Learning Visible Connectivity Dynamics for Cloth Smoothing

Xingyu Lin*, Yufei Wang*, David Held
Robotics Institute, Carnegie Mellon University
Email: {xlin3, yufeiw2, dheld}@andrew.cmu.edu

***Abstract*—Robotic manipulation of cloth remains challenging for robotics due to the complex dynamics of the cloth, lack of a low-dimensional state representation, and self-occlusions. In contrast to previous model-based approaches that learn a pixel-based dynamics model or a compressed latent vector dynamics, 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. Compared to previous learning-based approaches, our model poses strong inductive bias with its particle based representation for learning the underlying cloth physics; it is invariant to visual features; and the predictions can be more easily visualized. We show that our method greatly outperforms previous state-of-the-art model-based and model-free reinforcement learning methods in simulation. Furthermore, we demonstrate zero-shot sim-to-real transfer where we deploy the model trained in simulation on a Franka arm and show that the model can successfully smooth different types of cloth from crumpled configurations. Videos can be found in the supplement and more videos are on our anonymous project website.**[1]

I. 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, deformable objects such as clothing cannot be easily described by low-dimensional state representations when placed in arbitrary configurations. Self-occlusions make state estimation especially difficult, such as when the cloth is in a crumpled configuration.

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 what is the appropriate state representation for the dynamics model. One possible state representation is to use a mesh model of the entire cloth [7]. However, fitting a full mesh model to an arbitrary crumpled cloth configuration is fairly difficult. Recent work have approached fabric manipulation by compressing the cloth representation into a fixed-size latent vector [32, 31, 17, 24]. Such compressed vector representations lose much of the spatial information of the cloth. Another previous approach is to learn a visual dynamics model in pixel space [6]. However, a pixel-based dynamics model is somewhat disconnected from the real physics of the cloth dynamics.

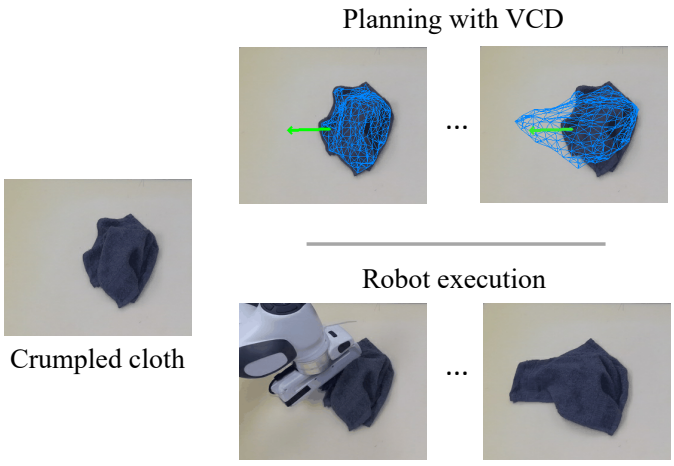


Fig. 1: We represent the cloth by inferring a visible connectivity graph from the partial point cloud. We then learn a dynamics model using this representation for planning to manipulate the cloth. The learned dynamics model is trained to be robust to the partial observability of the cloth as well as errors in the visible connectivity graph.

In contrast to a pixel-based dynamics model, particle-based models have recently been shown to be able to learn dynamics for fluid and plastics [12, 23, 22]. 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 macro-level by particles. Second, particle-based models are invariant to visual features such as object colors or patterns. Last, such models are interpretable, since a person can visualize the predicted movement of the object particles. As such, in this paper we aim to learn a particle-based dynamics model for the 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 due to self-occlusions when it is in a crumpled configuration.

Our insight into this problem is that, rather than fitting a mesh model to the observation, we should learn the *visible connectivity dynamics (VCD):* a dynamics model based on

* Equal contribution, order by dice rolling.
[1]https://sites.google.com/view/vcd-cloth

the connectivity structure of the visible portion of the cloth. To do so, we first learn to estimate the *visible connectivity graph*: we estimate which points in the point cloud observation are connected in the underlying cloth mesh. Estimating the mesh connectivity of the observation is a simplification of the problem of fitting a single full mesh model of the entire cloth to the observation; however, it is significantly easier to learn, since we do not need to find a globally consistent explanation of the observation; to estimate the mesh connectivity of the observation, we only need to consider the local cloth structure.

The downside of inferring the visible connectivity graph is that it cannot be used directly in a physics simulator since it does not capture the occluded parts of the object. We overcome this limitation by learning a dynamics model directly on the inferred visible connectivity graph (see Figure 1). Unlike a physics simulator, the learned dynamics model can be trained to be robust to the partial observability of the cloth as well as errors in the visible connectivity graph.

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 planning with a visible connectivity dynamics model can be effective in smoothing the cloth and our method greatly outperforms state-of-the-art model-based and model-free reinforcement learning methods that use a fixed-size latent vector representation or learn a pixel-based visual dynamics model. Furthermore, we demonstrate zero-shot sim-to-real transfer where we deploy the model trained in simulation on a Franka arm and show that the learned model can be used to successfully smooth different types of cloth from crumpled configurations.

II. Related Work

There has been a long history of cloth manipulation in robotics research, and they can be grouped into the following three categories:

Vision-based Cloth Manipulation: Some papers on cloth manipulation assume that the cloth is already lying flat on the table [18, 25, 26]. 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 [3, 15, 28]. 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 [3, 21, 8, 9, 10, 16] or after throwing the cloth on the table [28]. By constraining the cloth to this configuration set, the task of perceiving the cloth or fitting a mesh model [7] 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 [30] or a wrinkle-detector [27]. More examples of such approaches are described in a recent review article [7]. 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 to cloth manipulation 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 [24] or a scripted sequence of actions based on cloth descriptors [5]. Another approach to policy learning is model-free reinforcement learning (RL), which has been applied to cloth manipulation [17, 31, 11]. 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 [31] 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 [4, 6]. 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 [32] 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 [12, 23], 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.

