamfer loss (*No Consistency*

Method	Normalized Improvement
GarmentNets [15]	0.320 ± 0.146
No Mesh Reconstruction (VCD [65])	0.391 ± 0.174
MEDOR-no-finetuning-and-no-relative-prediction	0.560 ± 0.163
MEDOR-no-finetuning	0.585 ± 0.171
Joint Optimization	0.614 ± 0.157
No Consistency Loss	0.623 ± 0.148
Replace GNN by GT Dynamics	0.631 ± 0.161
Ours w/ Partial Reward	0.462 ± 0.210
Ours (full method)	$\mathbf{0.651 \pm 0.138}$
GT Mesh + Learned Dynamics	0.800 ± 0.096
GT Mesh + GT Dynamics	0.870 ± 0.076

Table 3.1: Ablation experiments.

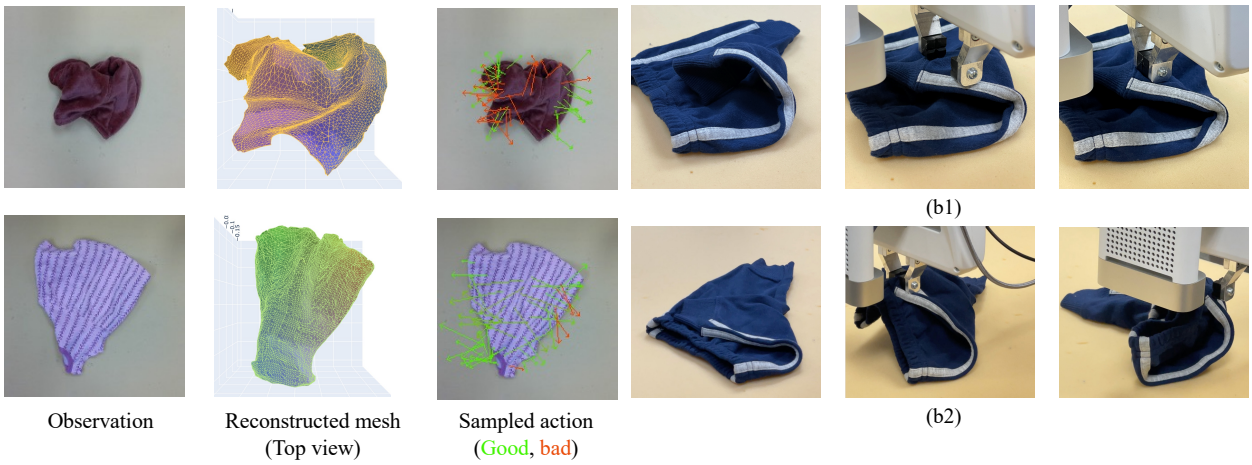
Loss) already improves the performance over no finetuning (*No Finetuning*) by 6.5%. Adding the Mapping Consistency Loss further boosts the performance from 0.623 (*No Consistency Loss*) to 0.651 (*Ours (full method)*); the combined improvement is 11.3%. Also, we find that without the 2-stage optimization scheme (*Joint Optimization*), the mapping consistency loss hurts the performance (comparing *No Consistency Loss* vs *Joint Optimization*).

HRNet [110] vs PointNet++ [91]: Looking at Table 3.1, the only difference between the methods in the first and third row is the use of HRNet instead of PointNet++. The huge difference in performance shows the benefits of the HRNet architecure for this task.

Does Relative Prediction help? Comparing the performance of *MEDOR-no-finetuning-and-no-relative-prediction* with *MEDOR-no-finetuning*, we see that this simple modification (described in Section 3.4.1) improves performance.

Do we need to learn the dynamics model instead of using the ground-truth dynamics model from the simulator? Once we reconstruct the full cloth mesh, one option is to plan using the physics-based dynamics of Nvidia Flex simulator, as similarly done in [16]. This ablation is shown as *Replace GNN by GT Dynamics* in Table 3.1. We can see that this ablation yields a slight performance drop compared to our full method. We speculate that this is because the analytical dynamics model is more sensitive to mesh prediction errors than the learned dynamics model. As an additional point, planning with a simulator is 1.4 times slower than using a GNN dynamics model even after heavy parallelization (247 seconds vs. 103 seconds for 500 rollouts).

Where are the remaining gaps in performance? As we can see in the last two rows, using the ground-truth mesh (instead of a learned mesh reconstruction) improves the performance by 23.1%. We can improve performance another 7% by also using the ground-truth dynamics model. The remaining gap come from the sampling-based planner itself, which might fail to sample good actions.



(a) Example reconstruction results in the real-world, after test-time fine-tuning. In the 1st row, the trousers are successfully reconstructed, including the occluded legs and wrinkles on the surface. The planner (right column) is able to distinguish good actions from bad ones given the reconstructed mesh. In the 2nd row, our model failed to capture the left-bottom corner which is folded under the visible layer. As such, the planner failed to choose actions to reveal the occluded part.

(b) Grasping failures: In (a), the cloth is deformed when the gripper moves down, resulting in missed grasping. In (b), the robot is supposed to grasp the upper layer and unfold it, but it mistakenly grasps the bottom layer as well.

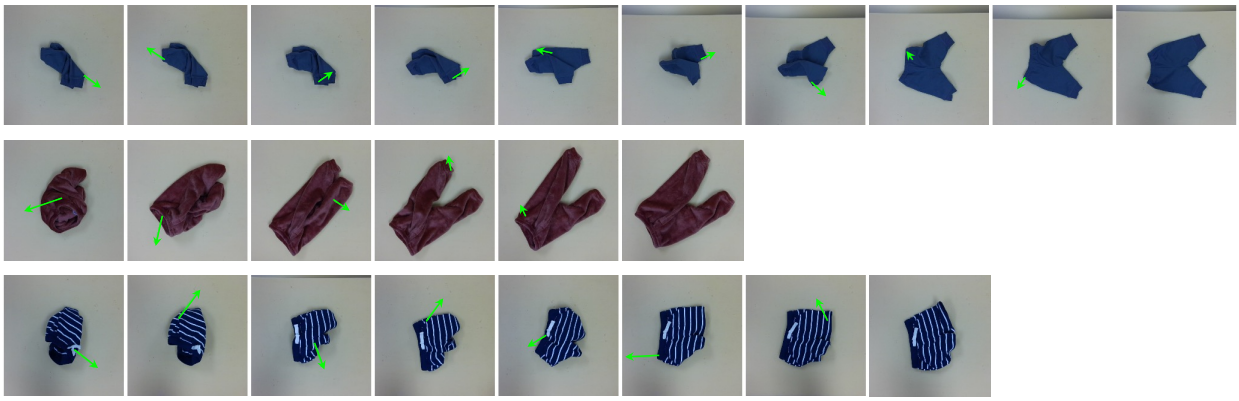


Figure 3.7: We evaluate our method of 5 pieces of clothing. Please refer to our website for videos.

3.5.4 Physical Experiments

We also evaluate our method in the real world by deploying it on a 7-DOF Franka Emika Panda robot. We mount an Azure Kinect depth sensor on the end-effector of the robot. When taking the depth image, the end-effector will move to be centered above the cloth. To obtain a valid plan for pick and place actions, we use MoveIt! [20].

We evaluate our method on Trousers (3 instances) and Dress (2 instances) on the cloth flattening task. For each article of clothing, we run 5 trajectories with at most 10 pick-and-place actions each. The trajectories will be terminated if the normalized improvements exceed 95%. We obtain random configurations by performing a random drop three times. We compare our method with two baselines. *Random* is a heuristic policy that performs random pick-and-place actions. The picked points are biased towards contour of the cloth and the place points are always outside the cloth region. VCD [65] is a prior method that plans with a partial point cloud instead of reconstructing the full mesh. As shown in Fig. 3.7, our model can efficiently smooth the clothes by only a few pick-

and-place actions. The performance of our method compared to the baselines is shown in Table 3.2.

	Random	VCD [65]	MEDOR (Ours)
Trousers	0.044	0.562	**0.647**
Dress	0.036	0.361	**0.468**

Table 3.2: Results of physical robot experiments.

We do observe a gap between the performance in simulation and in the real-world. The first source of error is the reconstruction error. Since the model is trained in simulation, it suffers from a distribution shift resulting from the difference in modeling the cloth physics and material properties, e.g., stiffness, thickness. In Figure 3.6a, we show a successful and a failed reconstruction result. Another source of error comes from the grasp execution, which mostly occur when multiple layers of cloth are stacked together. There are two main failure modes: (1) The robot gripper deforms the cloth, causing a failed grasp; (2) The robot grasps the wrong number of cloth layers. Figure 3.6b illustrates the two cases.

Chapter 4

Self-supervised Cloth Reconstruction via Action-conditioned Cloth Tracking

4.1 Introduction

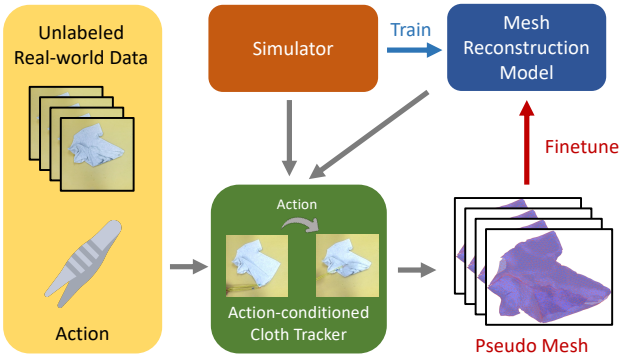


Figure 4.1: We propose a self-supervised cloth reconstruction method that uses action-conditioned cloth tracking to generate pseudo-labels of the full mesh on real world data.

Despite the ubiquitous presence of cloth in real-world, manipulating it with a robot remains a difficult task. Specifically, the high dimensionality and self-occlusion of cloth pose significant challenges for precise state estimation. Prior works [48, 26, 96, 107, 19] try to reconstruct the full mesh of cloth from an RGB or depth observations; the mesh reconstruction model can be used for robot cloth manipulation [60, 59, 72, 61, 72, 39]. However, the mesh reconstruction model is typically trained in simulation and suffers from a sim2real gap between simulated cloth and real cloth. One approach to mitigate the distribution shift from sim2real is to finetune the model with real world data. On the other hand, obtaining the ground-truth full mesh of crumpled clothes is extremely challenging, because the occluded regions are not observable; this presents a challenge for real-world finetuning.

In this work, we present a self-supervised method that leverages a dynamics model and test-time optimization to generate pseudo-ground-truth for mesh reconstruction from a depth image. The high-level idea is to use a human to collect real-world trajectories via a sequence of pick-and-place actions. Then we reconstructed the initial mesh and track the motion of clothes during action execution. By doing so, we are able to obtain full mesh annotations that are previously only available in simulation.

However, tracking the full cloth reliably is a challenging problem. Motivated by the theory of Bayes Filtering, we propose an action-conditioned model-based tracking method. First, we roll out a dynamics (motion) model conditioned on the action to obtain an initial estimate of the motion. This estimate of motion is grounded in physics and accounts for all particles, including

the occluded ones. However, there will inevitably be gaps between the dynamics model and the real world [29, 69, 55], due to incorrect physical parameters and simplified dynamics. To account for dynamics model errors, we further design a test-time optimization method to minimize the discrepancy with the observations at each simulation step, similar to a measurement model of Bayes Filtering.

Our primary contributions are as follows:

1. We propose a new method for self-supervised fine-tuning in the real world of a mesh reconstruction model.
2. To do so, we introduce an action-conditioned model-based cloth tracking method that is robust to occlusions and errors in the dynamics model.

We use our tracking method to fine-tune a mesh reconstruction model on unlabeled real world data. Our experiments 4.4 demonstrate that our method is able to generate plausible pseudo-labels for cloth with complex configurations. We also examine the importance of each component of our method using an ablation study.

4.2 Related works

Cloth Perception and Manipulation. Perception and manipulation of clothes has a long history [49]. Earlier works design heuristic features for specific tasks [71, 84, 124, 92, 111, 77, 105]. More recently, data-driven methods have shown promising results in learning policies [73, 126, 35] or dynamics model [100, 65, 39] for cloth smoothing and folding. Particularly for model-based approaches, prior works have shown that learning a dynamics model over the full mesh with occlusion reasoning can significantly improve the planning performance [39]. While there has been a line of research dedicated to estimating the full mesh of the cloth [19, 39, 48, 26, 87, 96, 107, 60], most of these methods are trained on synthetic data and then transfer to the real world, since obtaining mesh data in the real world can be difficult [6]. As such, this work aims to narrow the sim2real gap by training on real world data collected from cloth tracking.

Deformable Object Tracking. Numerous deformable object tracking algorithms have been developed, such as template-based tracking [58, 132], or simultaneous tracking and reconstruction [83, 42, 34, 102, 11, 10, 12]. Another line of works frame point cloud-based tracking as point set registration [22, 21, 80, 18, 121]. However, these methods are not guaranteed to satisfy physical constraints; further, they do not explicitly model occluded regions. To circumvent the drawbacks of model-free methods, Tang *et al.* [115] propose to refine the result of CPD by inputting it to a physical simulator. Schulman *et al.* [99] designed a modified expectation-maximization (EM) algorithm and perform inference through calls to a physics simulator. These approaches are applied to tracking rope, sponge, and folded cloth. In contrast, we are able to track the configuration of cloth in highly crumpled configurations, which has not been achieved in prior work. We also demonstrate how model-based tracking can be used for self-supervised fine-tuning of a mesh reconstruction model.

Closing the Gap Between Sim and Real. Simulation has shown considerable promise for generating large amounts of labelled data at low cost, especially for domains where groundtruth supervision is difficult to obtain, such as optical flow and scene flow estimation [27, 41, 109, 125, 116] or 3D reconstruction [86, 17, 75, 19]. However, models trained in simulation do not always readily transfer to the real-world due to the sim2real distribution shift. One strategy to bridge this

gap ("sim2real") is to randomize the simulation parameters to create a diverse set of training data [95, 117, 45, 46, 73, 47, 1], and adapting the model by making it domain-invariant [31, 67, 8] or domain-transferable [113, 9, 101, 108, 32, 13, 33]. However, it has not been demonstrated how to effectively apply sim2real methods to the task of mesh reconstruction from depth images. An alternate approach is to use real world data to calibrate the simulator to the real world ("real2sim") [94, 14, 74, 4, 40, 66, 79, 112]. We show that our method is robust to simulator errors.

The main challenges for finetuning a mesh reconstruction model in the real-world is the absence of ground-truth data (i.e., the full mesh). While there exists several real world datasets [7, 131] for on-body cloth reconstruction, directly obtaining the ground-truth full mesh of crumpled clothes in the real-world is very challenging due to self-occlusion, i.e., the occluded portion of the clothes is not observable. Therefore, we propose to generate pseudo labels using model-based tracking.

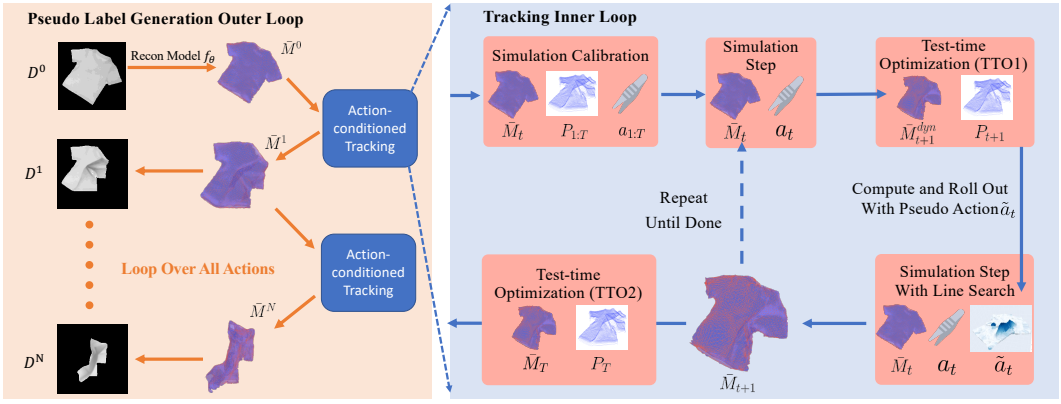


Figure 4.2: Left: The figure on the left demonstrates the workflow for generating pseudo labels for one trajectory. We first reconstruct the initial mesh by using the pretrained mesh reconstruction model f_θ. Then, we estimate the deformation caused by each pick-and-place action through action-conditioned tracking. By tracking all transitions sequentially, we obtain the pseudo mesh for crumpled clothes. **Right**: Before the tracking starts, we first run a parameter search to calibrate the simulation. Then we iterate over all low-level actions by: 1) rolling out the simulation with the picker action for one step; 2) running test-time optimization (TTO1) to align the simulation result with observation, which produces a per-vertex "pseudo action"; 3) running the simulation again with a line search. After all the low-level actions within a pick-and-place action is executed, we run another optimization step (TTO2) to account for the tracking errors.

4.3 Method

The goal of this project is to reconstruct the mesh of a cloth and estimate its (possibly crumpled) configuration, from a depth image observation. Past work in this area has trained a mesh reconstruction model in simulation and transferred the trained model to the real world [48, 26, 96, 107, 19, 60, 39, 87]. However, such methods can suffer from a performance drop in the real-world due to the sim2real gap between simulated clothes and real clothes. To circumvent the issue, we design a self-supervised learning method for finetuning a mesh reconstruction model with unlabeled real data.

The high-level idea of our method is as follows: suppose that we know the full configuration of the current mesh and a dynamics model of the cloth. If we take an action on the cloth, then we can use the dynamics model to estimate the configuration of the cloth at the next timestep.

However, since the dynamics model might not be perfect, we augment the rollout by aligning the predicted mesh with the observation through an optimization procedure. We view this procedure as the motion update and measurement update of Bayesian filtering.

Given an initial mesh reconstruction model trained in sim, the whole system can be divided into 3 stages (Fig. 4.1):

1. Collect real-world trajectories
2. Track the motion of the clothes with our action-conditioned model-based tracking method.
3. Use the tracking output to generate pseudo-ground truth labels and finetune the mesh reconstruction model.

We describe our method in more detail below.