III. Method

An overview of our method, VCD (Visible Connectivity Dynamics), can be found in Figure 2. We represent the cloth using a Visible Connectivity Graph, in which we infer the collision and mesh edges between points of a partial point cloud. Next, we learn a dynamics model over this graph, and finally we use this dynamics model for planning robot actions. Details of each of these components are described below.

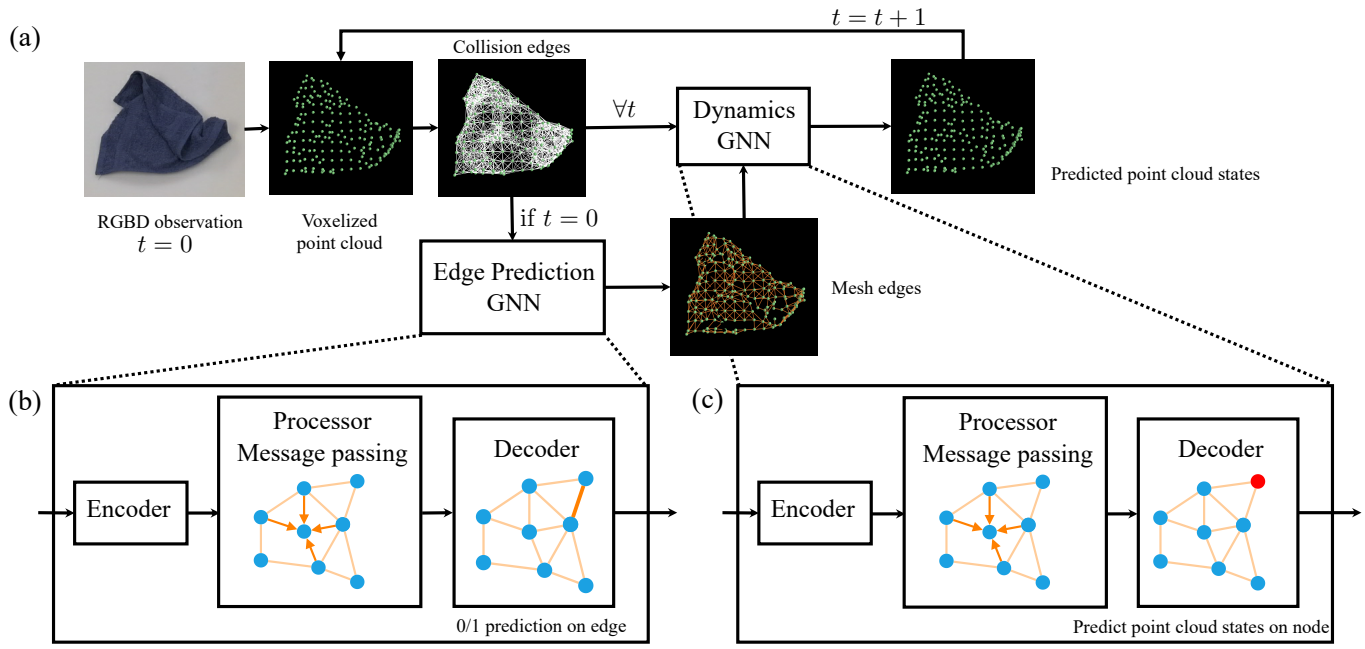


Fig. 2: (a) Overview of our visible connectivity dynamics model. It takes in the voxelized point cloud, construct the mesh and predict the dynamics for the point cloud. (b) Architecture for the edge prediction GNN which takes in the point cloud connected by the collision edges and predict for each collision edge whether it is an mesh edge. (c) Architecture for the dynamics GNN. Takes in the point cloud connected by both the collision edges and the mesh edges and predict the state of each point in the point cloud.

A. Graph Representation of Cloth Dynamics

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

$$E_t^C = \left\{ e_{ij} \middle| \; ||x_{i,t} - x_{j,t}||_2 < R \right\}, \tag{1}$$

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. 1.

B. 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 preprocess 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 evaluation. The voxel filter makes the particle density more consistent between the simulation and real world, which enables us to perform zero-shot transfer from simulation to the real world. It also speeds up computation by reducing the number of particles.

We create a graph node v_i for each point x_i in the voxelized point cloud P. The collision edges are then inferred by applying the criterion from Eqn. 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 to infer the mesh edges from the voxelized point cloud. Given the positions of the points in the voxelized point cloud P, we first construct a graph $\langle P, E^C \rangle$ with only the collision edges based on Eqn. 1. Based on our assumption that $E^M \subset E^C$,

we train a classifier to estimate whether each collision edge $e \in E^C$ is also a mesh edge. Our classifier takes the form of a graph neural network (GNN) [1]. We term this classifier the edge GNN, and denote it as G_{edge}. It takes as input the graph $\langle P, E^C \rangle$, and outputs a binary label for each edge $e \in E^C$, indicating whether such a collision edge is also a mesh edge. 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 the edge GNN architecture to the end of Sec. III-C and details on how we obtain the ground-truth mesh labels in Sec. III-E.

C. 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 [2, 13, 20], 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. III-B may not all be accurate. To handle these challenges, we learn a dynamics model based on the voxelized point cloud and its inferred visible connectivity (Sec. III-B). Formally, given the cloth graph $G_t = \langle V, E \rangle$, a dynamics GNN G_{dyn} predicts the particle states in the next time step, which includes both the positions and the 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 non-mesh collision edges. We take the network architecture in previous work [23] (referred to as GNS) for our dynamics GNN G_{dyn}, which contains an encoder, a processor, and a decoder, as described below:

Encoder: The encoder consists of two separate multi-layer perceptrons (MLP), denoted as ϕ_p, ϕ_e, that map the node and edge features, respectively, into latent embeddings. 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 III-D), and the distance to the table plane. The node encoder ϕ_p maps the node feature for node v_i into the node embedding h_i. 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 $||x_j - x_k||$, and a one-hot encoding of the edge type (mesh edge or non-mesh collision edge).

Processor: The processor consists of L stacked Graph Network (GN) blocks [1] that update the node and edge embeddings, with residual connections between blocks. 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 embeddings g^l and h^l respectively and outputs updated embeddings g^{l+1} and h^{l+1} (we denote g^0 and h^0 as the edge and node embeddings output by the encoder). For each GN block, first the edge update MLP updates the edge embedding; it takes as input the current edge embedding g_{jk}^l as well as the node embeddings h_j^l, h_k^l for the nodes that it connects:

$$g_{jk}^{l+1} = f_e^l(h_j^l, h_k^l, g_{jk}^l) + g_{jk}^l, \quad \forall e_{jk} \in E. \tag{2}$$

The node update MLP then updates the node embeddings; its input consists of the current node embedding h_i^l and the sum of the updated edge embeddings for the edges that connect to the node:

$$h_i^{l+1} = f_p^l(h_i^l, \sum_j g_{ji}^{l+1}) + h_i^l, \quad \forall i = 1, ..., N_p. \tag{3}$$

As shown in Equations (2) and (3), the edge and node updates both have residual connections between consecutive blocks.

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:

$$\ddot{x}_i = \psi(h_i^L) \tag{4}$$

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. III-E 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 $\langle P, E^C \rangle$, and aims to infer which collision edges are also mesh edges. The node feature is just the point position (without the point velocity or type). 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.

D. Planning with Pick-and-place Actions

We plan in a high-level, pick-and-place action space. For each action, denoted by two positions $a = \{x_{pick}, x_{place}\}$, the gripper grasps the cloth at x_{pick}, moves to x_{place}, and then drops the cloth. However, 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, while a pick-and-place action spans a larger time interval. As such, we first generate a sequence of waypoints $\Delta x_1, ..., \Delta x_H$ from the high-level action, where $x_{pick} + \sum_{i=1}^{H} \Delta x_i = x_{place}$. The high-level pick-and-place action can then be decomposed to H low-level actions that move the gripper by a small amount along the waypoints. We can then rollout our dynamics model with these low-level delta movements, each spans only a small time interval. 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. Then, when considering the effect of the t^{th} low-level action, we modify the graph by directly setting the picked point u's position $x_{u,t}$ and velocity $\dot{x}_{u,t}$ to be

$$x_{u,t} = x_{pick} + \sum_{i=1}^{t} \Delta x_i \qquad \dot{x}_{u,t} = \Delta x_i / \Delta t, \tag{5}$$

where Δt is the time for one low-level action step. 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 (Eqn. 5 is not used).

Our goal is to smooth a piece of cloth from a crumpled configuration. To compute the reward function r, 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. Due to computational limitations, we greedily optimize this reward over the predicated 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 of the robot 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. 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. We summarize our full method in Algorithm 1.

Algorithm 1: 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 by Eqn. (1)
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 $\mathbf{x}_t$ by Eq (1)
    Move picked point according to gripper movement by Eq (5):
    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t, \;\; \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$
    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 Eq (5):
    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t, \;\; \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$
  end
  Compute reward r based on final point cloud predicted position $\mathbf{x}_H$
end
return *pick and place action with maximal reward*

E. Training in Simulation

The simulator we use for training is Nvidia Flex, a particle-based simulator with position-based dynamics [19, 14], wrapped in SoftGym [13]. 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 position and velocity for each point in the observed point cloud and the mesh edges among them. To address this issue, we perform bipartite graph matching to match each point in the voxelized point cloud to a simulated particle by minimizing the Euclidean distance between the matched pairs. Details about the matching can be found in the appendix. After we get the mapping from the points to the simulator particles, the ground-truth acceleration of each point is simply assigned to be the acceleration of its mapped particle, which is used for training the dynamics GNN.

We train the dynamics GNN with "teacher-forcing" [29]: suppose that we sample a transition (V_t, a'_t, V_{t+1}), where a'_t is a low-level action (see Sec. III-D). 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.

At test time, we do not have the ground-truth mesh edges; instead, the mesh edges are inferred by the edge GNN. Thus we first train the edge GNN, and then we train the dynamics GNN using edges predicted by the trained edge GNN. This ensures that the mesh edges used by the dynamics GNN have the same distribution at training and test time.

IV. Experiment

A. Experimental Setup

Simulation Setup As mentioned, we use the Nvidia Flex simulator, wrapped in SoftGym [13], 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 and another 40 initial configurations for testing. The initial configurations are generated by picking the cloth up and then dropping it on the table in simulation.

The performance metric we use for evaluation is the covered area of the cloth in the top-down view. We normalize this metric in the following way: suppose the covered area of the initial cloth configuration is s_0 and the maximal possible covered area of the cloth is s_{max}. If the final covered area achieved by the algorithm is s, the normalized performance metric is

$$\tilde{s} = \frac{s - s_0}{s_{max} - s_0}. \tag{6}$$

In other words, the normalized performance metric $\tilde{s} = 0$ if no action is taken and $\tilde{s} = 1$ if the cloth is fully flattened ($\tilde{s}$ can also be negative if actions are taken that make the cloth more crumpled).

We compare our proposed method VCD (Visible Connectivity Dynamics) with the following baselines which are state-of-the art methods for cloth smoothing:

1) VisuoSpatial Foresight (VSF) [6], which learns a visual dynamics model using RGBD data;
2) Contrastive forward model (CFM) [32], which learns a latent dynamics model via contrastive learning;
3) Maximal Value under Placing (MVP) [31], which uses model-free reinforcement learning with a specially designed action space.

For all the baselines, we try our best to adjust the SoftGym cloth environment to match the cloth environment used in the original papers. For VSF, we place the camera to be top-down and zoomed in so that the cloth covers the entire image when fully flattened. We also changed the color of the cloth to be bluish as in the original paper. We collect 7115 trajectories, each consisting of 15 pick-and-place actions for training the VSF model (same as in the VSF paper). For CFM, we also use a top-down camera and change the color of the cloth to be the same on both sides, following the suggestion of the authors (personal communication). We collect 8000 trajectories each consisting of 50 pick-and-place actions for training the contrastive forward model (same as in the CFM paper). For MVP, we collect 5000 trajectories each with 50 pick-and-place actions and report the performance of the best performing model during training. We trained each of the baselines for at least as many pick-and-place actions as they were trained in their original papers. For training our method, VCD, we collect 1500 trajectories, each consisting of 1 pick-and-place action decomposed into 100 low-level actions for training. Note that this is fewer pick-and-place actions than any of the baselines used for training. More details for the implementation of the baselines and our method are in the appendix. Each method is evaluated on 40 test configurations and we report the mean and standard deviation of the results.