4.3.1 Data Collection in Real-world

In this section, we explain how we instrument and collect real-world trajectories. In order to finetune the mesh reconstruction model, we need to collect real-world data of clothes in random configurations. We use a top-down camera placed above the workspace to capture all the observations.We initialize the state of the cloth into some configuration in which our mesh reconstruction model works reasonably accurately; we then perform a sequence of pick-and-place actions. Then we reset the cloth into a new configuration in which our mesh reconstruction model works reasonably accurately and repeat. When executing each action, we record a full RGB-D video including the intermediate states. The actions are conducted with a tweezer, which helps reduce the amount of occlusion compared to a robot gripper or human hand (Fig.4.3).

Considering each pick and place action, since we record the intermediate states, we obtain a sequence of point clouds $P_{1:T}$ and low-level picker actions $a_{1:T}$ where T is the length of the trajectory. Note that the entire sequence $a_{1:T}$ corresponds to a single pick and place action, and we have a separate action and point cloud sequence for each pick and place action. In our experiments, we set initial configuration to be flattened clothes, and apply 3 pick-and-place actions ($N = 3$) in each trajectory, which takes around one minute (per trajectory) for an experienced human collector.

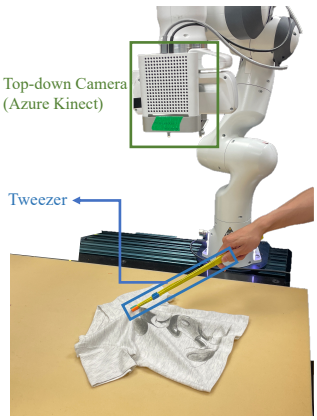


Figure 4.3: A human collector uses a tweezer to conduct pick-and-place actions. RGB-D videos are captured by a top-down camera.

4.3.2 Pseudo Label Generation by Model-based Cloth Tracking

Given an initial depth image D^0, a pretrained mesh reconstruction model, an (imperfect) dynamics model, as well as the action and point cloud sequences recorded in the previous section, our next goal is to estimate the full meshes for every state recorded in the dataset. We assume that the initial depth image D_0 is recorded from the cloth in a flattened state in which the pre-trained mesh reconstruction model is reasonably accurate. Let us denote the estimated reconstruction of this initial mesh as M^0, where a mesh is defined as $M = (V, E)$ with vertices V and edges $E \subseteq V \times V$.

Given the initial estimated mesh $\bar{M}_0$, a sequence of point clouds $P_{1:T}$, and a sequence of actions, $a_{1:T}$, our objective is to estimate a sequence of meshes corresponding to each timestep $\bar{M}_{1:T}$. To obtain an accurate estimate of the motion of the clothes, we developed an action-conditioned model-based cloth tracking method that is robust to occlusions. We first simulate the action with the imperfect dynamics model to obtain an initialization of the motion. We then run an optimization to match the visible mesh with the observed point cloud. Finally, we use the dynamics model again to obtain the final prediction.

Initialize the Motion with a Dynamics Model

Our method falls under the category of "model-based tracking" [99, 115, 121]. Compared to model-free tracking, one of the most appealing properties of model-based tracking is that it models the whole object, including the occluded part. This is significant when we track double-layer clothes, because part of the clothes might be occluded through out the entire trajectory. In this case, model-free tracking is only able to estimate the motion of the visible part while leaving occluded portion of the mesh unchanged. On the other hand, model-based tracking can use a physics prior of cloth to estimate the motion of the occluded regions and estimate the configuration of the full mesh that satisfies physical motion constraints.

At each timestep t, we directly modify the position of the picked particle according to the picker action, a_t. We then run the dynamics model dyn for one step to propagate the effect to the whole cloth, holding fixed the position of the picked particle. Suppose $x_t \in \mathbb{R}^{|V|\times 3}$ are the positions of all vertices; given the vertices x_t and action a_t, the dynamics model will predict the next state $x_{t+1}^{dyn} = dyn(x_t, a_t)$. The motion of all vertices, $\Delta x_{t+1}^{dyn} = x_{t+1}^{dyn} - x_t$, will be recorded and used as an initialization for the test-time optimization step.

In order to make the dynamics model as realistic as possible, we calibrate the simulation by searching for the optimal physical parameters(such as friction and stiffness). To do so, we first simulate the entire action sequence $a_{1:T}$ to obtain the final predicted mesh $\hat{M}_T$. We then use a z-buffer to compute the visible portion of the mesh $\hat{M}_T^{vis}$. Next, we run a grid search over the dynamics model paraemeters to minimize the Chamfer distance between the visible portion of the simulated mesh $\hat{M}_T^{vis}$ and the point cloud at the final step P_T. Then we use the optimized parameters to obtain x_{t+1}^{dyn} as explained above. For our dynamics model, we use a position-based cloth dynamics model implemented in the Nvidia FleX simulator [64, 35].

Augment Imperfect Dynamics Model by Aligning with Measurement

Due to the complexity of real-world dynamics and the challenges of system calibration, our dynamics model will have errors. Even with accurate estimation of the initial state, it will deviate from the real-world rollout with errors accumulating over time. To tackle this challenge, we draw inspiration from Bayesian Tracking [2]: we eliminate the compounding errors due to the inaccurate dynamics model by running a test-time optimization (TTO1 in Fig. 4.2) and thereby reduce the discrepancy between the dynamics prediction and the measurement (the observed pointcloud).

From the dynamics model (Sec. 4.3.2), we obtain an initial estimate of the state of cloth x_{t+1}^{dyn} . The goal of the test-time optimization step is to compute a correction term Δx_{t+1}^{corr} that adjusts the predicted mesh to better match the observed point cloud. Thus, we optimize a 3-D translation

for each vertex Δx_{t+1}^{corr} so that $x_{t+1} = x_{t+1}^{dyn} + \Delta x_{t+1}^{corr}$ is aligned with the observation. The specific optimization objectives are:

Chamfer Loss. The first objective we have is the one-way Chamfer distance between the next point cloud P_{t+1} and the visible portion of the mesh x_{t+1}^{vis}, given by $\mathcal{L}_{Chamf} = \mathcal{D}_{chamf}(P_t, x_{t+1}^{vis})$. We use the one-way Chamfer distance because the observed point cloud is incomplete due to occlusions induced by the tweezer. For each point on the observed point cloud, we find the nearest neighbor within the visible set of mesh vertices.

Laplacian Loss. To avoid implausible artifacts in the predicted mesh, we introduce a smoothing term to regularize the mesh. Given the mesh $M_t = (V_t, E_t)$, with vertices V_t and edges E_t, we use $x_{t,k}$ to denote the position of the k-th vertex and $E_{t,i}$ to denote the set of edges that are connected to the k-th vertex. The Laplacian loss pulls each vertex to the average positions of its neighbors:

$$\mathcal{L}_{Lap} = \frac{1}{K} \sum_{k=1}^{K} \left\| \left(\frac{1}{|E_{t,k}|} \sum_{j \in E_{t,k}} x_{t,j} \right) - x_{t,k} \right\|_2^2 \tag{4.1}$$

We also use other losses to regularize the mesh, such as mesh edge loss, collision loss, sparsity loss, and rigidity loss. The details of these additional losses can be found in the appendix. At each step, we optimize Δx_{t+1}^{corr} with respect to the losses defined above for 100 iterations by using Adam optimizer [51].

Rollout Augmented Dynamics with Line Search

In the previous section, we described how to compute a correction term for simulated results based on the measurement. However, the optimized mesh is not guaranteed too satisfy all physical constraints. Similar to [115, 114], we use the dynamics model again to verify the physical plausibility of the mesh. Therefore, rollout the dynamics model again from the original state x_t; this time, we use both the picker action a_t as well as a "pseudo-action" $\tilde{a}_t = \Delta x_{t+1}^{dyn} + \Delta x_t^{corr}$. The picker action is executed as described in Sec. 4.3.2. On the other hand, the pseudo-action $\tilde{a}_t$ is applied to all visible particles. We then use the dynamics model to adjust the positions of the occluded particles.

As mentioned above, although pseudo action $\tilde{a}_t$ helps align the rollout with observation, it may potentially create physically infeasible configurations. Therefore, we run a line search on the correction component Δx_t^{corr}. If the simulation produces an error, we multiply Δx_t^{corr} with a decaying factor γ (in our experiments, we set $\gamma = 0.7$). If the simulation fails for 10 times, we set $\tilde{a}_t = 0$.

After executing the dynamics model, we run another test-time optmization (TTO2) procedure (Alg.1 line 9) to ensure that the estimated mesh $\bar{M}_t$ matches the observation. The losses we use are similar to the test-time optmization (TTO1) in Sec. 4.3.2, excluding the rigidity loss and sparsity loss which are used for regularizing the correction term. Finally, we use optimized mesh as the initial state for the next pick-and-place action and iterate until all pick-and-place actions have been tracked.

4.3.3 Model finetuning

Using the above tracking model, we obtain a pseudo-labeled dataset, with an estimated mesh for each observed point cloud or depth image. Our dataset consists of the final estimated mesh at

Algorithm 1: Pseudo label generation by model-based tracking

Input : A sequence of depth image sequences $\{D_{1:T}\}_{i=0}^{N}$, picker actions $\{a_{1:T}\}_{i=0}^{N}$, point cloud sequence $\{P_{1:T}\}_{i=0}^{N}$, a pretrained mesh reconstruction model f_θ, and a dynamics model dyn.

Output: A pseudo-labeled dataset that includes paired observations and pseudo labels: B=$\{(D_i, \bar{M}^i)\}_{i=0}^{N}$

Reconstruct initial mesh $\bar{M}_0^0$ by reconstruction model f_θ
Initialize the pseudo-labeled dataset B with $(D^0, \bar{M}^0)$
for $i \leftarrow 0$ ***to*** N **do**
    for $t \leftarrow 1$ ***to*** T **do**
        $M_{t+1}^{dyn,i} \leftarrow$ *SimulationStep*$(\bar{M}_{t-1}^i, a_t^i)$,
        $\tilde{a}_t^i \leftarrow$ Compute pseudo action through test-time optimization (Sec. 4.3.2)
        $\bar{M}_t^i \leftarrow$ *SimulationStepWithLineSearch*$(\bar{M}_{t-1}^i, a_t^i, \tilde{a}_t^i)$
    end
    Optimize $\bar{M}_T^i$ with test-time optimization
    Add depth image D_T^i and pseudo mesh $\bar{M}_T^i$ to the pseudo-labeled dataset B
    Use the final mesh in the current iteration as the initialization of next iteration: $\bar{M}_0^{i+1} \leftarrow \bar{M}_T^i$
end
return *Pseudo-labeled dataset B*

the end of each pick and place action $\bar{M}_T$ and the corresponding point cloud P_T. After curating this pseudo-ground-truth dataset, we use it to finetune the mesh reconstruction model. In our experiments, we finetune the mesh-reconstruction model in MEDOR [39], which is built off GarmentNets [19]. Given a depth image, GarmentNets [19] and MEDOR [39] first predict the canonical coordinates of each pixel. They then complete the shape in canonical space and finally transform the completed shape back to observation space. It should be noted that our method not only provides the pseudo ground-truth mesh for the observation space, but our method also can compute a pseudo-ground-truth mesh in the canonical space. This is because GarmentNets [19] and MEDOR [39] simultaneously reconstruct both meshes in the initial configuration; tracking the meshes helps preserve the mapping between canonical space and observation space. Thus, we are able to fine-tune both the model that maps from observation space to canonical space as well as the model that maps from canonical space back to the observation space (i.e. both parts of the mesh reconstruction model). In terms of the GarmentNets [18] componnents, we train the canonicalization model, as well as the shape completion and warp field prediction model. For details, please refer to GarmentNets [18].

4.4 Experiments

Through the experiments, we seek to the answer the following questions:

1. Can our method generate approximately correct pseudo labels for mesh reconstruction and dynamics learning?

2. Can our model adapt quickly after being finetuned on the pseudo labels?

4.4.1 Evaluation on the Quality of Pseudo Labels

Setup. We collect 50 trajectories in the real world, each of which contains 3 pick-and-place actions. Including the initial state, there are 200 pseudo labels in total.

Baselines. We compare our method to 4 baselines:

- **No Pseudo Action**. The goal of pseudo action $\tilde{a}_t$ is to "patch" the inaccuracies of the dynamics model. We verify its effectiveness by removing it from the method and only rollout with the picker action $a_{1:T}$.
- **No Action Conditioning**. In this baseline, we assume the picker action information is not known, which has two implications. 1) When computing the pseudo action, since we don't know the picker action, we cannot roll out the simulator to initialize TTO; 2) When we simulate the pseudo action with line search, we only apply the pseudo action alone (not the picker action).
- **No Dyn Init**. In this baseline, we directly run TTO on the current mesh M_t instead of the simulated next mesh M_{t+1}^{dyn}. This is to verify whether using simulation to bootstrap the optimization is critical to the performance.
- **No Test-time Finetuning (TTO2)**. Although we conduct TTO in between the simulation process, the rollout may still drift due to imperfect dynamics. In this baseline, we remove the optimization step at the end of the tracking procedure (TTO2) to see whether this component is necessary.

Metrics. Evaluating the quality of pseudo label is challenging, due to the absence of ground-truth mesh. To evaluate the quality of generated pseudo label, we use the bidirectional Chamfer distance between the visible surface of pseudo mesh and the observed point cloud, which we refer as *Chamfer PC*. Our assumption is that if the tracking is accurate, then the visible surface of the pseudo mesh should match the observed point cloud, which is the visible surface of ground-truth mesh. We compute the metric with the mesh before Test-time optimization 2. This is because optimizing a point set is too flexible. Even for completely erroneous prediction, the shape can match the observation and achieve a low cost, even if the structure is completely wrong. Therefore, comparing the loss after TTO2 is not very indicative of accurate tracking.

Results. In the Fig. 4.4, we show the side-by-side comparison of the tracking results of the different methods. Videos and 3D visualizations of the pseudo mesh can be found on our anonymized website. Our full method (second row) is able to track the clothes even under complicated configurations, i.e., multiple folds (right figure). In contrast, the other methods all produce pseudo meshes that don't align with the observations.

Method	Chamfer PC (1×10^{-4})
No Pseudo Act	2.22 ± 1.8
No Dyn Ini	1.82 ± 1.72
No Act Cond	8.73 ± 7.28
No TTO2	2.20 ± 2.49
MEDOR	3.0 ± 2.1
Ours (full method)	$\mathbf{1.46 \pm 1.75}$

Table 4.1: Quantitative results of different variants of our method.

In Table. 4.1, we show the quantitative results of the baselines and our method. Means and standard deviations of the generated pseudo meshes are computed across the whole dataset. Comparing our method with *No Pseudo Act*, we see that dynamics errors can be significantly reduced by aligning with measurements during the rollout. By comparing *No Dyn Ini* with our method, we see the importance of using

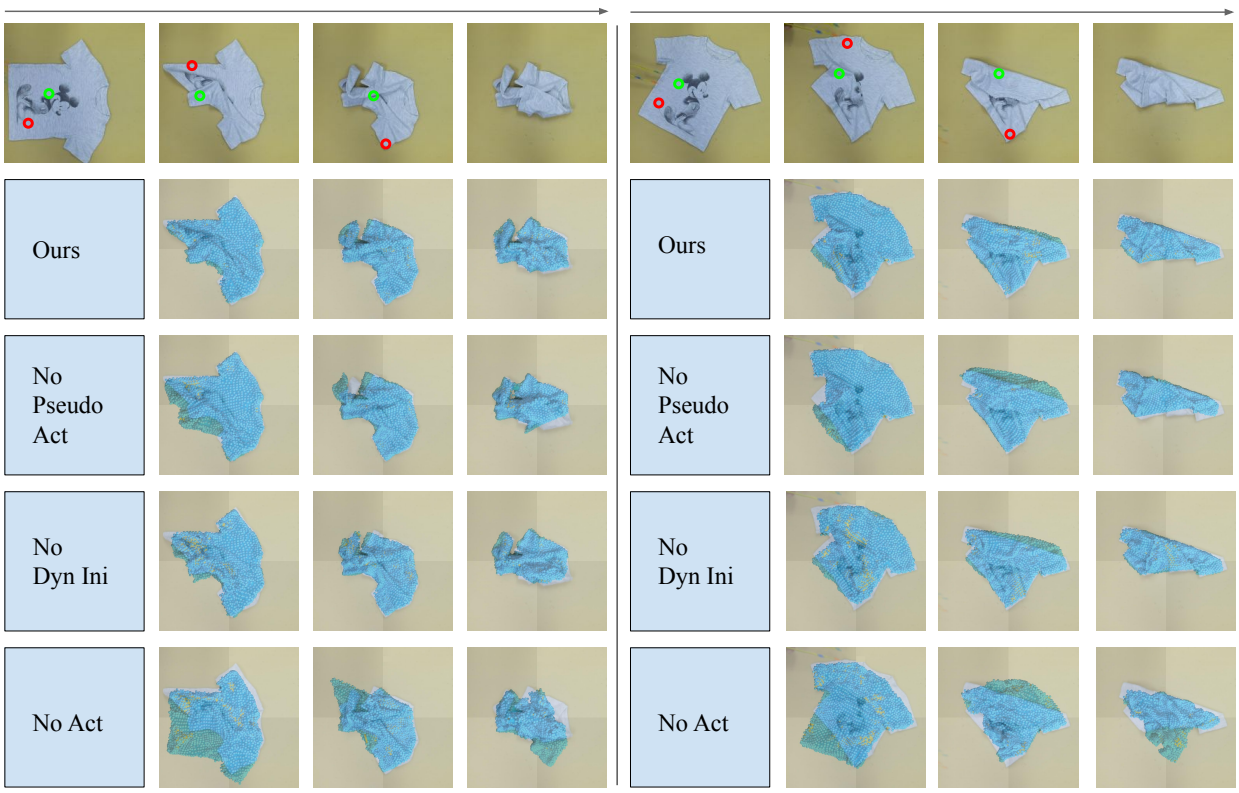


Figure 4.4: Qualitative results of pseudo label generation. First row is the real-world rollout, with pick points and place points denoted by red and green circles respectively.

dynamics prior as the initialization for the optimization problem. Since tracking is a correspondence problem, our conjecture is that initializing with a dynamics rollout makes it easier to find correspondences. Looking at *No Act Cond*, we see the benefits for tracking deformable objects conditioned on the action, and a pure optimization method failed in this case. Comparing *No TTO2* with our method, we observe that TTO2 helps reduce the compounding error; thus removing it hurts the performance.

4.4.2 Model Performance After Finetuning

In this section, we investigate whether the pseudo label generated from our proposed workflow is beneficial for the self-supervised learning of a mesh reconstruction model. We finetune MEDOR [39] which is purely trained in simulation (code obtained from the authors), and show its performance before and after finetuning.