For planning with VCD, 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. The unnormalized direction vector $p = (\Delta x, \Delta y, \Delta z)$ (y is the up axis) of the pick-and-place is uniformly sampled as follows: $\Delta x, \Delta z \in [-0.5, 0.5]$, and $\Delta y \in [0, 0.5]$. The vector is normalized and then the distance is separately sampled from $[0.05, 0.2]$. More details about the action sampling of VCD and the baselines are in the appendix.

Real World Setup After training in simulation, we evaluate our trained dynamics model in the real world with a Franka Emika Panda robot arm with a standard panda gripper. We obtain RGB and depth information from a side view Azure Kinect camera and crop the RGBD image into the size of [345, 400], 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 two pieces of cloth: One is made of soft polyester satin, with a size of 0.25 x 0.25 meters, denoted as Silk. The other is made of cotton, with a size of 0.3 x 0.3 meters, denoted as Cotton.

We use the same covered area as our reward function and the evaluation metric in Eqn. 6 for the real world experiments. For the maximum covered area of a cloth used in normalizing the performance, we use a human flattened reference image: starting from a crumpled configuration of the cloth, a human sends a pick-and-place command to the robot by clicking on a screen with the cloth image, and then the robot arm executes it with pinch grasping. The human sends a maximum of 10 robot commands for each cloth. The human flattened reference images can be seen on the left side of Figure 4.

For each cloth, we evaluate 10 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. We report the mean and standard deviation of the normalized performance. For each trajectory, we reset the cloth configuration using the following protocol: Each time, the arm picks a random pixel 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.

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 pixel location corresponding to the segmented cloth. We generate a random

direction $\theta \in [0, 2\pi]$ and distance $l \in [0.02, 0.1]$ meters. We only accept an action if both the pick and the place points are within the work space. We additionally apply a heuristics to filter out actions whose place points are overlapping with the cloth. This heuristic saves computation time without sacrificing in performance.

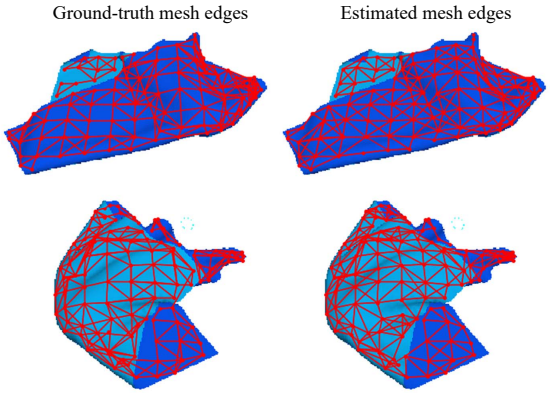


Fig. 3: The edge prediction result of our edge GNN. Red lines visualize the ground-truth (left) or inferred (right) mesh connections.

B. Simulation Results

For each method, we report the achieved normalized performance after different numbers of actions (counting high-level pick-and-place actions). The trajectory ends if the normalized performance is above 0.95 or if the maximal number of actions is achieved. For CFM and MVP, we additionally test them with 100 pick-and-place actions to compensate for the typically small action size of these methods.

The results are summarized in Table I. Under any given number of pick-and-place actions, VCD greatly outperforms all of the baselines. Due to their use of RGB data, the baselines do not incorporate the inductive bias of the cloth structure into their dynamics model or policy. In contrast, our method learns a particle-based dynamics model that incorporates this inductive bias and leads to better performance. Please see the appendix for examples of some planned pick-and-place action sequences of our method as well as visualizations of the predictions of our model.

We compare predictions of our edge prediction model with the ground-truth edges used for training the edge model in Figure 3. As shown, the edge GNN prediction reasonably matches the ground-truth, and thus well captures the cloth structure; it can also correctly disconnect the top layer from the bottom layer when the cloth is folded, e.g., the top left part of the first example and the bottom right part of the second example. Note our method uses only the point cloud as input and the color in this figure is only used for visualization. As our dynamics model is trained with our inferred edges, a few differences between the ground-truth mesh edges and our predicted edges is tolerable as the learned message passing in the dynamics model will adapt to the predicted edges.

C. Real-world Results

We also evaluate our method for smoothing in the real world. 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. VisuoSpatial Foresight (VSF) [6] evaluates their approach using an L2 cost over depth (without RGB) and they show that they achieve a 0% success rate. 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.

The quantitative results of our method can be found in Table II and a visualization of smoothing sequences is shown in Figure 4. We can see that, despite the drastic differences in visual appearances, as well as the different dynamics of the cotton and silk cloths, our model is able to smooth them. We also evaluate our model performance if we were 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 II. Videos of the complete trajectories and our model-prediction rollout can be found on our project website.

Furthermore, we show in Figure 5 our model's predicted score for each of the sampled actions during smoothing of the cotton cloth. Interestingly, though there is no explicit optimization for this, VCD favours picking corner or edge points and pulling outwards, which is a very natural and effective strategy for smoothing. This demonstrates the effectiveness of VCD for planning.

D. Ablation Study

We perform the following ablations to study the contribution of each component of our method. The first ablation aims to test whether a learned dynamics model gives better performance for our task than using a non-learned simulator. To evaluate this, we replace the learned graph neural network (GNN) dynamics model with the Flex simulator. In more detail, after we use the edge GNN to infer the mesh edges on the point cloud, we can create a cloth using the Flex simulator, where a particle is created at each location of the downsampled points, and a spring connection is added for each inferred mesh edge. The results is shown in Table III, row 2. We see that using the Flex simulator instead of the dynamics GNN has significantly worse performance. The main reason is that the cloth created from the partial point cloud with the inferred mesh edges deviates from the common cloth mesh structure used in Flex; thus, using the Flex simulator under this condition does not create realistic dynamics. On the other hand, the dynamics GNN is trained directly on the partial point cloud with the inferred mesh edges; therefore it can learn to compensate for the partial observability and for errors in the mesh edges when predicting the cloth dynamics. This ablation

Algorithm \ # of pick-and-place actions	5	10	20	50	100
VCD (Ours)	$\mathbf{0.620 \pm 0.252}$	$\mathbf{0.738 \pm 0.273}$	$\mathbf{0.879 \pm 0.202}$	$\mathbf{0.956 \pm 0.196}$	-
VSF [6]	0.337 ± 0.141	0.585 ± 0.194	0.750 ± 0.180	0.932 ± 0.102	-
CFM [32]	0.028 ± 0.070	0.053 ± 0.082	0.072 ± 0.122	0.122 ± 0.149	0.132 ± 0.177
MVP [31]	0.330 ± 0.296	0.341 ± 0.262	0.337 ± 0.296	0.421 ± 0.304	0.379 ± 0.268

TABLE I: Normalized performance of all methods in simulation, for varying numbers of allowed pick and place actions.

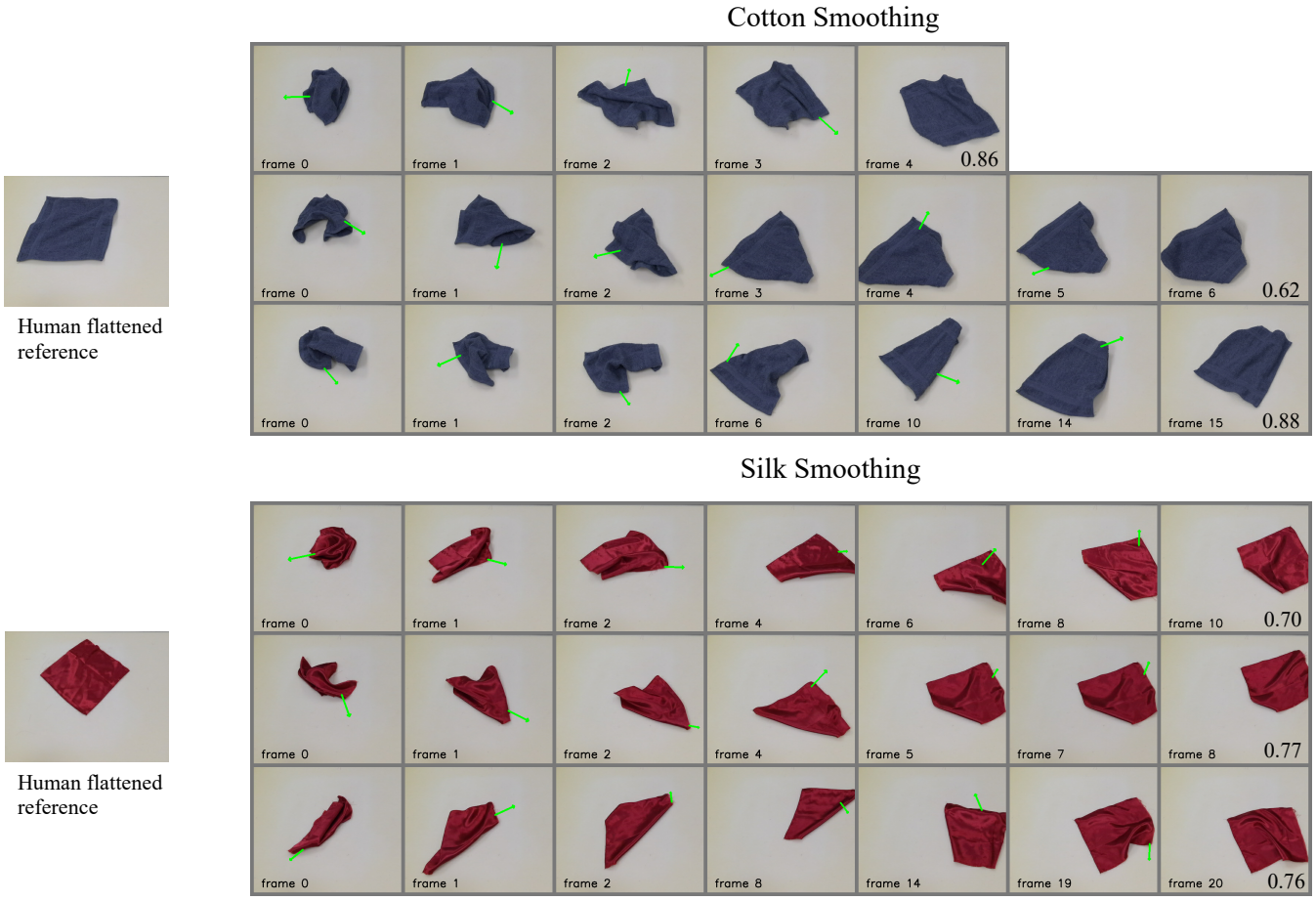


Fig. 4: Smoothing cotton (top) and silk (down) cloths with our method on a Franka robot. Each row shows one trajectory. Frame 0 shows the initial configuration of the cloth, and each frame after shows the observation after some number of pick-and-place actions, with the number labeled on the frame. The green arrow shows the 2D projection of the pick and place locations. Normalized performance of each trajectory is shown on the bottom right of the last frame of each row. A human flattened reference image shows the image of the cloth flattened by a human sending pick-and-place commands to the robot arm, which is used for normalizing the score.

validates the importance of using a dynamics GNN to learn the dynamics of the partially observable point cloud.