Method	Chamfer PC	Chamfer Mesh
MEDOR [39] w/o ft	3.0 ± 2.1	2.2 ± 1.7
MEDOR [39] w/ ft	1.6 ± 1.4	1.2 ± 0.8

Table 4.2: Performance of the method before and after finetuning. We show that after finetuning, the model is able to achieve lower chamfer distance to the observed point cloud and L2 distance to validation set of pseudo label.

In Table 4.2, we show 2 metrics. The *Chamfer PC* is the bidirectional Chamfer distance between the visible predicted mesh and the partial point cloud, which is the same metric as in Table. 4.1. *Chamfer PC* indicates how well the predic-

tion aligns with the observation. *Chamfer Mesh* is the chamfer distance between the full predicted mesh and pseudo mesh, which indicates how well the model learns from this pseudo dataset. As we can see from the Table 4.2, both metrics are significantly improved after finetuning (46% for *Chamfer PC* and 45% for Chamfer Mesh).

4.4.3 Limitations

Although our proposed method provides a viable way to train a mesh reconstruction model in the real world in a self-supervised manner. There are several limitations: 1) The data collection system relies on a human (instead of using a robot) due to the difficulty of using a robot for flattening and grasping arbitrary clothes reliably. 2) We record full RGB-D videos during data collection, while only use depth information for tracking. Potentially, our method can set up an extra optimization objective based on RGB information [42, 10]. 3) We don't leverage of the intermediate full meshes generated during tracking, which can potentially be used for a learned dynamics model.

Chapter 5

Conclusion

In this thesis, we investigate how to enable to reason about occlusion to attain more efficient cloth manipulation. Specifically, we studied the problem of reasoning about occlusion both implicitly and explicitly.

In the first part of thesis, we describe how to leverage inductive bias to build a more generalizable dynamics model. Concretely speaking, we infer a visible connectivity graph from the partial point cloud and learn a particle-based dynamics model. Since partial point cloud only captures the structure of the visible parts, the unmodeled occluded regions confounds the dynamics model. To alleviate this issue, we propose graph imitation learning to build an occlusion-aware dynamics model.

In the second part of the paper, we introduce a cloth manipulation system that explicitly reasons about the occluded regions of cloth. At test-time, we optimize the predicted mesh with self-supervised losses. Then we use a learned mesh-based dynamics model to plan over the predicted mesh and find optimal actions for cloth manipulation. We compare against state-of-the-art cloth manipulation methods that do not account for partial observability and show significant improvements from explicit occlusion reasoning and test-time fine-tuning. We also demonstrate the efficacy of our method in real world experiments.

Lastly, we proposed a self-supervised mesh reconstruction in the real world, via action-conditioned cloth tracking. We show that by leveraging a dynamics model and point cloud observations, we can accurately track the clothes and compute pseudo-labels of the reconstructed mesh for crumpled clothes. By finetuning a simulation-trained mesh reconstruction model on the pseudo labels, we can partially close the sim2real gap and improve the reconstruction results in the real world.

Chapter 6

Future Work

In this thesis, we investigate how to reason about occlusion in different ways, and use it for model-based cloth manipulation. Although showing impressive performance, there remains several challenges that are left for future work.

- Visuo-tactile cloth grasping. One of the failure modes in the physical experiment is the cloth grasping. Due to the thin structure of the cloth, precise grasping of the cloth is very difficult. For example, the gripper may collide with the cloth and result in deformation. It will be interesting to see how to integrate visual and tactile information to grasp cloth more intelligently.
- Probabilistic occlusion reasoning. One property of the problem of occlusion reasoning is ambiguity. Since the configurations of the occluded part cannot be fully determined by the visible surface, there is a whole distribution of possible poses for the occluded part. Therefore, one interesting direction for future work might be modeling the uncertainty and predict the distribution of possible configurations.
- Occlusion-aware policy learning. All the methods presented in this paper rely on the dynamics model. While the model-based method is effective and performs well, it is usually very slow. One solution for the issue is to train a model-free policy. It will be interesting to see whether explicit occlusion reasoning is also helpful for model-free method, or implicit one is already good enough.

Appendix A

Appendix for VCD

A.1 VCD Implementation

A.1.1 GNN Architecture

As mentioned in the main paper, we take the network architecture in previous work [97] (referred to as GNS) for our dynamics GNN G_{dyn} and the edge GNN G_{edge}. Both GNN consists of three parts: encoder, processor and decoder. Since the dynamics and edge GNNs have very similar architectures, we first describe the architecture of the dynamics GNN, and then describe how the edge GNN architecture differs from that.

Input: The input to the dynamics GNN is a graph, where the nodes are the points in the voxelized point cloud of the cloth, and the edges consist of the collision edges (built using Eq. (1)) and mesh edges (inferred by a trained edge GNN). The node feature for a point v_i consists of the concatenation of its past m velocities, a one-hot encoding of the point type (picked or unpicked - see details about picking in Section 3.4 in the main paper and Section A.1.3 in the appendix), and the distance to the table plane. For edge e_{jk} that connects nodes v_j and v_k, its edge feature consists of the distance vector $(x_j - x_k)$, its norm $||x_j - x_k||$, a one-hot encoding of the edge type (mesh edge or collision edge), and the current displacement from the rest position $||x_j - x_k|| - r_{jk}$, where r_{jk} is the distance between x_j and x_k at the rest positions. The displacement from the rest positions are set to zero for collision edges which do not have rest positions.

We now describe how the robot action is incorporated into the input graph of the dynamics GNN as follows. As mentioned in Section 3.4 of the main paper, when we want to use the dynamics GNN to predict the effect of a pick-and-place robot action $a = \{a_{pick}, a_{place}\}$ on the current cloth, we first decompose the high-level action into a sequence of low-level movements, where each low-level movement is a small delta movement of the gripper and can be achieved in a short time. Specifically, we generate a sequence of small delta movements $\Delta x_1, ..., \Delta x_H$ from the high-level action, where $x_{pick} + \sum_{i=1}^{H} \Delta x_i = x_{place}$. Each delta movement Δx_i moves the gripper a small distance along the pick-and-place direction and the motion can be predicted by the dynamics GNN in a single step. We then incorporate the small delta movement into the input graph as follows. When the gripper is grasping the cloth, we denote the picked point as u. We assume that the picked point is rigidly attached to the gripper; thus, when considering the effect of the t^{th} low-level movement of the robot gripper, we modify the input graph by directly setting the picked point u's position $x_{u,t} = x_{pick} + \sum_{i=1}^{t} \Delta x_i$ and velocity $\dot{x}_{u,t} = \Delta x_i / \Delta t$, where Δt is the time for one low-level movement step. The dynamics GNN will then propagate the effect of the robot action along the graph when predicting future states.

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

Processor: The processor consists of L stacked Graph Network (GN) blocks [5] that update the node and edge embedding, with residual connections between blocks. We use $L = 10$ in both edge GNN G_{edge} and dynamics GNN G_{dyn}. The l^{th} GN block contains an edge update MLP f_e^l and a node update MLP f_p^l that take as input the edge and node embedding g^l and h^l respectively and outputs updated embedding g^{l+1} and h^{l+1} (we denote g^0 and h^0 as the edge and node embedding output by the encoder). It also contains a global update MLP f_c^l that takes as input a global vector embedding c^l, and outputs the updated global embedding c^{l+1}. The initial global embedding c^0 is

set to be 0. For each GN block, first the edge update MLP updates the edge embedding; it takes as input the current edge embedding g_{jk}^l, the node embedding h_j^l, h_k^l for the nodes that it connects, as well as the global embedding c^l: $g_{jk}^{l+1} = f_e^l(h_j^l, h_k^l, g_{jk}^l, c^l) + g_{jk}^l, \ \forall e_{jk} \in E$. The node update MLP then updates the node embedding; its input consists of the current node embedding h_i^l, the sum of the updated edge embedding for the edges that connect to the node, and the global embedding c^l: $h_i^{l+1} = f_p^l(h_i^l, \sum_j g_{ji}^{l+1}, c^l) + h_i^l, \ \forall i = 1, ..., N_p$. Note the edge and node updates both have residual connections between consecutive blocks. Finally, the global update MLP takes as input the current global embedding c^l, the mean of the updated node and edge embedding, and updates the global embedding as: $c^{l+1} = f_c^l(c^l, \frac{1}{|V|}\sum_{i=1}^{|V|} h_i^{l+1}, \frac{1}{|E|}\sum_{e_{jk}} g_{jk}^{l+1})$.

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

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

Hyperparameters In simulator, we set the radius of particles to be 0.00625, an All MLPs that we use has three hidden layers with 128 neurons each and use ReLU as the activation function. The detailed parameters of the GNN architecture, as well as the simulator parameters, can be found in Supplementary Table B.5.

A.1.2 VCD Training Details

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

For training the edge GNN, we need to obtain the ground-truth of which collision edges are also mesh edges. During simulation training, a collision edge is assumed to be a mesh edge if the mapped simulation particles of the edge's both end points are connected by a spring in the simulator.

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

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

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

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

Training parameters: We use Adam [51] with an initial learning rate of 0.0001 and reduce it by a factor of 0.8 if the training plateaus. We train with a batch size of 16. The training of the dynamics GNN takes roughly 4 days to converge on a RTX 2080 Ti. The training of the edge GNN usually converges in 1 or 2 days. Detailed training parameters can be found in Supplementary Table B.5.

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

Supplementary Table A.1: Summary of all hyper-parameters.

A.1.3 VCD Planning Details

We summarize the planning procedure of VCD in Algorithm 3. We sample K high-level pick-and-place actions. For each sampled high-level action, we roll out our dynamics model using that action for H low-level steps and obtain the sequence of predicted point positions.

Action sampling during planning in simulation As described in the main text, we sample 500 pick-and-place actions, where the pick point is first uniformly sampled from a bounding box of the cloth and then projected to be on the cloth mask. For generating the bounding box, we first obtain the cloth mask from the simulator. We then obtain the minimal and maximal pixel coordinates u, v value of the cloth mask. The bounding box is the rectangle with corners $(min(u) - padding, min(v) - padding)$ and $(max(u) + padding, max(v) + padding)$, where padding is set to be 30 pixels for the 360×360 image size we use. We use rejection sampling to make sure the place point is within the image to keep the action within the depth camera view. The unnormalized direction vector $p = (\Delta x, \Delta y, \Delta z)$ (y is the up axis) of the pick-and-place is uniformly sampled as follows: $\Delta x, \Delta z \in [-0.5, 0.5]$, and $\Delta y \in [0, 0.5]$. The vector is normalized and then the distance

Algorithm 2: Planning with pick-and-place actions

input : Voxelized partial point cloud P, Edge GNN G_{edge}, Dynamics GNN G_{dyn}, number of sampled actions K

output: pick-and-place action $a = \{x_{pick}, x_{place}\}$

Build collision edges E_0^C with P; Infer mesh edges $E^M \leftarrow G_{edge}(P, E_0^C)$

for $i \leftarrow 1$ ***to*** K **do**

  Sample a pick-and-place action x_{pick}, x_{place}

  Compute low-level actions $\Delta x_1, ..., \Delta x_H$

  Get picked point v_u from x_{pick}

  Pad historic velocities with 0: $\mathbf{x}_0 \leftarrow P, \dot{\mathbf{x}}_{-m...0} \leftarrow \mathbf{0}$

  for $t \leftarrow 0$ ***to*** H **do**

    Build collision edges E_t^C with $\mathbf{x}_t$

    Move picked point according to gripper movement by :

    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t,\ \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$

    Predict accelerations using G_{dyn}: $\ddot{\mathbf{x}}_t \leftarrow G_{dyn}(\mathbf{x}_t, \dot{\mathbf{x}}_{t-m...t}, u, E^M, E_t^C)$

    Update point cloud predicted positions & velocities:

    $\dot{\mathbf{x}}_{t+1} = \dot{\mathbf{x}}_t + \ddot{\mathbf{x}}_t \Delta t,\ \mathbf{x}_{t+1} = \mathbf{x}_t + \dot{\mathbf{x}}_{t+1} \Delta t$

    Readjust picked point according to gripper movement by

    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t,\ \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$

  end

  Compute reward r based on final point cloud predicted position $\mathbf{x}_H$

end

return *pick and place action with maximal reward*

is separately sampled from $[0.05, 0.2]$ meters. We decompose the pick-and-place action into 10 low-level actions and wait for another 6 steps for the cloth to stabilize.

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

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

A.2 Baselines Implementation

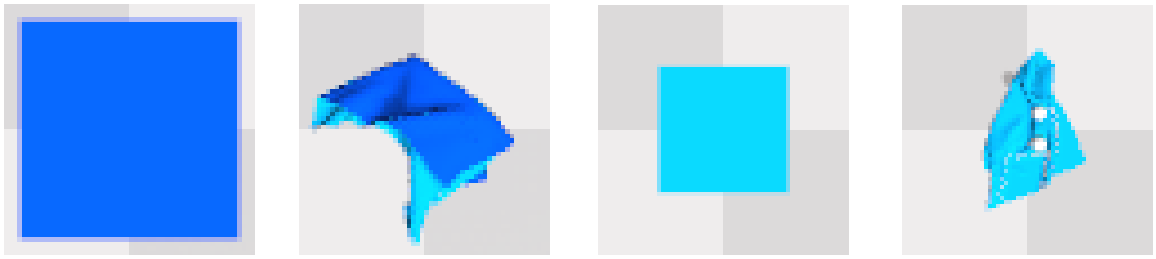
For all the baselines, we try our best to adjust the SoftGym cloth environment to match the cloth environment used in the original papers. For VSF, we place the camera to be top-down and zoomed in so that the cloth covers the entire image when fully flattened. We also changed the color of the cloth to be bluish as in the original paper. We collect 7115 trajectories, each consisting of 15 pick-and-place actions for training the VSF model (same as in the VSF paper). For CFM, we also use a top-down camera and change the color of the cloth to be the same on both sides, following the suggestion of the authors (personal communication). We collect 8000 trajectories each consisting of 50 pick-and-place actions for training the contrastive forward model (same as in the CFM paper). For MVP, we collect 5000 trajectories each with 50 pick-and-place actions and report the performance of the best performing model during training. We trained each of the baselines for at least as many pick-and-place actions as they were trained in their original papers. For training our method, , we collect 2000 trajectories, each consisting of 1 pick-and-place action decomposed into 20 low-level actions for training. Note that this is fewer pick-and-place actions than any of the baselines used for training. We now describe each compared baseline in more details below:

A.2.1 VisuoSpatial Forsight (VSF)

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

Image: Following the original paper, we use images of size 56×56. we place the camera to be top-down and zoomed in so that the cloth covers the entire image when fully flattened. An example goal image of the smoothed cloth for VSF is shown in Supplementary Figure A.1.

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



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

Supplementary Figure A.1: Images of cloth configurations used by the baseline methods.

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

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

A.2.2 Contrastive Forward Model (CFM)

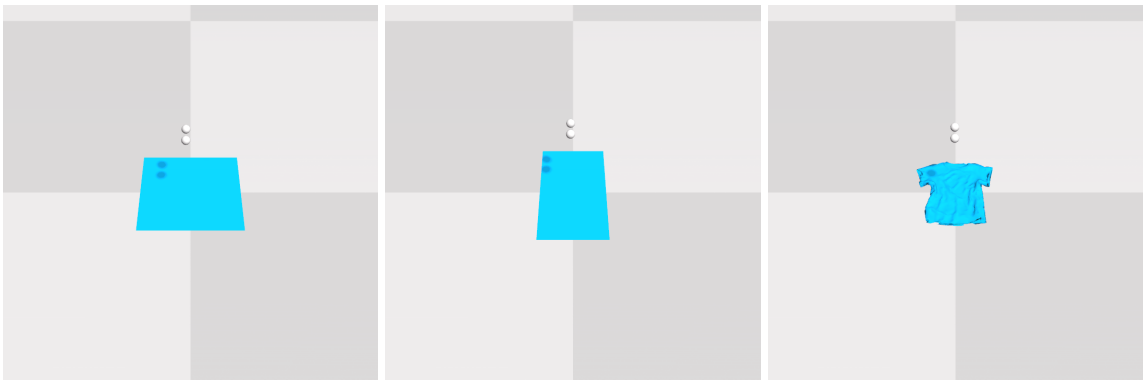
We use the official code of CFM provided by the authors[2].

Image: Following the original paper, we use images of size 64×64. We also place the camera to be top-down and adjust the camera height so the cloth contains a similar portion of the image as in the original paper. Following the suggestions from the authors (personal communication), we also set the color of the cloth to be the same on both sides. See Supplementary Figure A.1 for an example of the images we use.

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

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

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



Supplementary Figure A.2: Images of the square cloth, rectangular cloth, and t-shirt used in simulation.

A.2.3 Maximal Value under Placing (MVP)

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

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

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

Action space: The action space for the MVP policy is in 5 dimension: $(u, v, \Delta x, \Delta y, \Delta z)$, where u, v is the image coordinate of the pick point and is sampled for the segmented cloth pixel. We use the depth information to back project the pick point to 3d space. $(\Delta x, \Delta y, \Delta z)$ is the displacement of the place location relative to the pick point and is clipped to be within 0.5. Additionally, the height Δy is clipped to be non-negative.

A.3 Experimental Setup

A.3.1 Simulation Setup

We use the Nvidia Flex simulator, wrapped in SoftGym [64], for training. In SoftGym, the robot gripper is modeled using a spherical picker that can move freely in 3D space and can be activated so the nearest particle will be attached to it. For the simulation experiments, we use a nearly square cloth, composed of a variable number of particles sampled from $[40, 45] \times [40, 45]$; this corresponds to a cloth of size in the range of $[25, 28] \times [25, 28]$ cm. Detailed cloth parameters such as stiffness are listed in the appendix. For all methods, we randomly generate 20 initial cloth configurations for training. The initial configurations are generated by picking the cloth up and then dropping it on the table in simulation. For evaluation, we consider three different geometries: 1) the same type of square cloth as used in training; 2) Rectangular cloth. The length and width of the rectangular cloth is sampled from $[19, 21] \times [31, 34]$ cm. 3) T-shirt. Images of these three shapes of cloth in simulation are shown in Supplementary Figure A.2.