The next set of ablations aims to test whether using an edge GNN to infer the mesh edges as described in Section III-A is necessary for learning a good dynamics model. To test this, we carry out the following ablations. First, we train a dynamics GNN without using the edge GNN, where the edges are constructed solely based on distance by Eqn. (1). Since this ablation does not use an edge GNN, it cannot have two different edge types (collision vs mesh edges), as in our model. Thus at test time, all edges can either be kept fixed throughout the trajectory (similar to how we treat mesh edges in our model), or dynamically reconstructed using Eqn. (1) at each time step (similar to how we treat collision edges in our model). The results of these two ablations are shown in Table III, rows 3 and 4. As can be seen, the performance is much worse without the edge GNN. Indeed, the training in such a case is very unstable (see appendix for details).

Last, we perform another ablation where we train with both collision and mesh edges, but at test time, we do not use an

Material \ # of pick-and-place actions	5	10	20	Best
Cotton	0.504 ± 0.155	0.682 ± 0.253	0.797 ± 0.297	0.903 ± 0.153
Silk	0.412 ± 0.164	0.457 ± 0.236	0.456 ± 0.236	0.648 ± 0.169

TABLE II: Normalized performance of our method in the real world on two cloths of different materials.

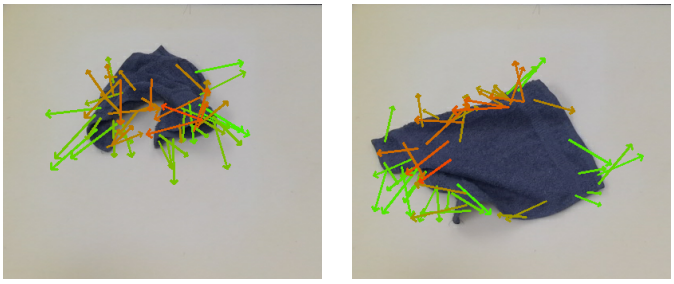


Fig. 5: Examples of 50 sampled actions used for planning. Each arrow goes from the 2D projection of the pick location to that of the place location. The action with the highest predicted reward is shown in green and the action with the lowest predicted reward is shown in red.

Algorithm	Normalized performance
Our full method	0.738 ± 0.273
Replace GNN with Flex	0.390 ± 0.239
Use only collision edges	0.317 ± 0.309
Use only mesh edges	0.227 ± 0.292
Remove edge GNN at test time	0.248 ± 0.314

TABLE III: Normalized performance of all ablations in simulation after 10 pick-and-place actions.

edge GNN to infer the edge type; instead we consider the edges that satisfy the criteria of Eqn. (1) in the first time step as the mesh edges. The result of this ablation is shown in Table III, row 5. The performance is again much worse and the model predictions are unstable (see figures in the appendix). All these ablations validate the importance of using an edge GNN to infer the mesh edges.

V. Conclusion

In this paper, we propose the visible connectivity dynamics (VCD) model, under a framework that infers a visibility connectivity graph from the partial point cloud, learns a particle-based dynamics model over the graph, and apply it to plan action sequences for the task of cloth smoothing. VCD has the advantage of posing strong inductive bias that fits the underlying cloth physics, being invariant to visual features, and being interpretable. We show that VCD greatly outperforms previous state-of-the-art methods for cloth smoothing, and achieves zero-shot sim-to-real transfer when deployed on a Franka arm for smoothing different types of cloth from crumpled configurations.

References

[1] 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.

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

[3] 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] 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] 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 *arXiv preprint arXiv:2003.12698*, 2021. URL https://arxiv.org/abs/2003.12698.

[6] 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.

[7] Pablo Jiménez and Carme Torras. Perception of cloth in assistive robotic manipulation tasks. pages 1–23. Springer, 2020.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[30] 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.

[31] 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.

[32] Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. 2020.

A Implementation Details

A.1 Visible Connectivity Dynamics (VCD)

Network architecture: The dynamics GNN and the edge GNN have the same architecture. Both the node and edge encoders are Multi-Layer Perceptrons (MLPs) of three hidden layers with 128 hidden neurons, and use ReLU as the activation function. The processor consists of 10 stacked GN blocks. The edge and node update models during the message passing are again MLPs with three hidden layers of 128 neurons, and ReLU as the activation function. The decoder has the same architecture as the node / edge encoder.

Training data: We collect 1000 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: $\Delta x, \Delta z \in [-0.5, 0.5]$, $\Delta y \in [0.1, 0.3]$. The direction vector is then normalized and the move distance is sampled uniformly from $[0.2, 0.4]$. 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.

During planning, we sample pick-and-place actions in a different way: Δy is sampled from $[0, 0.5]$ and the distance is sampled from $[0.05, 0.2]$. Therefore, after training the dynamics GNN on the above dataset, we additionally collect 500 trajectories with the pick-and-place action sampled from the distribution used in planning. We fine-tune the dynamics GNN model on this dataset to match the distribution used in planning.

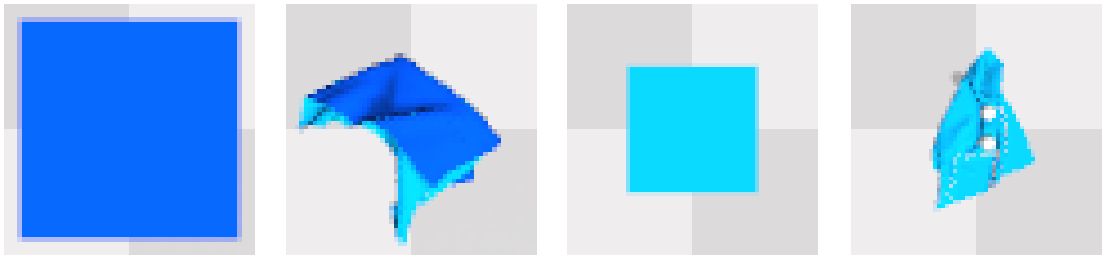
Training procedure: We use Adam [Kingma and Ba(2014)] with an initial learning rate of 0.0001 and reduce it by a factor of 0.8 if the training plateaus. We train with a batch size of 1. We first train the edge GNN, where the training usually converges in 1 or 2 days. After edge prediction GNN training is done, we train the dynamics GNN using mesh edges inferred by the trained edge GNN. The training of the dynamics GNN takes roughly 1 to 1.5 weeks to converge. After the training converges, we further fine-tune the dynamics GNN as described above for 1-2 days.

Action sampling during planning in simulation As described in the main text, we sample 500 pick-and-place actions, where the pick point is first uniformly sampled from a bounding box of the cloth and then projected to be on the cloth mask. For generating the bounding box, we first obtain the cloth mask from the simulator. We then obtain the minimal and maximal pixel coordinates u, v value of the cloth mask. The bounding box is the rectangle with corners $(min(u) - padding, min(v) - padding)$ and $(max(u) + padding, max(v) + padding)$, where padding is set to be 30 pixels for the 360×360 image size we use. We use rejection sampling to make sure the place point is within the image to keep the action within the depth camera view. The unnormalized direction vector $p = (\Delta x, \Delta y, \Delta z)$ (y is the up axis) of the pick-and-place is uniformly sampled as follows: $\Delta x, \Delta z \in [-0.5, 0.5]$, and $\Delta y \in [0, 0.5]$. The vector is normalized and then the distance is separately sampled from $[0.05, 0.2]$. We decompose the pick-and-place action into 50 low-level actions and wait for another 30 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 in performance.

Bipartite graph matching 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. 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.

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



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

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

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 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 1.

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.3 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 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.

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

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

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.

A.4 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.5 Simulated Cloth Details

As described in the main text, 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. FleX uses stretch, bend, and shear constraints to construct the cloth, and those parameters are set to be $0.8, 1, 0.9$, respectively.

B More Simulation Results

Supplementary Figure 2 shows four planned pick-and-place action sequences of VCD in simulation. As shown, VCD successfully plans actions that gradually smooths the cloth. We observe note that VCD favours picking edge / corner points and pulling outwards, which is an effective smoothing strategy, demonstrating the effectiveness of VCD for planning.

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

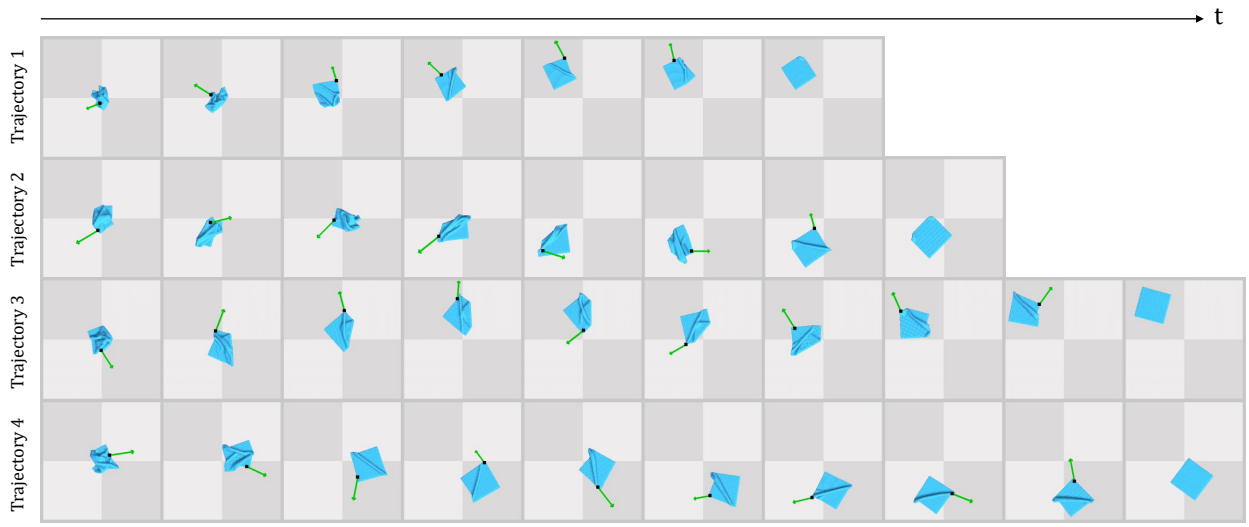
C Failures of Ablations

Here we show some visualizations of the failures of ablations.

The open-loop predictions of our dynamics model trained without mesh edges is shown in Supplementary Figure 4. As can be seen, the predicted particles diverge and behave like fluid without using mesh edges. This is expected, as mesh edges are the key for capturing the underlying cloth structure.

We also visualize the open-loop predictions of removing edge prediction GNN at test time, i.e., we treat edges constructed via Eqn. (1) in the first time step as mesh edges. The visualization is shown in Supplementary Figure 5. As can be seen, the model prediction just explodes. This is because the edges constructed via Eqn. (1) are much denser than the true mesh edges, and treating all of them as mesh edges causes the model to fail. Both ablations validate the necessity of using an edge prediction GNN to infer the mesh edges.