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

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

A.3.2 Real-world Setup

Real World Setup Our real robot experiments use a Franka Emika Panda robot arm with a standard panda gripper. We obtain RGBD from a side view Azure Kinect camera and crop the RGBD image into the size of [345, 425], which corresponds to a workspace of 0.4 x 0.5 meters. To obtain the cloth point cloud, we first use color thresholding to remove the table background and obtain the cloth segmentation mask and then back project each cloth pixel to 3d space using the depth information. We evaluate on three pieces of cloth: Two squared towels made of silk and cotton respectively and one shirt made of cotton. We use the covered area as described in Sec. A.1.3 as our reward function.

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

A.4 Additional Experimental Results

A.4.1 Simulation Experiments

Normalized Improvement and Normalized Coverage in Simulation

NI and NC of our simulation experiments are reported in Supplementary Table B.4 and Supplementary Table A.3. With different metrics, our method consistently outperforms all baselines.

	# of pick-and-place actions / Method	5	10	20	50
Square	(Ours)	0.624 ± 0.217	0.778 ± 0.222	0.968 ± 0.307	1.000 ± 0.043
	-graph-imitation (Ours)	$\mathbf{0.692 \pm 0.258}$	$\mathbf{0.919 \pm 0.377}$	$\mathbf{0.990 \pm 0.122}$	$\mathbf{1.000 \pm 0.039}$
	VSF [38]	0.321 ± 0.112	0.561 ± 0.127	0.767 ± 0.134	0.968 ± 0.021
	CFM [127]	0.053 ± 0.051	0.077 ± 0.053	0.109 ± 0.066	0.105 ± 0.106
	MVP [126]	0.399 ± 0.210	0.435 ± 0.137	0.421 ± 0.361	0.307 ± 0.310
Rectangular	(Ours)	$\mathbf{0.585 \pm 0.359}$	$\mathbf{0.918 \pm 0.413}$	0.973 ± 0.341	0.979 ± 0.399
	-graph-imitation (Ours)	0.556 ± 0.372	0.912 ± 0.393	$\mathbf{0.985 \pm 0.164}$	$\mathbf{1.000 \pm 0.028}$
	VSF [38]	0.268 ± 0.090	0.356 ± 0.163	0.542 ± 0.177	0.715 ± 0.162
T-shirt	(Ours)	$\mathbf{0.595 \pm 0.279}$	0.533 ± 0.285	0.738 ± 0.465	$\mathbf{0.979 \pm 0.399}$
	-graph-imitation (Ours)	0.595 ± 0.385	$\mathbf{0.633 \pm 0.357}$	$\mathbf{0.838 \pm 0.450}$	0.969 ± 0.8860
	VSF [38]	-0.009 ± 0.125	0.004 ± 0.188	0.176 ± 0.237	0.219 ± 0.218

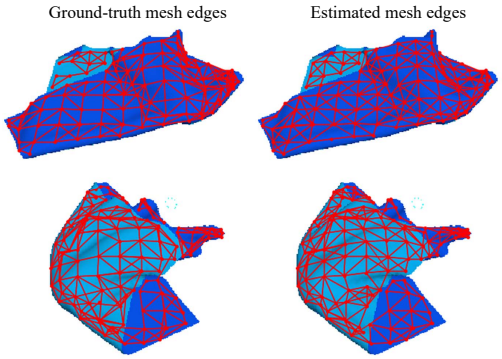
Supplementary Table A.2: Normalized Improvement (NI) of all methods in simulation, for varying numbers of allowed pick and place actions.

Visualization of Edge GNN

We compare predictions of our edge prediction model with the ground-truth edges used for training the edge model in Supplementary Figure A.3. As shown, the edge GNN prediction reasonably matches the ground-truth, and thus well captures the cloth structure; it can also correctly disconnect

	Method \ # of pick-and-place actions	5	10	20	50
Square	(Ours)	0.776 ± 0.132	0.872 ± 0.128	0.985 ± 0.1873	1.000 ± 0.023
	-graph-imitation (Ours)	$\mathbf{0.837 \pm 0.150}$	$\mathbf{0.966 \pm 0.236}$	$\mathbf{0.994 \pm 0.076}$	$\mathbf{1.000 \pm 0.021}$
	VSF [38]	0.629 ± 0.053	0.762 ± 0.093	0.878 ± 0.090	0.984 ± 0.010
	CFM [127]	0.445 ± 0.052	0.466 ± 0.044	0.494 ± 0.031	0.538 ± 0.044
	MVP [126]	0.667 ± 0.121	0.667 ± 0.124	0.661 ± 0.194	0.609 ± 0.179
Rectangular	(Ours)	$\mathbf{0.785 \pm 0.182}$	$\mathbf{0.957 \pm 0.233}$	0.985 ± 0.183	0.998 ± 0.017
	-graph-imitation (Ours)	0.768 ± 0.191	0.949 ± 0.215	$\mathbf{0.992 \pm 0.080}$	$\mathbf{1.000 \pm 0.015}$
	VSF [38]	0.622 ± 0.078	0.664 ± 0.078	0.765 ± 0.119	0.860 ± 0.072
T-shirt	(Ours)	0.837 ± 0.107	0.828 ± 0.096	0.897 ± 0.150	$\mathbf{0.991 \pm 0.189}$
	-graph-imitation (Ours)	$\mathbf{0.867 \pm 0.143}$	$\mathbf{0.901 \pm 0.179}$	$\mathbf{0.960 \pm 0.218}$	0.991 ± 0.331
	VSF [38]	0.636 ± 0.086	0.653 ± 0.090	0.676 ± 0.079	0.698 ± 0.075

Supplementary Table A.3: Normalized coverage (NC) of all methods in simulation on the regular cloth, for varying numbers of allowed pick and place actions.



Supplementary Figure A.3: The edge prediction result of our edge GNN. Red lines visualize the ground-truth (left) or inferred (right) mesh connections.

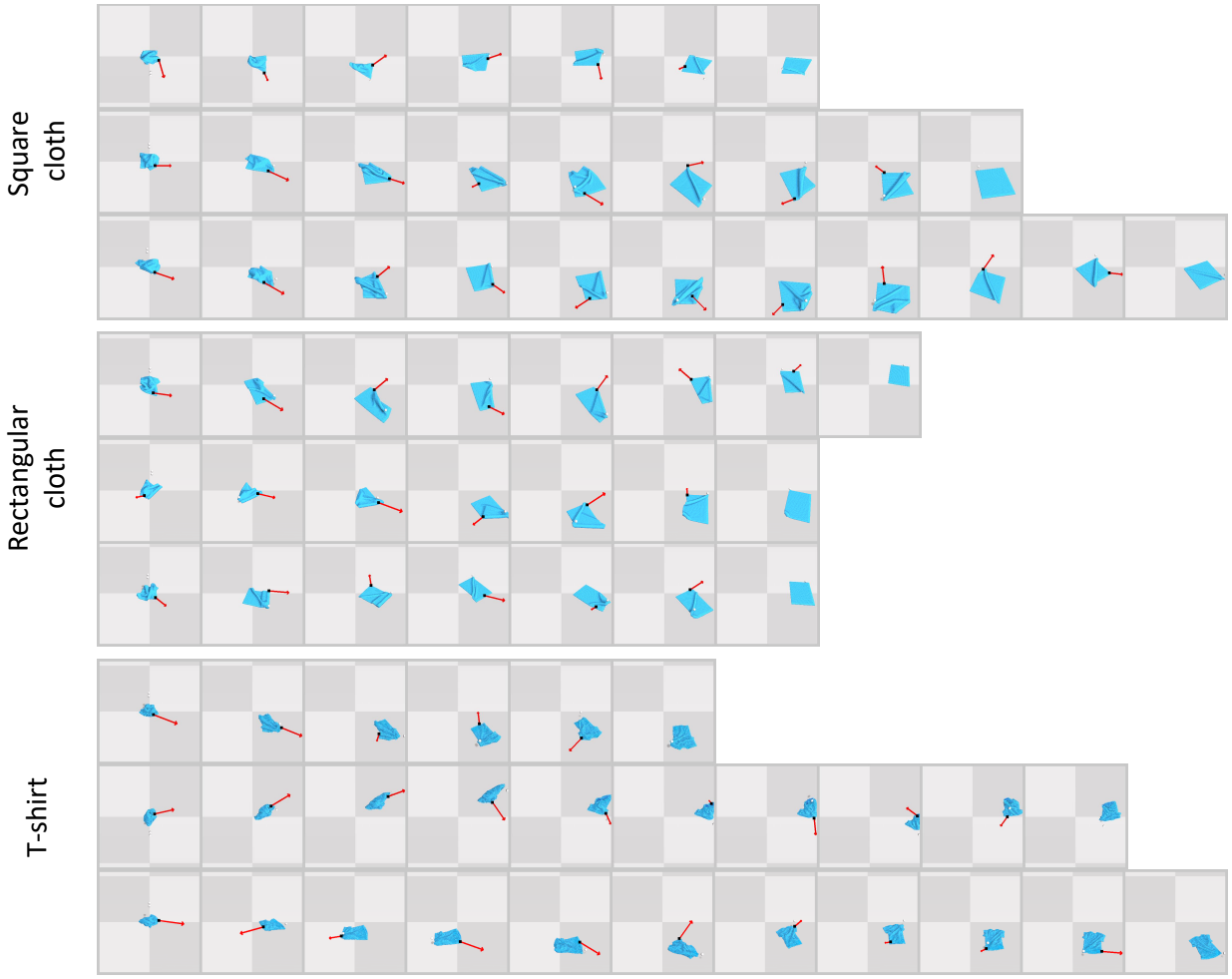
the top layer from the bottom layer when the cloth is folded, e.g., the top left part of the first example and the bottom right part of the second example. Note our method uses only the point cloud as input and the color in this figure is only used for visualization. The edge GNN is trained on the same dataset as the dynamics GNN (described in Sec. A.1.2), and on the validation set, it achieves a prediction accuracy of 0.91.

Visualizations of Planned Actions in Simulation

Supplementary Figure A.4 shows three planned pick-and-place action sequences of in simulation. As shown, VCD successfully plans actions that gradually smooths the cloth. We observe note that VCD favours picking edge / corner points and pulling outwards, which is an effective smoothing strategy, demonstrating the effectiveness of VCD for planning.

Visualizations of Open-loop Predictions in Simulation

In order to understand better what our model is learning, we visualize the prediction of our model compared to the simulator output in Supplementary Figure A.5, A.6, A.7. Given a pick-and-place action decomposed into 75 low-level actions, the model is given the 5^{th} point cloud in the trajectory with the past 4 historical velocities, and the dynamics model is used to generate the



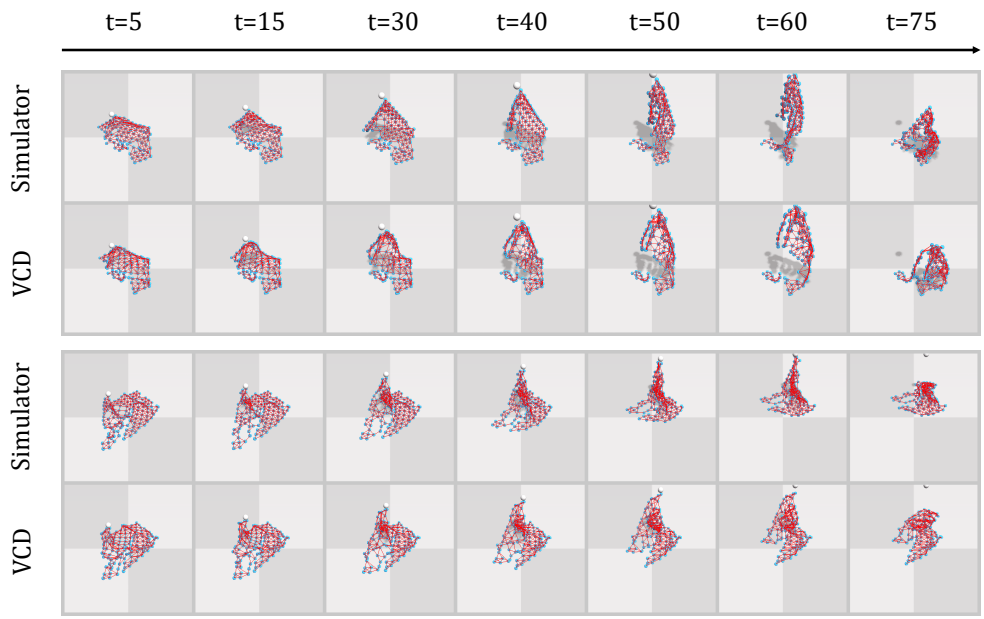
Supplementary Figure A.4: Three example planned pick-and-place action sequences for square cloth, rectangular cloth, and t-shirt. All trajectories shown achieve a normalized improvement above 0.98.

future predictions. As shown, even if the prediction horizon is as long as 70 steps, is able to give relatively accurate predictions on all cloth shapes, indicating the effectiveness of incorporating the inductive bias of the cloth structure into the dynamics model.

A.4.2 Robot Experiments

Running Time

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



Supplementary Figure A.5: Two open-loop predictions of on square cloth. Blue points are observable particles/point cloud points and red lines are mesh edges. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by , in which the mesh edges are inferred by the edge prediction GNN.

Normalized Coverage (NC) of Robot Experiments

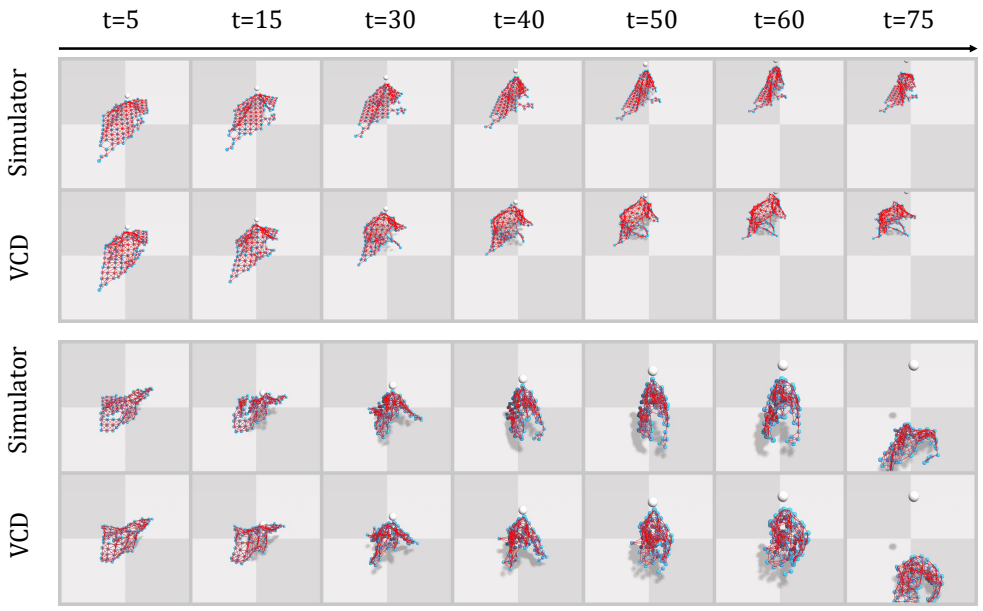
For the robot experiments, the main text reports the normalized improvement (NI). NC are reported here in Supplementary Table A.4.

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

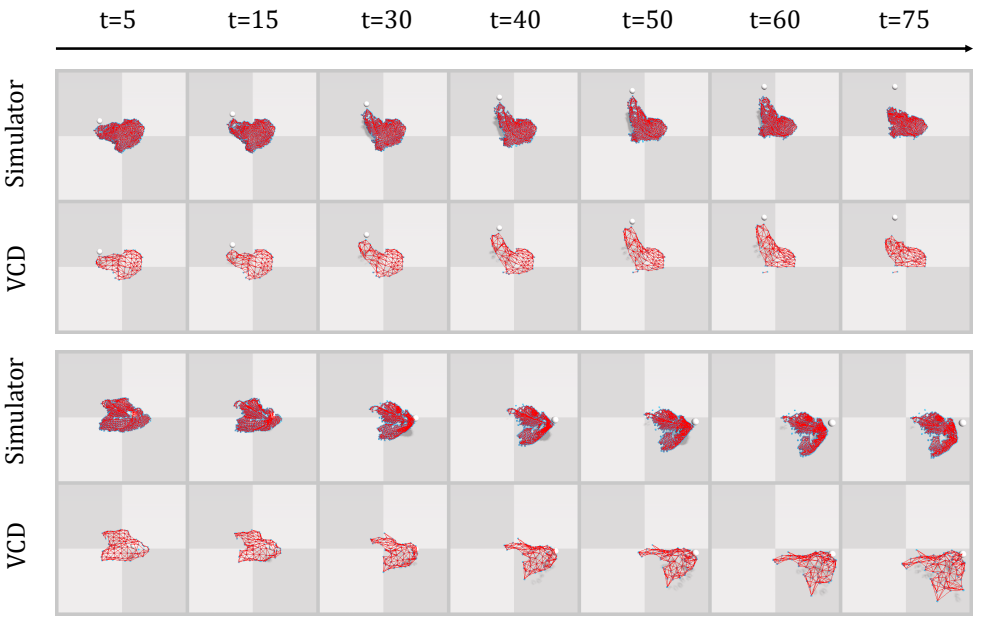
Supplementary Table A.4: Normalized coverage (NC) of VCD in the real world.

Visualization of Sampled Actions in The Real World

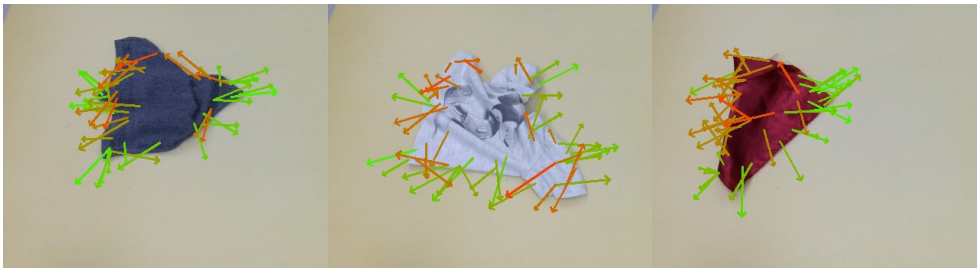
We show in Supplementary Figure A.8 VCD's predicted score for each of the sampled action during smoothing of the cloth. Interestingly, though there is no explicit optimization for this, favours picking corner or edge points and pulling outwards, which is a very natural and effective strategy for smoothing. This demonstrates the effectiveness of for planning.



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



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



Supplementary Figure A.8: Examples of 50 sampled actions used for planning. Each arrow goes from the 2D projection of the pick location to that of the place location. The actions with the higher predicted reward are shown in greener color and the actions with the lower predicted reward are shown in redder colors.

A.5 Planning with VCD for Cloth Folding

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

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

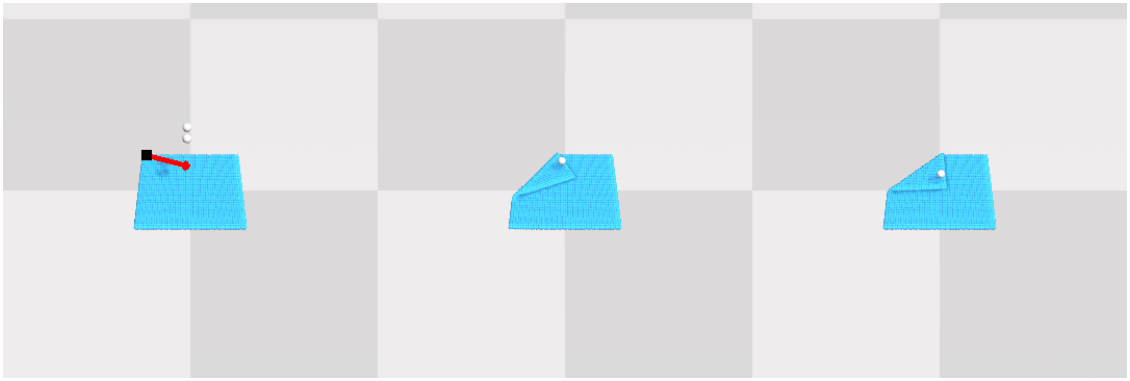
We evaluate VCD on three goals as shown in Supplementary Figure A.9, A.10, A.11: (1) one-corner-in, which folds one corner of the square cloth towards the center; (2) diagonal, which folds one corner of the square cloth towards the diagonal corner; (3) arbitrary, which folds one corner of the square cloth towards the middle point of the opposite edge. For evaluation, we report the average particle distance between the achieved cloth state and the goal cloth state. The numerical results are shown in Supplementary Table A.5 and the qualitative results are shown in Supplementary Figure A.9, A.10, A.11.

As the result shows, VCD can be applied for folding with the above three ways for goal specification. For the ground-truth goal specification and cost computation using simulator particles, VCD performs fairly well for folding (average particle error within 0.3 - 1.3 cm, also see Supplementary Figure A.9, A.10, A.11 for qualitative results). With goal specification via point cloud and Chamfer distance as the cost, the performance of VCD is also reasonable (average particle error 0.3 - 2 cm, also see below for qualitative results), making it a practical choice to apply VCD for folding in the real world.

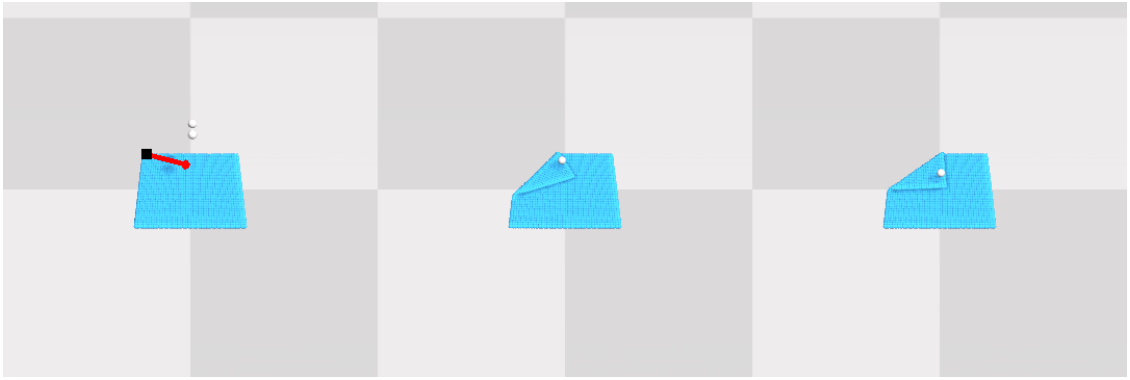
We also note that this VCD model is trained with random pick-and-place actions; the folding performance could be further improved if we add bias (such as corner grasping) during data collection to train VCD with more folding motions.

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

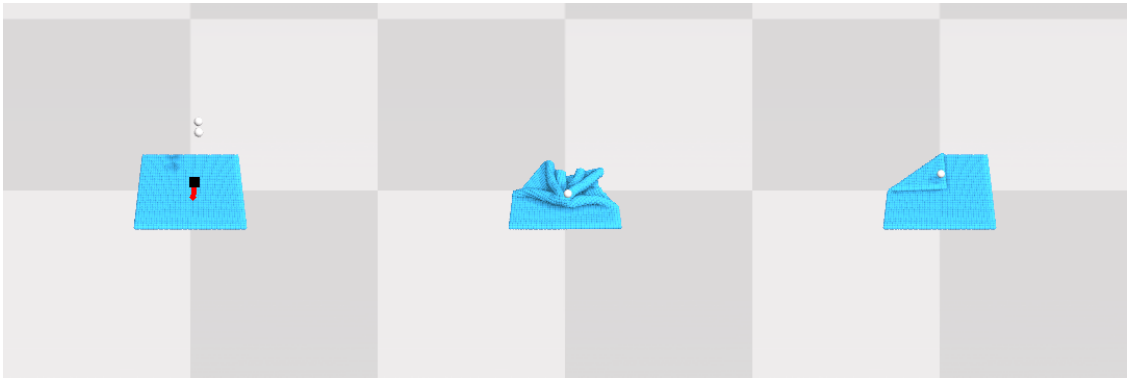
Supplementary Table A.5: Average particle distance (mm) between final achieved cloth state and goal cloth state.



(a) Cost: groundtruth mapping

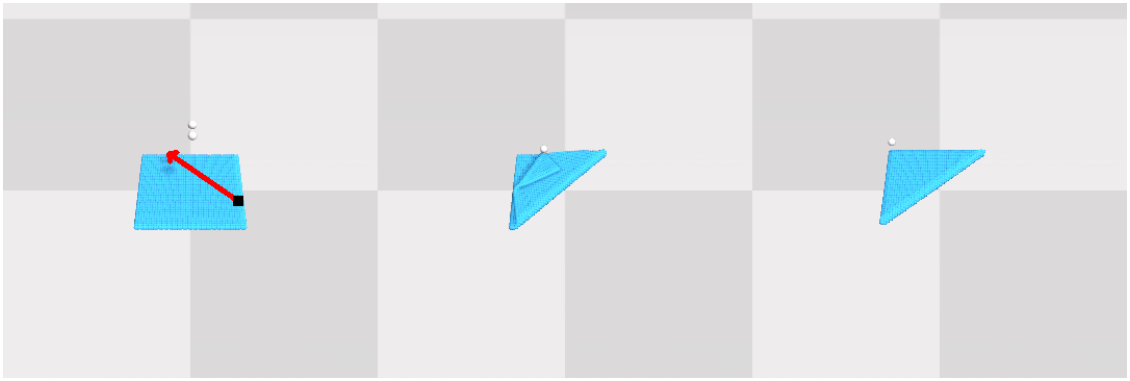


(b) Cost: Chamfer distance

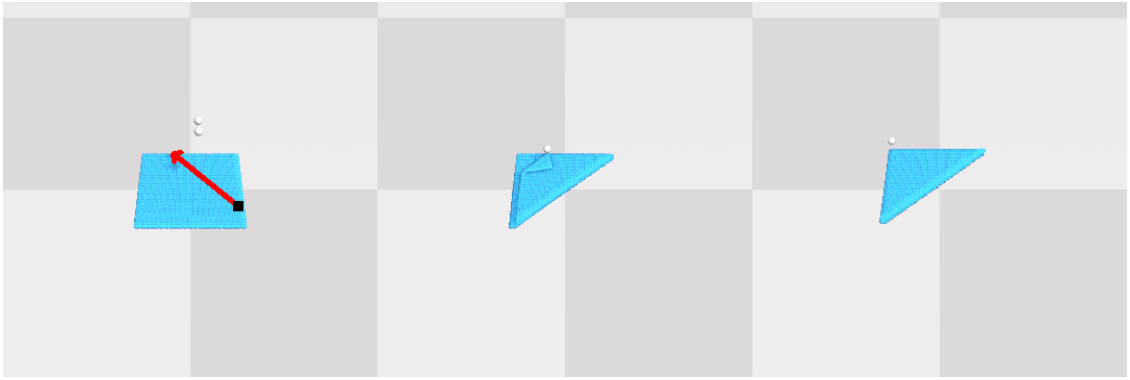


(c) Cost: IOU

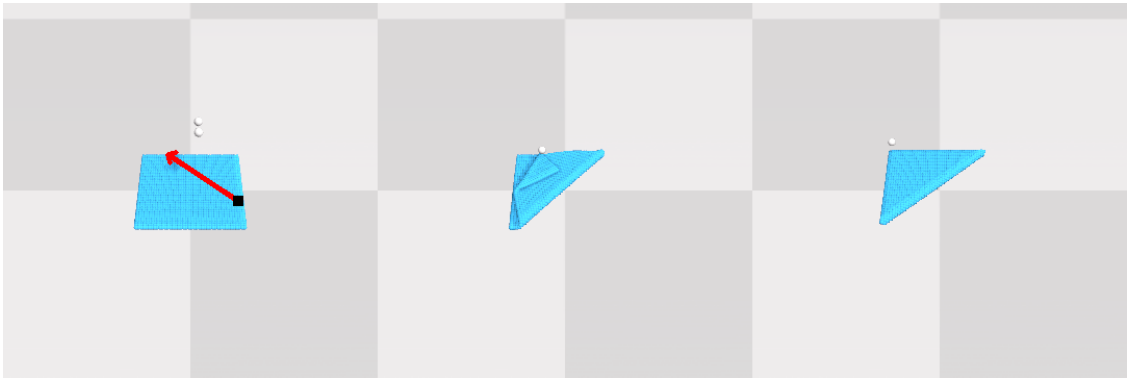
Supplementary Figure A.9: VCD for folding, one-corner-in goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.



(a) Cost: groundtruth mapping

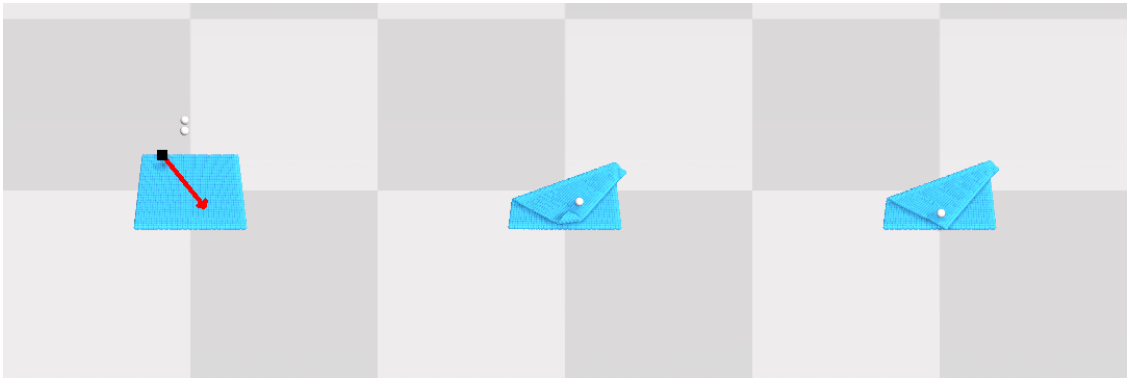


(b) Cost: Chamfer distance

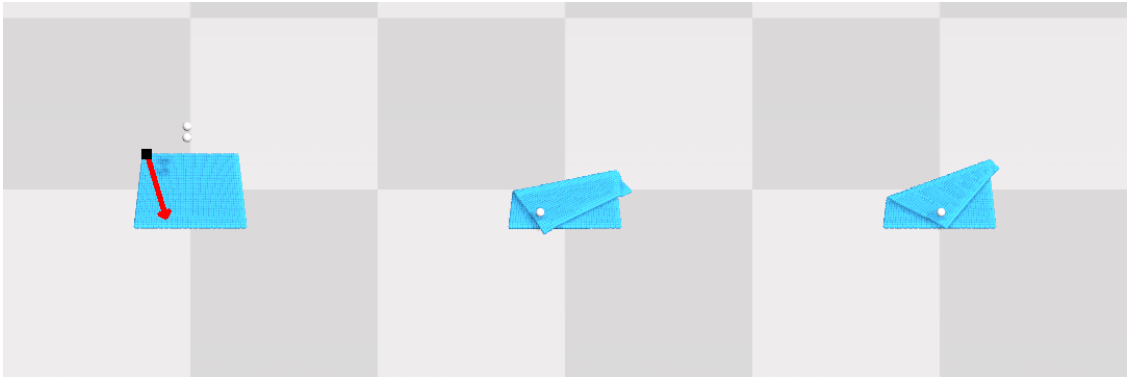


(c) Cost: IOU

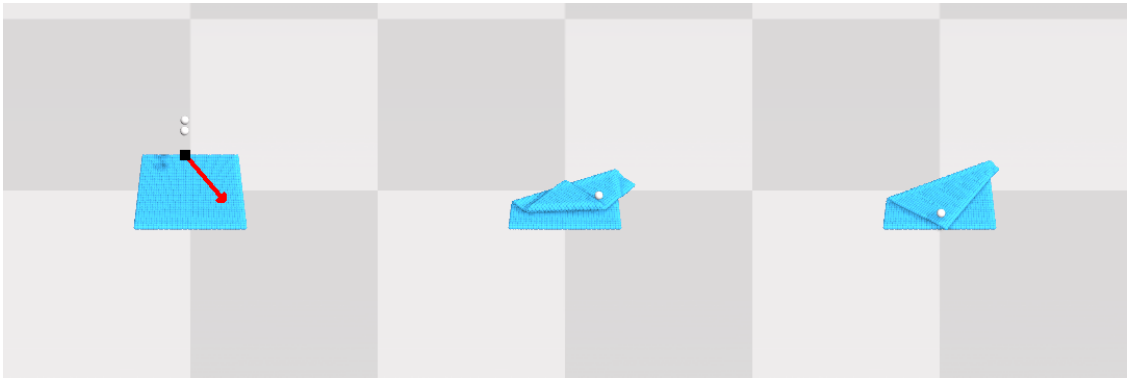
Supplementary Figure A.10: VCD for folding, diagonal goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.



(a) Cost: groundtruth mapping



(b) Cost: Chamfer distance

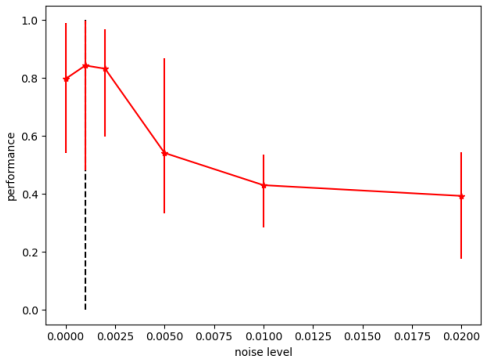


(c) Cost: IOU

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

A.6 Robustness to Depth Sensor Noise

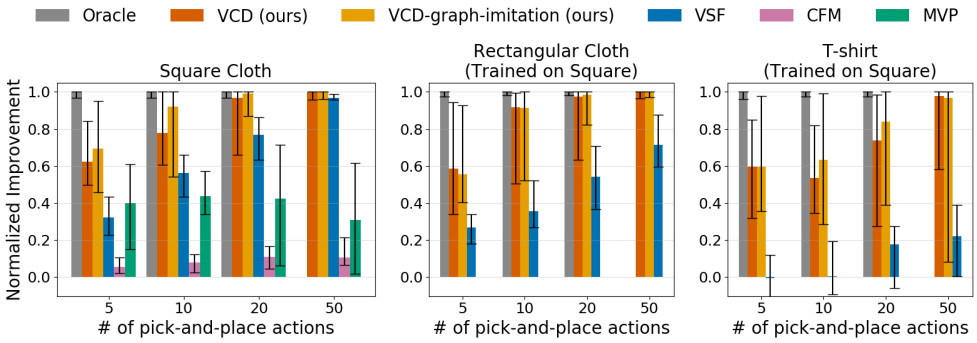
When deployed in the real world, VCD might suffer from the depth camera noise. To investigate this, we manually add different levels of noise (Gaussian noise with different levels of variance) to the depth map in the simulation and test VCD's planning performance (with a maximal number of 10 pick-and-place actions). The result is shown in Supplementary Figure A.12. The dashed vertical line is the noise level of Azure Kinect depth camera that we use in the real world, as measured by Michal et al. [118]. As shown, VCD is quite robust within the noise range of the Azure Kinect depth sensor.



Supplementary Figure A.12: Normalized Improvement of VCD under different levels of depth sensor noise, with a maximal number of 10 pick-and-place actions for smoothing. The vertical dashed line represents the typical level of Azure Kinect noise, which is the depth sensor that we use for the real-world experiment. The error bars show the 25% and 75% percentile.

A.7 Comparison to Oracle using the FleX Cloth Model

How good can the system be if we know the full cloth dynamics? To answer this question, for our simulation experiments (shown in Supplementary Figure A.13), we additionally show the performance of an oracle that uses the FleX cloth model for planning in Supplementary Figure A.13. Here, oracle uses the same planning method as VCD and achieves perfect results in different clothes. This shows that better performance can be achieved if the full cloth model and dynamics can be better estimated, which we leave for future work.