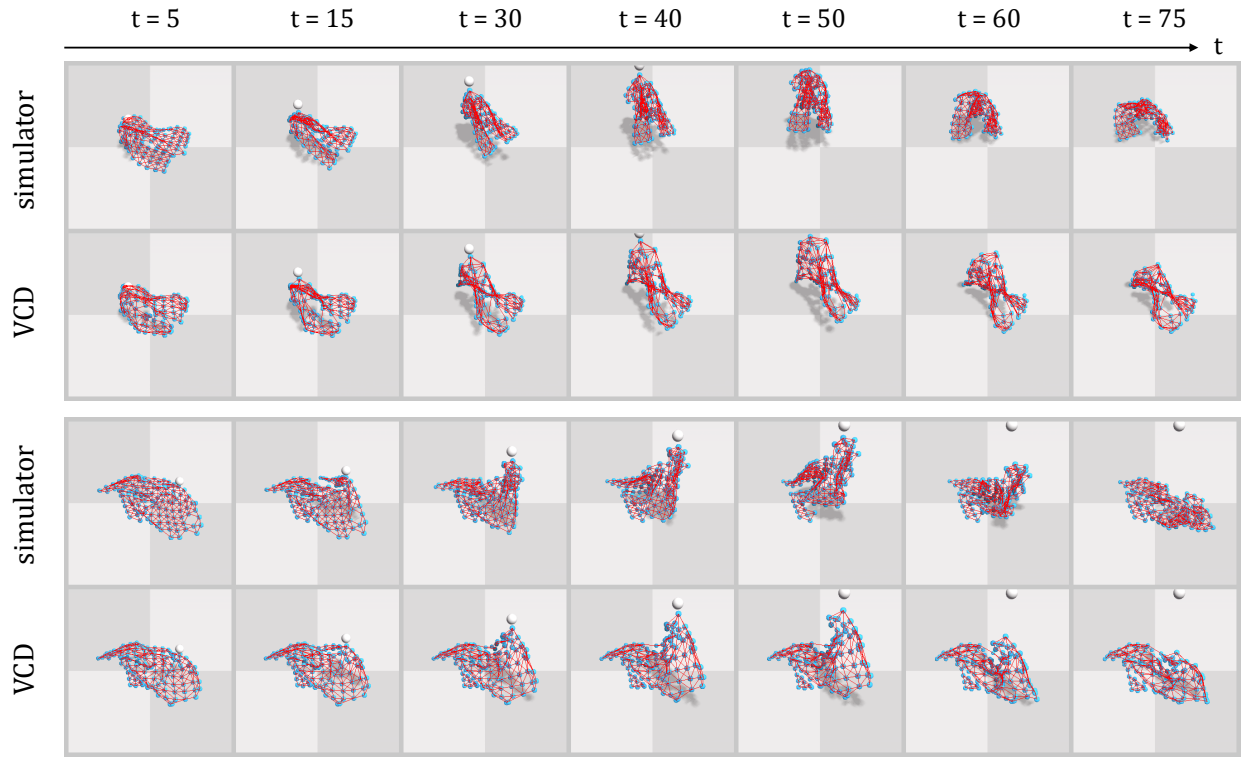
[3] https://github.com/wilson1yan/rlpyt



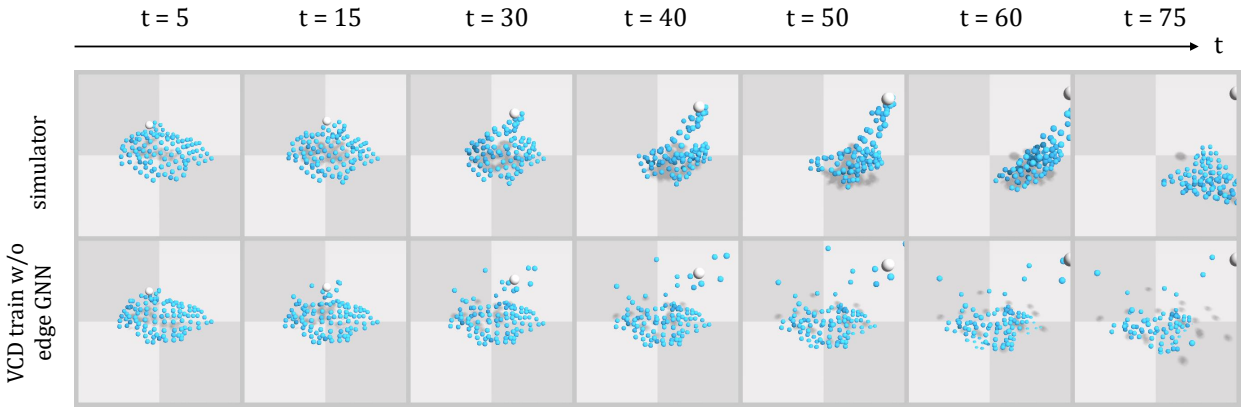
Supplementary Figure 2: Four example planned pick-and-place action sequences. All trajectories shown achieve a normalized performance above 0.98.

References

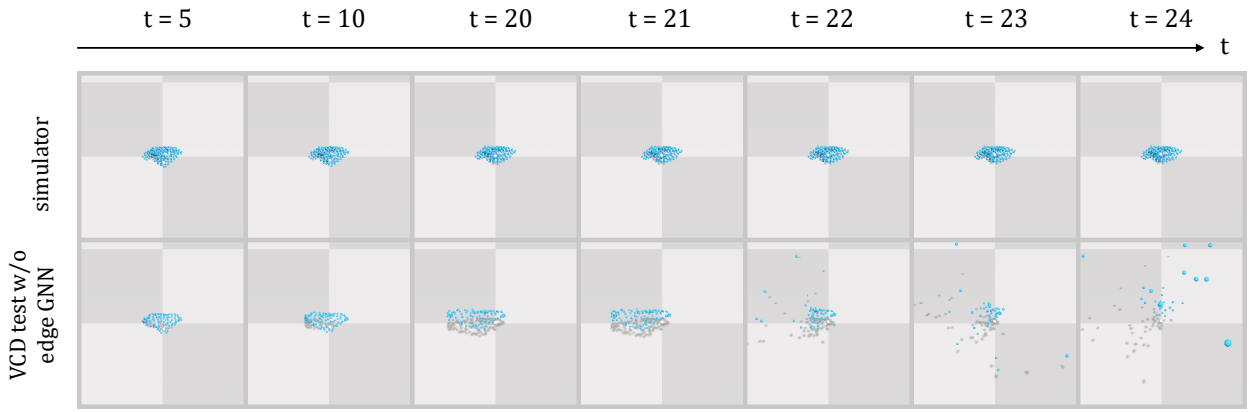
[Kingma and Ba(2014)] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.



Supplementary Figure 3: Two open-loop predictions of VCD. Blue points are particles/point cloud points and red lines are mesh edges. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by VCD, in which the mesh edges are inferred by the edge prediction GNN.



Supplementary Figure 4: Open-loop predictions of the dynamics model trained without the edge GNN.



Supplementary Figure 5: Open-loop predictions of the dynamics model, removing the edge GNN at test time.