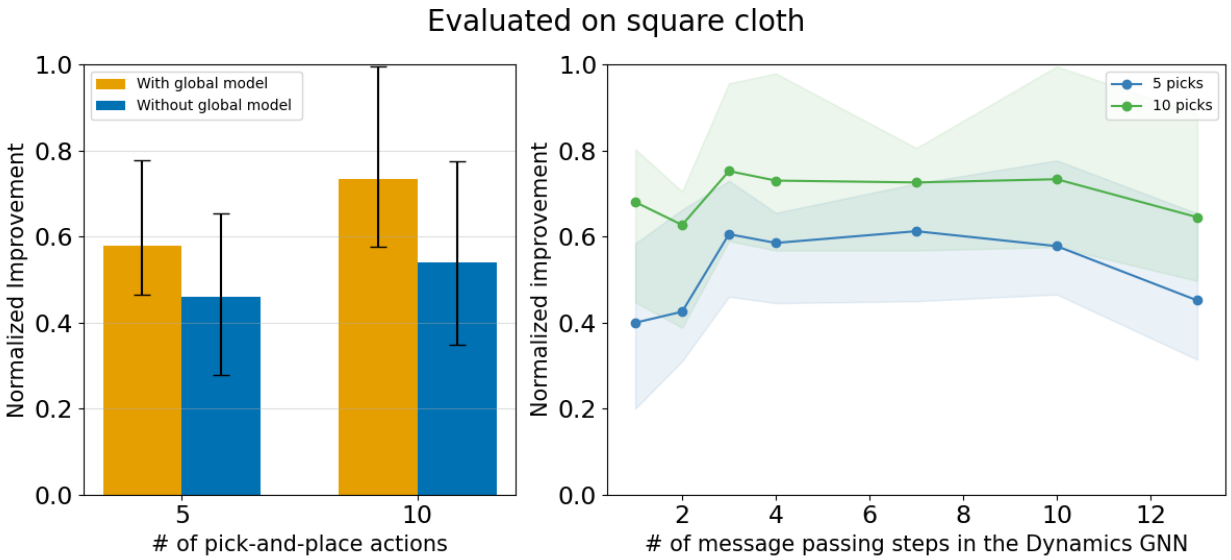


Supplementary Figure A.13: Normalized improvement on square cloth (left), rectangular cloth (middle), and t-shirt (right) for varying number of pick-and-place actions. The height of the bars show the median while the error bars show the 25 and 75 percentile.

A.8 Ablations on architectural choices

For our edge and dynamics GNNs, we adopt the model architecture from GNS [97], as described in Appendix A.1.1. In Sanchez-Gonzalez, et al [97], a comprehensive analysis on architectural design decisions for the GNS model was investigated. We modify the GNS architecture by adding a global model in each GN block of the processor, which has the potential to speed up the propagation of information across the graph. The global model has been widely used in previous works in graph neural networks [5, 122, 36]. Supplementary Figure A.14 (left) shows that using a global model in the dynamics model yield better planning performance than without it.

We also evaluate the sensitivity of our dynamics model to the number of message passing steps (L). As shown in the right figure of Supplementary Figure A.14, our dynamics model is robust to a broad range of values for the number of message passing steps. We speculate that, when the number of message passing is too small, the effect of action cannot propagate to the particles that are distant from the picked point. With too many message passing steps, the model is prone to overfitting. Nonetheless, Supplementary Figure A.14 (right) shows that there is a broad of values for the number of message passing steps that lead to similar performance; thus, our model is fairly robust to this parameter.



Supplementary Figure A.14: We evaluate the effects of a global model and the number of message passing steps in the dynamics GNN on the square cloth. The left figure shows that the usage of a global model is helpful to the planning performance. The right figure shows that our model is generally robust to the number of message passing steps as long as the number lies within the range of [3, 10].

Appendix B

Appendix for MEDOR

B.1 Experiments Setup

B.1.1 Simulation Setup

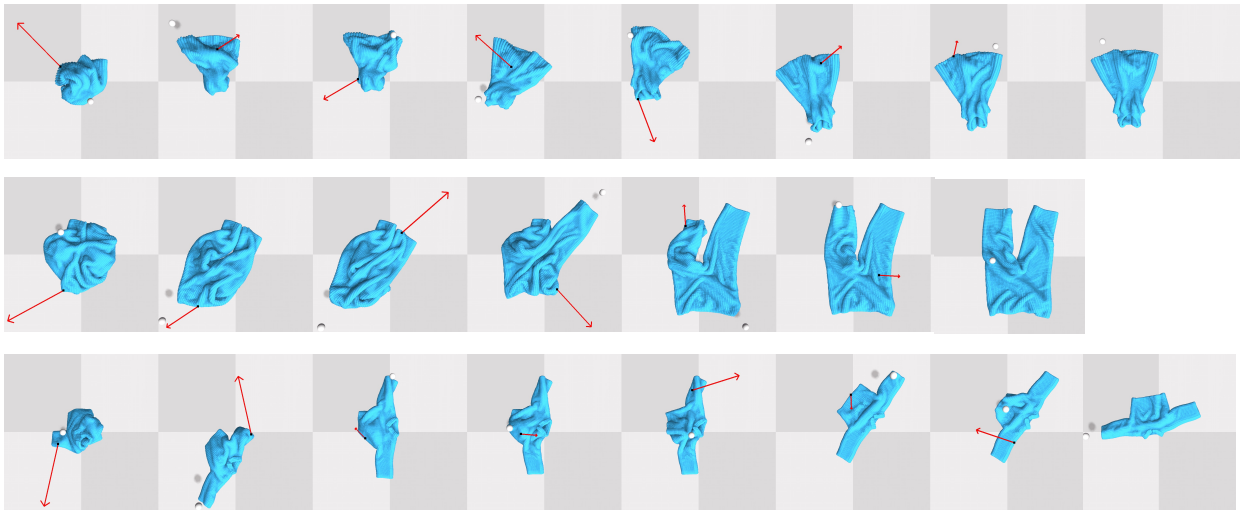
We conduct all simulation experiments in Softgym [64], a simulation environment for deformable objects built on the particle-based simulator, Nvidia Flex. We model a flying gripper as a spherical picker that can move freely in the 3D space. When a cloth particle is "picked", it will move rigidly with the gripper. The simulation parameters of softgym can be found in Table. B.1. We obtained the 3D cloth models from CLOTH3D dataset [6]. Considering the physical experiments, we rescale the cloth models so that they could fit in the workspace of our real robot. Also, to make the GNN dynamics computationally feasible, we create a downsampled version of the cloth models. During data collection, we still use the original dense mesh in softgym, but register the downsampled mesh onto the dense mesh by finding the nearest neighbor of each vertex. Therefore, we obtain the trajectory of downsampled mesh and use that for dynamics learning. The detailed specifications of the preprocessed CLOTH3D dataset can be found in Table. B.3. We use a top-down camera and it's placed at a fixed height of 0.65 m. The valid workspace is 0.53 m $\times$ 0.53 m.

Flattening For flattening task, the goal is to maximize the coverage of the cloth in the current configuration. To compute the coverage, we treat each node on the graph as a sphere with a radius of 0.005 and compute the covered area when projected to the ground plane.

Canonicalization In CLOTH3D, the canonical pose of the cloth is defined to be the pose of cloth when wore by a T-pose human. However, since we are considering cloth manipulation on a planar surface, it is not achievable and cannot be used as the goal pose directly. Therefore, we use gravity (10 times the gravity on Earth) to obtain a flattened version of the canonical pose. Another thing that needs to be handled is the ambiguity caused by reflection symmetry and rotation symmetry. For example, trousers are approximately 180° rotation symmetry, which means that the shapes before and after rotating around the center axis for 180° are the same. Therefore, during the evaluation of cloth canonicalization, we define a canonical goal set $\mathcal{G} = \{G_i^{N\times 3}\}_{i=1...A}$, for each type of cloth. A is the number of valid canonical poses. The cost is computed as the minimum of average pairwise distance to each of the possible canonical poses. Suppose that the current configuration of the cloth is $V^{N\times 3}$, where N is the number of vertices, g_j and v_j is the j-th vertex of the mesh, the cost is computed as

$$Cost_{canon} = \min_{G_i \in \mathcal{G}} \sum_{j=n}^{N} \frac{(g_j - v_j)^2}{N} \tag{B.1}$$

The rotation symmetry of each category is described in Table. B.2. It should be noted that since we maintain a discrete set of plausible goal pose, although skirt has infinite order of rotation symmetry, we cannot iterate over all possible cases. Instead, we make an approximation that the order of rotation symmetry of skirt is 12. The order of rotation symmetry is the times the shape fits onto itself when rotating for 360 degrees. An order of 2 means that the shape remains unchanged when rotating for 180 or 360 degrees.

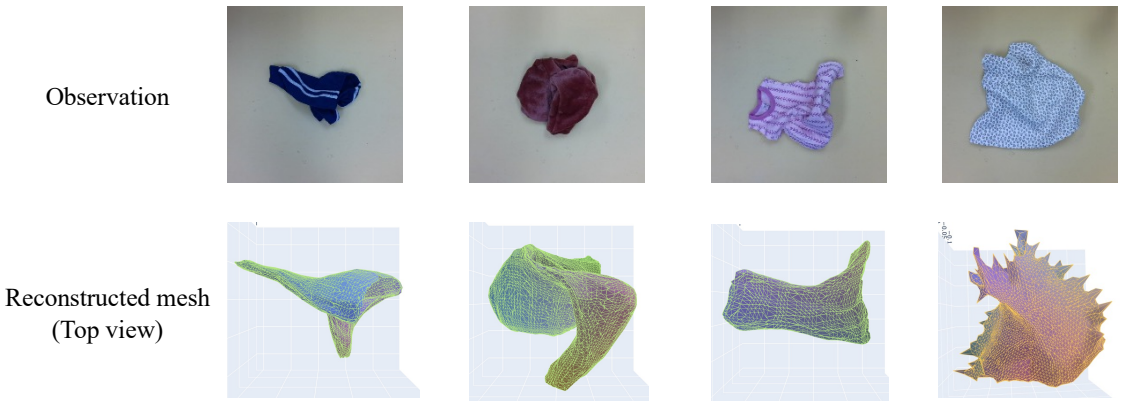


Supplementary Figure B.1: Examplar trajectories of canonicalization task in simulation. As we can see, our method is able to quickly unfold the cloths from extremely crumpled configurations in a few steps.

B.2 Additional results

B.2.1 Simulation Experiments

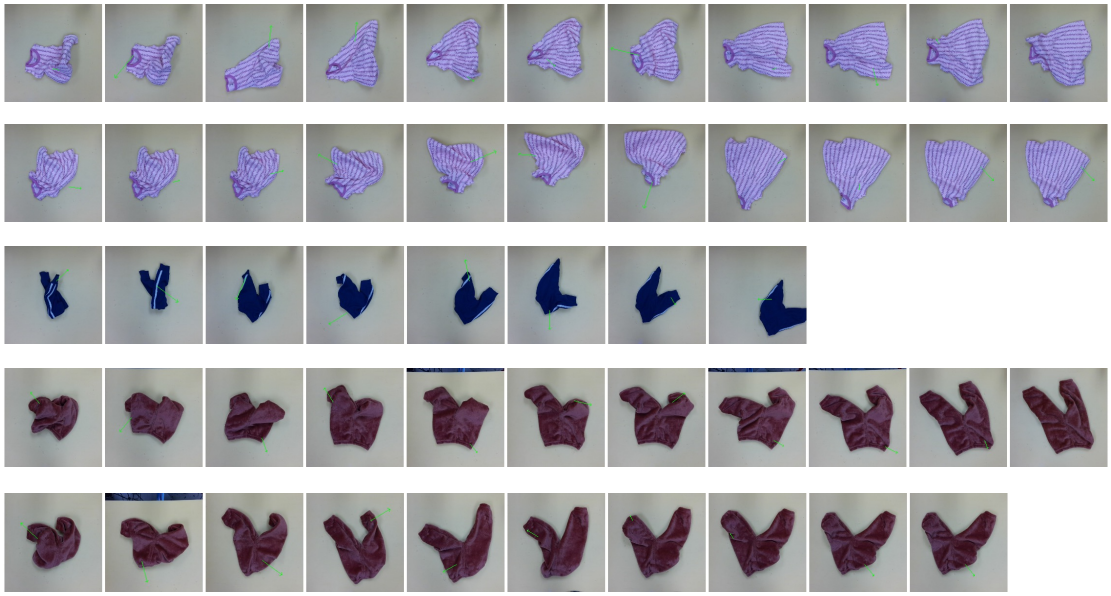
The full results of simulation are shown in the Table. B.4, best performance within each category is bolded. We also show several trajectories of cloth canonicalization in Fig. B.1.



Supplementary Figure B.2: Reconstruction results in real world.

B.2.2 Physical Experiments

Some additional results of physical experiments are shown in Fig. B.2 and Fig. B.3.



Supplementary Figure B.3: Rollout of cloth flattening in real world.

B.3 Implementation details of MEDOR

B.3.1 GarmentNets-style Mesh Reconstruction Model

As described in the main paper, we employ a GarmentNets-style model to reconstruct the mesh from depth image. GarmentNets formulates the pose estimation problem of clothes as a shape completion task in the canonical space. By doing so, the model learns meaningful correspondence between There are several advantages about GarmentNets: 1) it reconstructs the occluded part of the clothes due to self-occlusion; 2) it estimates the correspondence between clothes in canonical space and observation space; 3) it estimates the pose of the clothes (per-vertex location).

Model Architecture

We build our mesh reconstruction model based on GarmentNets [19] with some decisive modifications to make it fit in our setup. Here, we give a brief description of the reconstruction pipeline and the modifications we made. For details, please refer to GarmentNets [19].

Canonicalization Given the observation at current state, we first use *canonicalization model* to map it to canonical space by conducting pixel-wise canonical coordinate prediction. We don't pick up the clothes for state estimation, because it may disrupt the configuration of partially folded or almost smooth clothes. We use depth images captured by a top-down camera as the observation and High-Resolution Network (HRNet-32) [110] as the backbone of canonicalization model. HRNet is a convolutional architecture that specializes in producing high-resolution and spatially precise representations. It uses smaller convolutional kernels and avoids overly downsampling the feature map. This is critical for our task because our model needs to infer the structure of crumpled clothes by subtle changes at the contour of different layers. The architecture change improves the performance on cloth smoothing and canonicalization tasks for 75%. Following GarmentNets [19],

we formulate the prediction problem as a classification task by dividing each axis into 64 bins and use Cross-entropy loss for training. **Feature Scattering** Given the predicted canonical coordinate, we scattered the features of each pixel to corresponding locations in the canonical space. The aggregated feature volume is further transformed by a 3D UNet [23], which is trained by shape completion and flow prediction.

Shape Completion Before estimating the pose of occluded surface, we first perform volumetric shape completion in the canonical space, which helps capture the shared structure within the same category. Since the structure of clothes are thin and non-watertight, GarmentNets [19] proposes to use Winding Number Field [43] as the shape representation. The shape completion network is instantiated as an implicit network. It takes the dense feature produced by 3D UNet and a canonical coordinate and output the winding number field in that coordinate.

Predicting pose in the observation space After we complete the shape of clothes in the canonical space, we estimate the pose of the clothes in observation space. We cast it as a 3D flow prediction problem by predicting per-vertex flow that transforms the clothes from canonical space to observation space.

$$\tilde{x}_i^o = \tilde{x}_i^c + \tilde{f}_i \tag{B.2}$$

Training and Testing Details

. Now we describe how we train and test the model.

Dataset collection. Our model is category-specific, so we collect a dataset for each category separately in Softgym [64]. We obtained 3D clothes models from CLOTH3D dataset [6].

Each dataset contains 4,000 trajectories of length 5 for training and 400 for testing. The At the beginning of each trajectory, we randomly sample a clothes mesh and initialize it by random drop or flattened pose with equal probability. Then we disrupt the clothes with random pick-and-place actions. Random actions are biased towards picking corners (obtained by Harris corner detection[**?**]) 90% of the time, otherwise it is sampled uniformly on the clothes. The distance between the pick point and the place point is uniformly sampled between [25, 150] pixels.

Training details The model is trained in two-stage. In the first stage, we train the canonicalization network with Cross-entropy loss till convergence. In the second stage, we freeze the canonicalization network and train the rest of the models for shape completion and flow prediction by Mean-square error.

Inference details At test-time, given a depth image, we first use canonicalization network and 3D UNet to obtain dense feature volume in the canonical space. Then we discretize the canonical space into 128x128x128 grid and evaluate shape completion network at every cell. To retrieve the mesh, we compute the Gaussian derivatives for the predicted winding number field and run Marching Cube algorithm [**?**].

B.3.2 Dynamics Model

Similar to VCD [65], we use a learned GNN-based dynamics model proposed in GNS [97]. The difference between VCD and ours is that we don't have a GNN edge model because edges are estimated by our mesh reconstruction model. The original mesh models in CLOTH3D [6] (see Table. B.3) are too dense that the rollout becomes computationally infeasible. Therefore, we down-

sample the mesh by using Vertex Clustering [68] with a voxel size of 0.025m. For the complete list of hyperparameters of the GNN dynamics model, please refer to Table B.6.

B.3.3 Planning

The planning algorithm is outlined in Algorithm 3. We plan in the space of pick-and-place primitive with horizon equal to 1. To simulate the effect of each pick-and-place, we divide them into low-level actions and roll out by the dynamics model in parallel. Following [65], the action is encoded into the input mesh by directly modifying the position and picked point, and the displacement will be propagated to the rest of mesh during message passing.

Action sampling during planning For both cloth flattening and canonicalization, we bias the actions sampling toward the contour of the cloth. More specifically, we first obtain the bounding box of the cloth in current observation and expand it by 30 pixels in each direction. Then we randomly sample picked points within the the bounding box region. For points that are not on the cloth, we map them to the nearest point on the cloth. The place direction is uniformly sampled in all directions and place distance is sampled uniformly from [0.05, 0.2]. A dummy action which corresponds to "no action" is added to the list of candidate actions. We sample 500 pick-n-place actions at each timestep.

Reward computation For flattening, we treat each mesh vertex as a sphere of radius 0.01, and the total reward is the covered area of the projection of all vertices to the ground plane. For canonicalization, we rotate the predicted canonical pose according to the predefined rotation symmetry (see Table. B.2). For each valid canonical pose, we input it into the simulator to flatten by gravity, which constitutes a predicted goal set. The reward is computed as negative of smallest distance to goals in goal set. We use pairwise l2 distance.

B.4 Implementation Details of Baselines

B.4.1 VisuoSpatial Forsight (VSF)

We use the official codebase of VSF[1]. Image: we train the model with RGB-D images of size 56 x56 pixels, according to the original paper. To reproduce the performance of the original paper, we collected 7115 trajectories with 15 pick-and-place actions for each category. In total, it amounts to 100,000 environment steps, which is 5 times the data compared to our methods.

Action sampling We use a similar action sampling strategy as MEDOR for VSF, that is, bounding box sampling B.3.3. We use a smaller padding size (6 pixels) because of the smaller image size.

Reward computation For flattening, we use color thresholding to compute the coverage of cloth. For canonicalization, we project the predicted and goal RGB-D image into 3D rgb point cloud. The RGB values are scaled to be similar to the coordinate values. Then we run ICP [130] for 5 iterations to align the predicted point cloud with the goal point cloud.

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

Algorithm 3: Planning pipeline of MEDOR

input : Depth Image D, partial point cloud P, mesh reconstruction Model ϕ, dynamics GNN G_{dyn}, number of sampled actions K

output: pick-and-place action $a = \{x_{pick}, x_{place}\}$

Estimate the full mesh of clothes by mesh reconstruction model: $\tilde{M}_{init} = \phi(D)$

Perform test-time fine-tuning: $\tilde{M}^0_{tuned} = finetune(\tilde{M}_{init}) = (\tilde{V}^0, \tilde{E}_M)$.

for $i \leftarrow 1$ ***to*** K **do**

  Sample a pick-and-place action x_{pick}, x_{place}

  Compute low-level actions $\Delta x_1, ..., \Delta x_H$

  Get picked point v_{picked} from x_{pick}

  Pad historic velocities with 0: $\mathbf{x}_0 \leftarrow \tilde{V}^0, \dot{\mathbf{x}}_{-m...0} \leftarrow \mathbf{0}$

  for $t \leftarrow 0$ ***to*** H **do**

    Build collision edges E^t_C with $\mathbf{x}_t$

    Move picked point according to gripper movement by :

    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t,\ \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$

    Predict accelerations using G_{dyn}: $\ddot{\mathbf{x}}_t \leftarrow G_{dyn}(\mathbf{x}_t, \dot{\mathbf{x}}_{t-m...t}, E_M, E^t_C)$

    Update point cloud predicted positions & velocities:

    $\dot{\mathbf{x}}_{t+1} = \dot{\mathbf{x}}_t + \ddot{\mathbf{x}}_t \Delta t,\ \mathbf{x}_{t+1} = \mathbf{x}_t + \dot{\mathbf{x}}_{t+1} \Delta t$

    Readjust picked point according to gripper movement by

    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t,\ \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$

  end

  Compute reward r based on final mesh nodes position $\mathbf{x}_H$

end

return *pick and place action with maximal reward*

B.4.2 Visible Connectivity Dynamics (VCD)

We use the official code of VCD, with modifications so that it works well on our dataset.

Given point cloud and mesh, the original VCD conduct bipartite matching to map point cloud to mesh nodes. If the corresponding mesh nodes are connected by mesh edges, we also construct mesh edges for the point cloud points. We found that this approach is highly sensitive to density of point cloud and mesh. Imagine the mesh is denser than the point cloud, there might be many mesh vertices between two adjacent points on point cloud. Thus they are not connected although they should.

To solve this issue, we design a more robust approach for mesh edges construction that is agnostic to the density of mesh. First, for each point on the point cloud, we find the nearest mesh vertex. Then we compute the distance between neighboring points on point cloud by the pairwise geodesic distance of corresponding mesh vertices. A mesh edge is constructed if the geodesic distance is below a threshold.

Training To makes a fair comparison, we train the edge GNN dataset of different categories separately. Each dataset contains 20,000 envrironment steps, which is same as the dataset used for training the mesh reconsctruction model of MEDOR. For dynamics model, similar to MEDOR, we train a single model on Trousers dataset but use it for all categories at test-time.

Planning Same as MEDOR, except that we run ICP [130] for 5 iterations to align the predicted point cloud with the goal point cloud before computing the cost function for canonicalization task.

Simulation parameters	Value
Camera view	Top-down
Camera position	[0, 0.65 m, 0]
Field of view	90
Picker radius	0.01 m
Picker threshold	0.00625 m
dt	0.01 second
Damping	1
Dynamic friction	1.2
Particle friction	1
Stiffness (strech, bend, shear)	[1.2, 0.6, 1]
Mass	0.0003
Particle radius	0.005
Gravity	-9.8

Supplementary Table B.1: Hyper-parameters of softgym.

Trousers	Shirt	Dress	Skirt	Jumpsuit
2	1	2	12	2

Supplementary Table B.2: Order of rotation symmetry of different types of cloth. The order of rotation symmetry is the times the shape fits onto itself when rotating for 360 degrees. An order of 2 means that the shape remains unchanged when rotating for 180 or 360 degrees. We use it to construct goal sets for cloth canonicalization task.

	Trousers	Shirt	Dress	Skirt	Jumpsuit
No. of Meshes	1691	1111	2037	468	2279
Avg. No. of Vertices	7050 ± 1954	5308 ± 996	8030 ± 2159	4643 ± 1225	10686 ± 2572
Avg. No. of Vertices (downsampled)	278 ± 87	214 ± 59	291 ± 99	190 ± 80	215 ± 58
Rescaling Factor	0.42	0.36	0.29	0.28	0.28
Avg. X (m)	0.29 ± 0.03	0.32 ± 0.10	0.24 ± 0.07	0.20 ± 0.05	0.19 ± 0.08
Avg. Y (m)	0.12 ± 0.02	0.11 ± 0.02	0.15 ± 0.05	0.15 ± 0.05	0.09 ± 0.02
Avg. Z (m)	0.27 ± 0.08	0.19 ± 0.04	0.31 ± 0.06	0.18 ± 0.05	0.31 ± 0.06

Supplementary Table B.3: The statistics of the CLOTH3D dataset [6] after pre-processing.

	Task	Flattening			Canonicalization		
	Number of Pick-and-Place	1	2	3	1	2	3
Trousers	VSF [38]	0.09 ± 0.13	0.15 ± 0.07	0.14 ± 0.13	0.02 ± 0.02	0.03 ± 0.04	0.05 ± 0.05
	VCD [65]	0.34 ± 0.15	0.41 ± 0.14	0.53 ± 0.24	0.22 ± 0.14	0.26 ± 0.17	0.46 ± 0.16
	GarmentNets [19]	0.27 ± 0.15	0.37 ± 0.14	0.53 ± 0.16	0.14 ± 0.06	0.21 ± 0.09	0.33 ± 0.14
	MEDOR (no fine-tuning)	0.41 ± 0.14	0.57 ± 0.16	0.72 ± 0.08	0.48 ± 0.16	0.64 ± 0.10	0.77 ± 0.10
	MEDOR	$\mathbf{0.52 \pm 0.12}$	$\mathbf{0.69 \pm 0.12}$	$\mathbf{0.79 \pm 0.10}$	$\mathbf{0.59 \pm 0.13}$	$\mathbf{0.73 \pm 0.10}$	$\mathbf{0.77 \pm 0.07}$
Shirt	VSF [38]	0.12 ± 0.07	0.15 ± 0.08	0.20 ± 0.09	0.01 ± 0.02	0.01 ± 0.03	0.03 ± 0.04
	VCD [65]	0.39 ± 0.14	0.51 ± 0.23	0.70 ± 0.20	0.10 ± 0.20	0.14 ± 0.23	0.22 ± 0.21
	GarmentNets [19]	0.34 ± 0.21	0.47 ± 0.17	0.55 ± 0.22	0.08 ± 0.12	0.10 ± 0.14	0.11 ± 0.16
	MEDOR (no fine-tuning)	$\mathbf{0.59 \pm 0.17}$	$\mathbf{0.78 \pm 0.26}$	0.94 ± 0.22	0.46 ± 0.26	$\mathbf{0.60 \pm 0.12}$	$\mathbf{0.62 \pm 0.24}$
	MEDOR	0.59 ± 0.22	0.77 ± 0.19	$\mathbf{0.96 \pm 0.13}$	$\mathbf{0.53 \pm 0.26}$	0.56 ± 0.12	0.61 ± 0.09
Dress	VSF [38]	0.12 ± 0.07	0.15 ± 0.08	0.20 ± 0.09	0.02 ± 0.03	0.03 ± 0.03	0.04 ± 0.04
	VCD [65]	0.39 ± 0.14	0.51 ± 0.23	0.70 ± 0.20	0.18 ± 0.17	0.25 ± 0.16	0.34 ± 0.19
	GarmentNets [19]	0.38 ± 0.16	0.51 ± 0.22	0.57 ± 0.16	0.07 ± 0.10	0.13 ± 0.13	0.21 ± 0.19
	MEDOR (no fine-tuning)	0.43 ± 0.15	0.63 ± 0.19	0.69 ± 0.18	0.36 ± 0.17	0.53 ± 0.22	0.65 ± 0.15
	MEDOR	$\mathbf{0.50 \pm 0.14}$	$\mathbf{0.65 \pm 0.11}$	$\mathbf{0.80 \pm 0.17}$	$\mathbf{0.51 \pm 0.14}$	$\mathbf{0.60 \pm 0.08}$	$\mathbf{0.72 \pm 0.09}$
Skirt	VSF [38]	0.25 ± 0.13	0.32 ± 0.19	0.27 ± 0.24	0.06 ± 0.04	0.07 ± 0.05	0.10 ± 0.07
	VCD [65]	0.56 ± 0.13	0.67 ± 0.15	0.87 ± 0.12	0.19 ± 0.14	0.20 ± 0.14	0.21 ± 0.19
	GarmentNets [19]	0.46 ± 0.16	0.59 ± 0.13	0.70 ± 0.13	0.12 ± 0.10	0.20 ± 0.12	0.25 ± 0.16
	MEDOR (no fine-tuning)	0.56 ± 0.16	0.63 ± 0.19	0.73 ± 0.16	0.39 ± 0.14	0.41 ± 0.17	0.46 ± 0.21
	MEDOR	$\mathbf{0.58 \pm 0.12}$	$\mathbf{0.78 \pm 0.14}$	$\mathbf{0.91 \pm 0.13}$	$\mathbf{0.42 \pm 0.14}$	$\mathbf{0.47 \pm 0.18}$	$\mathbf{0.56 \pm 0.21}$
Jumpsuit	VSF [38]	0.12 ± 0.06	0.15 ± 0.07	0.19 ± 0.10	0.02 ± 0.03	0.03 ± 0.03	0.04 ± 0.05
	VCD [65]	0.36 ± 0.10	0.45 ± 0.15	0.64 ± 0.19	0.23 ± 0.09	0.19 ± 0.11	0.45 ± 0.19
	GarmentNets [19]	0.33 ± 0.12	0.41 ± 0.12	0.55 ± 0.14	0.13 ± 0.15	0.18 ± 0.14	0.33 ± 0.21
	MEDOR (no fine-tuning)	0.45 ± 0.17	0.65 ± 0.14	0.78 ± 0.19	$\mathbf{0.59 \pm 0.14}$	0.67 ± 0.09	0.76 ± 0.08
	MEDOR	$\mathbf{0.53 \pm 0.17}$	$\mathbf{0.73 \pm 0.17}$	$\mathbf{0.82 \pm 0.10}$	0.57 ± 0.14	$\mathbf{0.74 \pm 0.09}$	$\mathbf{0.81 \pm 0.05}$

Supplementary Table B.4: Normalized Improvement (NI) of cloth flattening and cloth canonicalization, for varying numbers of allowed pick and place actions.

Model parameter	Value
Canonicalization Network	
Backbone	HRNet-32
Dimension of output feature	489
3D CNN	
Backbone	3D Unet
Level	4
Feature maps	32
Dimension of output feature	128
Implicit Shape Completion Network	
Backbone	MLP
Number of hidden layers	3
Size of hidden layers	512
Implicit Shape Completion Network	
Backbone	MLP
Number of hidden layers	3
Size of hidden layers	512
Training parameters	**Value**
Optimizer	Adam
Learning rate	0.0001
Gaussian noise std	0.005
Random rotation	[-180, 180]

Supplementary Table B.5: Hyper-parameters of mesh reconstruction model

Model parameter	Value
Encoder(same for both node encoder and edge encoder)	
Number of hidden layers	3
Size of hidden layers	128
Processor	
Number of message passing steps	10
Number of hidden layers in each edge/node update MLP	3
Size of hidden layers	128
Decoder	
Number of hidden layers	3
Size of hidden layers	128
Training parameters	**Value**
Number of trajectories	5000
Learning rate	0.0001
Batch size	16
Training epoch	120
Optimizer	Adam
Beta1	0.9
Beta2	0.999
Weight decay	0
Others	**Value**
dt	0.05 second
Particle radius	0.005 m
Vertex clustering voxel size	0.025 m
Neighbor radius R	0.036 m

Supplementary Table B.6: Hyper-parameters of GNN dynamics model.

Appendix C

Appendix for Self-supervised Cloth reconstruction

C.1 Real-world Cloth Flattening

In order to further demonstrate the potential of our self-supervised mesh reconstruction method for robotic application, we deploy it in real world for a cloth flattening task.

C.1.1 Experiment Setup

The objective of the experiment is to flatten a crumpled Tshirt by using a 7-DoF Franka robot and pick-and-place action. The evaluation metric is the normalized improvements of coverage (0 if no changes, 1 if maximum coverage is reached). Since our goal is to evaluate whether the pseudo label sufficiently accurate to improve the performance of manipulation task, we use the same Tshirt for flattening as we collect the pseudo label dataset.

C.1.2 Model-based Cloth Manipulation System

After finetuning the mesh reconstruction model with pseudo label dataset, we integrate it with a learned graph dynamics model for planning. At each step, we first reconstruct the cloth with mesh reconstruction model. Then we sample 100 random pick-and-place actions and roll out with the dynamics model. We use cloth coverage as the reward function and execute the action the results in highest coverage.

C.1.3 Results

We test the model with and without finetuning for 6 trajectories separately, and calculate the average normalized improvement. In each trajectory contains 10 pick-and-place actions. It should be noted that flattened a double-layer Tshirt is a very challenging task. And we can observe a performance gain after finetuning with the pseudo label dataset, which shows that the quality is sufficiently accurate for improving downstream manipulation task.

Method	Tshirt
MEDOR [39] w/o finetuning	0.23
MEDOR [39] w finetuning	0.3

C.2 Additional Details

C.2.1 Simulation Calibration

Before we start to track to motion of cloth, we firstly calibrate the simulation by identifying the values of several critical physical parameters. Due to the simplified dynamics of simulation, one may not able to find a single set of parameters that allow the simulation to match real world in every possible transitions. Therefore, for each pick-and-place action, we

Parameters	Range
Stiffness	[0.2, 0.55, 0.9, 1.25, 1.6]
Dynamic Friction Coefficient	[0.5, 1.4, 2.3, 3.2, 4.1, 5]
Particle Friction Coefficient	[0.5, 1.4, 2.3, 3.2, 4.1, 5]

Supplementary Table C.1: Types and range of physical parameters that we optimize during simulation calibration phase.

search for the optimal system parameters that
best simulate the current action.

We use Nvidia Flex as our simulator, and we find clothes stiffness and friction to be the most parameters. During the simulation calibration, we directly roll out the dynamics model with actions $a_{1:T}$, without any bells and whistles. We run a grid search over all combinations of parameters (see Table. C.1). On a single Nvidia GTX 2080Ti, it takes around 70 seconds to run over the 125 combinations of parameters.

C.2.2 Test-time Optimization

Test-time Optimization (TTO) is an important component in our framework. It is applied twice in our action-conditioned tracking pipeline. *TTO1* is applied iteratively inside the simulation loop of tracking process. The main goal of TTO1 is to augment the dynamics model by computing a pseudo action that aligns the simulated result with the measurement. (2) Due to the inevitable gap between real world and simulation, it is possible that simulation cannot fully match the real world even with the help of pseudo action. For example, if the clothes in the simulation is thicker than the real world's, then the simulated mesh will always differ from the real mesh, otherwise the physics constraint will be violated. Therefore, after the inner simulation loop, we apply another test-time optimization, which we refer as *TTO2*.

Since the purpose of *TTO1* and *TTO2* are not exactly the same, we use different combinations of regularization losses. In the following section, we will first describe the definitions of all the regularization losses. Then, we will list the combinations of losses for *TTO1* and *TTO2* separately.

Rigidity Loss. As-Rigid-As-Possible (ARAP) [104] is a common assumption for modeling cloth-like shapes. In *TTO1*, the correction term Δx_{t+1}^{corr} can be viewed as motion that transforms the mesh to match the partial point cloud at next timestep. Based on the ARAP, we assume the clothes move as rigidly as possible. Since we only model the motion by a translation (Δx_{t+1}^{corr}), we obtain a simplified rigidity loss:

$$\mathcal{L}_{Rig} = \frac{1}{|E_t|} \sum_{i,j \in E_t} \left\| \Delta x_{t+1,i}^{corr} - \Delta x_{t+1,j}^{corr} \right\|_2^2 \tag{C.1}$$

Intuitively, this loss helps improve the consistency of motions of adjacent particles.

Collision Loss. This loss is the ensure the nearby vertices are at least r m away from each other. For each vertex i, we find k nearest neighbors $\mathcal{N}_i$ that are within radius r of itself.

$$\mathcal{L}_{Col} = -\frac{1}{K} \sum_{i=0}^{K} \sum_{j \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i s|} \left\| x_i - x_j \right\|_2^2 \tag{C.2}$$

Edge Loss. This loss is also to ensure the plausibility of mesh by constraining adjacent vertices not to be too far again from each other. First we compute the mean u_e and standard deviation σ_e of edge length in the original mesh. Then we constrain the edge length of optimized mesh by a margin loss.

$$\mathcal{L}_{Edge} = \frac{1}{|E_t|} \sum_{i,j \in E_t} \left\| \max(0, dis(x_i, x_j) - \sigma_e) \right\|_2^2 \tag{C.3}$$

where we use euclidean distance for dis.

Chamfer	Laplacian	Rigidity	Collision	Edge	L2
1	0.01	10	10	1	0.01

Supplementary Table C.2: The weight of different losses.

L2 regularization. We also add a L2 regularization on correction term $\Delta x^{corr}_{t+1,i}$, which helps avoid noisy motions and encourage simple solutions.

$$\mathcal{L}_{Rig} = \frac{1}{K}\sum_{i=0}^{K}\left\|\Delta x^{corr}_{t+1,i}\right\|_2^2 \tag{C.4}$$

C.2.3 Finetuning for MEDOR

MEDOR [39, 19] consists of 3 components, a canonicalization network that maps pixel from observation space to canonical space, a implicit shape completion network that predicts winding number field [43], and a warp field prediction network that predicts a per-vertex transformation from canonical pose to observation space. The model is finetuned in a two-stage process similar to training [39, 19].

In the first stage, we train the canonicalization network alone. It should be noted that at the beginning of the tracking procedure, we use a pretrained MEDOR model to reconstruct the flattened mesh. This can be seen as registrating the mesh to canonical space because we have the correspondence between observation space to canonical space. Then, by tracking the positions of vertices in the subsequent steps, we obtain the pseudo training label for the canonicalization network.

In the second stage, we train the shape completion network, and warp field prediction network with the reconstructed mesh in the canonical space and observation space separately. We use Adam [51] optimizer with cyclic learning rate [103] between $1e^{-5}$ and $1e^{-6}$. The model is trained for 1000 epochs in the first stage and 2000 epochs in the second training stage.

Bibliography

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 31

[2] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002. 33

[3] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5):698–700, 1987. 23

[4] Lucas Barcelos, Rafael Oliveira, Rafael Possas, Lionel Ott, and Fabio Ramos. Disco: double likelihood-free inference stochastic control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10969–10975. IEEE, 2020. 31

[5] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 7, 44, 64

[6] Hugo Bertiche, Meysam Madadi, and Sergio Escalera. Cloth3d: Clothed 3d humans. In *European Conference on Computer Vision*, pages 344–359. Springer, 2020. xvii, 21, 30, 66, 69, 73

[7] Bharat Lal Bhatnagar, Garvita Tiwari, Christian Theobalt, and Gerard Pons-Moll. Multi-garment net: Learning to dress 3d people from images. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5420–5430, 2019. 31

[8] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. *Advances in neural information processing systems*, 29, 2016. 31

[9] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017. 31

[10] Aljaz Bozic, Pablo Palafox, Michael Zollhöfer, Angela Dai, Justus Thies, and Matthias Nießner. Neural non-rigid tracking. *Advances in Neural Information Processing Systems*, 33:18727–18737, 2020. 30, 38

[11] Aljaz Bozic, Michael Zollhofer, Christian Theobalt, and Matthias Nießner. Deepdeform: Learning non-rigid rgb-d reconstruction with semi-supervised data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7002–7012, 2020. 30

[12] Aljaz Bozic, Pablo Palafox, Michael Zollhofer, Justus Thies, Angela Dai, and Matthias Nießner. Neural deformation graphs for globally-consistent non-rigid reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1450–1459, 2021. 30

[13] Rui Caseiro, Joao F Henriques, Pedro Martins, and Jorge Batista. Beyond the shortest path: Unsupervised domain adaptation by sampling subspaces along the spline flow. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3846–3854, 2015. 31

[14] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019. 31

[15] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020. 26

[16] Siwei Chen, Xiao Ma, Yunfan Lu, and David Hsu. Ab initio particle-based object manipulation. *arXiv preprint arXiv:2107.08865*, 2021. 17, 26

[17] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 30

[18] 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. 16, 30, 35

[19] Cheng Chi and Shuran Song. Garmentnets: Category-level pose estimation for garments via canonical space shape completion. *ICCV*, 2021. 1, 15, 16, 17, 18, 19, 22, 24, 25, 29, 30, 31, 35, 68, 69, 74, 80

[20] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012. 27

[21] Haili Chui and Anand Rangarajan. A feature registration framework using mixture models. In *Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis. MMBIA-2000 (Cat. No. PR00737)*, pages 190–197. IEEE, 2000. 30

[22] Haili Chui and Anand Rangarajan. A new algorithm for non-rigid point matching. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 2, pages 44–51. IEEE, 2000. 30

[23] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016. 69

[24] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019. 7

[25] Marco Cusumano-Towner, Arjun Singh, Stephen Miller, James F O'Brien, and Pieter Abbeel. Bringing clothing into desired configurations with limited perception. In *2011 IEEE International Conference on Robotics and Automation*, pages 3893–3900. IEEE, 2011. 4

[26] R Daněřek, Endri Dibra, Cengiz Öztireli, Remo Ziegler, and Markus Gross. Deepgarment: 3d garment shape estimation from a single image. In *Computer Graphics Forum*, volume 36, pages 269–280. Wiley Online Library, 2017. 16, 29, 30, 31

[27] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015. 30

[28] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018. URL https://arxiv.org/abs/1812.00568. 5

[29] Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 39–46, 2013. 30

[30] Aditya Ganapathi, Priya Sundaresan, Brijen Thananjeyan, Ashwin Balakrishna, Daniel Seita, Jennifer Grannen, Minho Hwang, Ryan Hoque, Joseph Gonzalez, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. Learning Dense Visual Correspondences in Simulation to Smooth and Fold Real Fabrics. In *ICRA*, 2021. 5, 22

[31] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016. 31

[32] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2066–2073. IEEE, 2012. 31

[33] Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *2011 international conference on computer vision*, pages 999–1006. IEEE, 2011. 31

[34] Kaiwen Guo, Feng Xu, Tao Yu, Xiaoyang Liu, Qionghai Dai, and Yebin Liu. Real-time geometry, albedo, and motion reconstruction using a single rgb-d camera. *ACM Transactions on Graphics (ToG)*, 36(4):1, 2017. 30

[35] 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. 17, 30, 33

[36] Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B

Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018. 64

[37] Fangzhou Hong, Liang Pan, Zhongang Cai, and Ziwei Liu. Garment4d: Garment reconstruction from point cloud sequences. *Advances in Neural Information Processing Systems*, 34, 2021. 16

[38] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation. In *Robotics: Science and Systems (RSS)*, 2020. URL `https://arxiv.org/abs/2003.09044`. 3, 5, 10, 15, 17, 18, 24, 25, 52, 53, 74

[39] Zixuan Huang, Xingyu Lin, and David Held. Mesh-based dynamics model with occlusion reasoning for cloth manipulation. In *Robotics: Science and Systems (RSS)*, 2022. 29, 30, 31, 35, 37, 78, 80

[40] Martin Hwasser, Danica Kragic, and Rika Antonova. Variational auto-regularized alignment for sim-to-real control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2732–2738. IEEE, 2020. 31

[41] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017. 30

[42] Matthias Innmann, Michael Zollhöfer, Matthias Nießner, Christian Theobalt, and Marc Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *European Conference on Computer Vision*, pages 362–379. Springer, 2016. 30, 38

[43] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, 32(4): 1–12, 2013. 18, 69, 80

[44] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016. 10

[45] Stephen James, Andrew J Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Conference on Robot Learning*, pages 334–343. PMLR, 2017. 31

[46] Stephen James, Michael Bloesch, and Andrew J Davison. Task-embedded control networks for few-shot imitation learning. In *Conference on robot learning*, pages 783–795. PMLR, 2018. 31

[47] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019. 31

[48] Boyi Jiang, Juyong Zhang, Yang Hong, Jinhao Luo, Ligang Liu, and Hujun Bao. Bcnet: Learning body and cloth shape from a single image. In *European Conference on Computer*

Vision, pages 18–35. Springer, 2020. 16, 29, 30, 31

[49] P Jiménez. Visual grasp point localization, classification and state recognition in robotic manipulation of cloth: An overview. *Rob. Auton. Syst.*, 92:107–125, June 2017. URL `https://linkinghub.elsevier.com/retrieve/pii/S0921889016303517`. 16, 30

[50] Pablo Jiménez and Carme Torras. Perception of cloth in assistive robotic manipulation tasks. In *Natural Computing*, pages 1–23. Springer, 2020. 3, 4

[51] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 20, 34, 46, 80

[52] Yasuyo Kita and Nobuyuki Kita. A model-driven method of estimating the state of clothes for manipulating it. In *Sixth IEEE Workshop on Applications of Computer Vision, 2002.(WACV 2002). Proceedings.*, pages 63–69. IEEE, 2002. 4

[53] Yasuyo Kita, Fuminori Saito, and Nobuyuki Kita. A deformable model driven visual method for handling clothes. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3889–3895. IEEE, 2004. 4

[54] Yasuyo Kita, Toshio Ueshiba, Ee Sian Neo, and Nobuyuki Kita. Clothes state recognition using 3d observed data. In *2009 IEEE International Conference on Robotics and Automation*, pages 1220–1225. IEEE, 2009. 1, 4, 15, 16

[55] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126, 2010. 30

[56] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Sci Robot*, 5(47), October 2020. xiii, 9

[57] Robert Lee, Daniel Ward, Akansel Cosgun, Vibhavari Dasagi, Peter Corke, and Jurgen Leitner. Learning arbitrary-goal fabric folding with one hour of real robot experience. *Conference on Robot Learning (CoRL)*, 2020. 5

[58] Hao Li, Bart Adams, Leonidas J Guibas, and Mark Pauly. Robust single-view geometry and motion reconstruction. *ACM Transactions on Graphics (ToG)*, 28(5):1–10, 2009. 30

[59] Yinxiao Li, Chih-Fan Chen, and Peter K Allen. Recognition of deformable object category and pose. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 5558–5564. IEEE, 2014. 1, 15, 16, 29

[60] Yinxiao Li, Yan Wang, Michael Case, Shih-Fu Chang, and Peter K Allen. Real-time pose estimation of deformable objects using a volumetric approach. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1046–1052. IEEE, 2014. 1, 15, 16, 22, 29, 30, 31

[61] 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. 1, 15, 16, 22, 29

[62] Yinxiao Li, Yan Wang, Yonghao Yue, Danfei Xu, Michael Case, Shih-Fu Chang, Eitan

Grinspun, and Peter K Allen. Model-driven feedforward prediction for manipulation of deformable objects. *IEEE Trans. Autom. Sci. Eng.*, 15(4):1621–1638, October 2018. URL http://dx.doi.org/10.1109/TASE.2017.2766228. 1, 15, 16, 22

[63] 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. 3, 5

[64] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. SoftGym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 2020. 7, 8, 10, 21, 33, 51, 66, 69

[65] 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. 15, 16, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 30, 69, 70, 74

[66] Fei Liu, Zihan Li, Yunhai Han, Jingpei Lu, Florian Richter, and Michael C Yip. Real-to-sim registration of deformable soft tissue with position-based dynamics for surgical robot autonomy. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12328–12334. IEEE, 2021. 31

[67] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015. 31

[68] Kok-Lim Low and Tiow-Seng Tan. Model simplification using vertex-clustering. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 75–ff, 1997. 21, 70

[69] Henrik Hautop Lund and Orazio Miglino. From simulated to real robots. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 362–365. IEEE, 1996. 30

[70] 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. 8

[71] 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. URL https://ieeexplore.ieee.org/abstract/document/5509439. 4, 15, 16, 30

[72] Ioannis Mariolis, Georgia Peleka, Andreas Kargakos, and Sotiris Malassiotis. Pose and category recognition of highly deformable objects using deep learning. In *2015 International conference on advanced robotics (ICAR)*, pages 655–662. IEEE, 2015. 1, 4, 15, 16, 22, 29

[73] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. *Conference on Robot Learning (CoRL)*, 2018. URL https://arxiv.org/abs/1806.07851. 3, 5, 17, 30, 31

[74] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020. 31

[75] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 30

[76] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020. 17

[77] Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Parametrized shape models for clothing. In *2011 IEEE International Conference on Robotics and Automation*, pages 4861–4868. IEEE, 2011. 4, 16, 30

[78] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. 8

[79] Fabio Muratore, Theo Gruner, Florian Wiese, Boris Belousov, Michael Gienger, and Jan Peters. Neural posterior domain randomization. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1532–1542. PMLR, 08–11 Nov 2022. URL `https://proceedings.mlr.press/v164/muratore22a.html`. 31

[80] Andriy Myronenko, Xubo Song, and Miguel Carreira-Perpinan. Non-rigid point set registration: Coherent point drift. *Advances in neural information processing systems*, 19, 2006. 30

[81] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2146–2153. IEEE, 2017. 17

[82] Rahul Narain, Armin Samii, and James F O'brien. Adaptive anisotropic remeshing for cloth simulation. *ACM transactions on graphics (TOG)*, 31(6):1–10, 2012. 7

[83] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015. 30

[84] Eiichi Ono, N Kits, and Shigeyuki Sakane. Unfolding folded fabric using outline information with vision and touch sensors. *Journal of Robotics and Mechatronics*, 10:235–243, 1998. 16, 30

[85] Fumiaki Osawa, Hiroaki Seki, and Yoshitsugu Kamiya. Unfolding of massive laundry and classification types by dual manipulator. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 11(5):457–463, 2007. 4

[86] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 30

[87] Chaitanya Patel, Zhouyingcheng Liao, and Gerard Pons-Moll. Tailornet: Predicting clothing

in 3d as a function of human pose, shape and garment style. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7365–7375, 2020. 16, 30, 31

[88] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. 3, 16, 21

[89] Gerard Pons-Moll, Sergi Pujades, Sonny Hu, and Michael J Black. Clothcap: Seamless 4d clothing capture and retargeting. *ACM Transactions on Graphics (ToG)*, 36(4):1–15, 2017. 16

[90] Jhony Kaesemodel Pontes, James Hays, and Simon Lucey. Scene flow from point clouds with or without learning. In *2020 International Conference on 3D Vision (3DV)*, pages 261–270. IEEE, 2020. 17

[91] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 18, 19, 24, 26

[92] Jianing Qian, Thomas Weng, Luxin Zhang, Brian Okorn, and David Held. Cloth region segmentation for robust grasp selection. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9553–9560. IEEE, 2020. 16, 30

[93] Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. Learning rgb-d descriptors of garment parts for informed robot grasping. *Engineering Applications of Artificial Intelligence*, 35:246–258, 2014. 16

[94] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019. 31

[95] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016. 31

[96] Shunsuke Saito, Jinlong Yang, Qianli Ma, and Michael J Black. Scanimate: Weakly supervised learning of skinned clothed avatar networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2886–2897, 2021. 16, 29, 30, 31

[97] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020. 3, 5, 7, 16, 21, 44, 64, 69

[98] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80, 2008. 21

[99] 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. 30, 33

[100] Daniel Seita, Aditya Ganapathi, Ryan Hoque, Minho Hwang, Edward Cen, Ajay Kumar

Tanwani, Ashwin Balakrishna, Brijen Thananjeyan, Jeffrey Ichnowski, Nawid Jamali, Katsu Yamane, Soshi Iba, John Canny, and Ken Goldberg. Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. URL `https://arxiv.org/abs/1910.04854`. 5, 17, 30

[101] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017. 31

[102] Miroslava Slavcheva, Maximilian Baust, and Slobodan Ilic. Sobolevfusion: 3d reconstruction of scenes undergoing free non-rigid motion. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2646–2655, 2018. 30

[103] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017. 80

[104] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, pages 109–116, 2007. 79

[105] Jan Stria, Daniel Prusa, and Vaclav Hlavac. Polygonal models for clothing. In *Conference Towards Autonomous Robotic Systems*, pages 173–184. Springer, 2014. 4, 16, 30

[106] Jan Stria, Daniel Prusa, Vaclav Hlavac, Libor Wagner, Vladimir Petrik, Pavel Krsek, and Vladimir Smutny. Garment perception and its folding using a dual-arm robot. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 61–67. IEEE, 2014. 4

[107] Zhaoqi Su, Tao Yu, Yangang Wang, and Yebin Liu. Deepcloth: Neural garment representation for shape and style editing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 16, 29, 30, 31

[108] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 31

[109] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8934–8943, 2018. 30

[110] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5693–5703, 2019. 19, 26, 68

[111] Li Sun, Gerarado Aragon-Camarasa, Paul Cockshott, Simon Rogers, and J Paul Siebert. A heuristic-based approach for flattening wrinkled clothes. In *Conference Towards Autonomous Robotic Systems*, pages 148–160. Springer, 2013. 4, 15, 16, 30

[112] Priya Sundaresan, Rika Antonova, and Jeannette Bohg. Diffcloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects. *arXiv preprint arXiv:2204.03139*, 2022. 31

[113] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016. 31

[114] Te Tang and Masayoshi Tomizuka. Track deformable objects from point clouds with structure preserved registration. *The International Journal of Robotics Research*, page 0278364919841431, 2018. 16, 34

[115] 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. 30, 33, 34

[116] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020. 30

[117] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017. 31

[118] Michal Tölgyessy, Martin Dekan, L'uboš Chovanec, and Peter Hubinskỳ. Evaluation of the azure kinect and its comparison to kinect v1 and kinect v2. *Sensors*, 21(2):413, 2021. 62

[119] Dimitra Triantafyllou and Nikos A Aspragathos. A vision system for the unfolding of highly non-rigid objects on a table by one manipulator. In *International Conference on Intelligent Robotics and Applications*, pages 509–519. Springer, 2011. 4

[120] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2642–2651, 2019. 17, 18

[121] 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. 16, 30, 33

[122] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 169–178, 2020. 64

[123] Thomas Weng, Sujay Man Bajracharya, Yufei Wang, Khush Agrawal, and David Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *Conference on Robot Learning*, pages 192–202. PMLR, 2022. 17

[124] Bryan Willimon, Stan Birchfield, and Ian Walker. Model for unfolding laundry using interactive perception. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4871–4876. IEEE, 2011. URL `https://ieeexplore.ieee.org/document/6095066`. 4, 16, 30

[125] Wenxuan Wu, Zhi Yuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. Pointpwc-net: Cost volume on point clouds for (self-) supervised scene flow estimation. In *European conference on computer vision*, pages 88–107. Springer, 2020. 30

[126] Wilson Wu, Yilin adn Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations. *Robotics Science and Systems*

(RSS), 2020. URL https://arxiv.org/abs/1910.13439. 3, 5, 11, 17, 18, 30, 52, 53

[127] Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. In *Conference on Robot Learning (CoRL)*, 2020. 3, 5, 11, 15, 17, 52, 53

[128] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 17

[129] Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *arXiv preprint arXiv:2011.02398*, 2020. 11

[130] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994. 70, 72

[131] Heming Zhu, Yu Cao, Hang Jin, Weikai Chen, Dong Du, Zhangye Wang, Shuguang Cui, and Xiaoguang Han. Deep fashion3d: A dataset and benchmark for 3d garment reconstruction from single images. In *European Conference on Computer Vision*, pages 512–530. Springer, 2020. 31

[132] Michael Zollhöfer, Matthias Nießner, Shahram Izadi, Christoph Rehmann, Christopher Zach, Matthew Fisher, Chenglei Wu, Andrew Fitzgibbon, Charles Loop, Christian Theobalt, et al. Real-time non-rigid reconstruction using an rgb-d camera. *ACM Transactions on Graphics (ToG)*, 33(4):1–12, 2014. 30