Learning Closed-loop Dough Manipulation Using a Differentiable Reset Module

Carl Qi[1], Xingyu Lin[1], David Held[1]

***Abstract*—Deformable object manipulation has many applications such as cooking and laundry folding in our daily lives. Manipulating elastoplastic objects such as dough is particularly challenging because dough lacks a compact state representation and requires contact-rich interactions. We consider the task of flattening a piece of dough into a specific shape from RGB-D images. While the task is seemingly intuitive for humans, there exist local optima for common approaches such as naive trajectory optimization. We propose a novel trajectory optimizer that optimizes through a differentiable "reset" module, transforming a single-stage, fixed-initialization trajectory into a multistage, multi-initialization trajectory where all stages are optimized jointly. We then train a closed-loop policy on the demonstrations generated by our trajectory optimizer. Our policy receives partial point clouds as input, allowing ease of transfer from simulation to the real world. We show that our policy can perform real-world dough manipulation, flattening a ball of dough into a target shape. Supplementary videos can be found on our project website: https://sites.google.com/view/dough-manipulation.**

***Index Terms*—Deep Learning in Grasping and Manipulation; Perception-Action Coupling; Perception for Grasping and Manipulation**

I. INTRODUCTION

DEFORMABLE object manipulation allows autonomous agents to expand their applicability in our daily lives. Many household tasks such as folding laundry, cleaning rooms, and cooking food require a substantial amount of interaction with deformable objects, and recent works [6, 9, 13, 20, 35, 36] have shown great progress towards building a household robot. However, there are many challenges within deformable object manipulation that are yet to be solved. For one, deformable objects lack a compact state representation for manipulation. The state of rigid objects can often be captured by their 6D poses [3, 25, 33]. On the other hand, deformable objects have high degrees of freedom, making them hard to model. Many prior works rely on hand-designed representations for deformable object manipulation [5, 13, 23], which results in task-specific representations that lack generalizability. Recent works design more general data-driven methods to directly learn a policy from RGB-D images [22, 36] or a dynamics model [9, 18, 20, 37]. These works often directly interact with the deformable objects with the robot gripper and use pick-and-place actions as the primitive. However, elastoplastic object manipulation is particularly challenging, as they require rich contact for interacting with the object [28], often with an external tool. Simple action primitives are not sufficient in these cases. In this work, we tackle the challenges of elastoplastic object manipulation without the aid of feature engineering or action primitives.

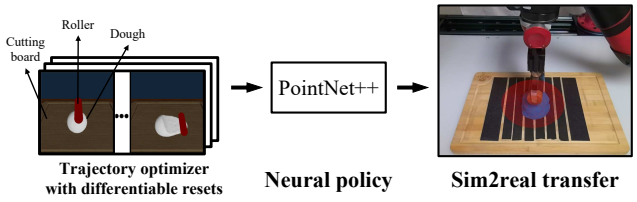


Fig. 1: We use a novel trajectory optimizer to generate demonstration data in simulation. We then use this demonstration data to train a closed-loop point cloud based policy using PointNet++ [27] as an encoder. Our policy can transfer to the real world without any fine-tuning.

Manipulation of elastoplastic objects like dough and clay has wide application for cooking and art-making, and there exists a rich list of literature on food manipulation [1, 5, 19, 34, 35]. Similar to other deformable objects, prior works have proposed action primitives for dough flattening [13, 23, 34]. They parameterize the action space by a few parameters such as the direction and length of a rolling trajectory, which limits the flexibility and efficiency of the obtained trajectory. In contrast, we learn closed-loop control policy by imitating a trajectory optimizer, allowing the robot to learn more complex motions without limiting it to one type of manipulation.

Enabled by recent progress on differentiable simulators [8, 11, 12], gradient-based trajectory optimization (GBTO) allows us to acquire many manipulation skills. However, many tasks require multistage manipulation and running GBTO from a fixed state initialization can result in locally optimal solutions. Solving these multi-stage tasks also require long-horizon reasoning. Prior work has proposed learning abstraction of the skills for planning over multi-stages [19]. In this work, we propose a simpler method to get around the local optima in GBTO by adding a differentiable reset module (DRM). Our key observation is that, we can often identify a few reset poses for the robot to be in between different stages of manipulation to help GBTO get around local optima. Consider the task of flattening a piece of dough into a circle. It is hard to perform the task with one simple rolling motion, and if one tries to roll multiple times, overly flattening the dough in early rolls will

Manuscript received: February 25, 2022; Revised: May 21, 2022; Accepted: June 16, 2022.

This paper was recommended for publication by Editor Markus Vincze upon evaluation of the Associate Editor and Reviewers' comments.

This material is based upon work supported by the National Science Foundation under Grant No. IIS-2046491, IIS-1849154 and LG Electronics. We thank Zhiao Huang for helping us with the simulator and thank Tim Angert and Sarthak Shetty for helping us with real world experiments.

Carl Qi, Xingyu Lin, and David Held are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. hanwenq@andrew.cmu.edu, xlin3@andrew.cmu.edu, dheld@andrew.cmu.edu.

Digital Object Identifier (DOI): see top of this page.

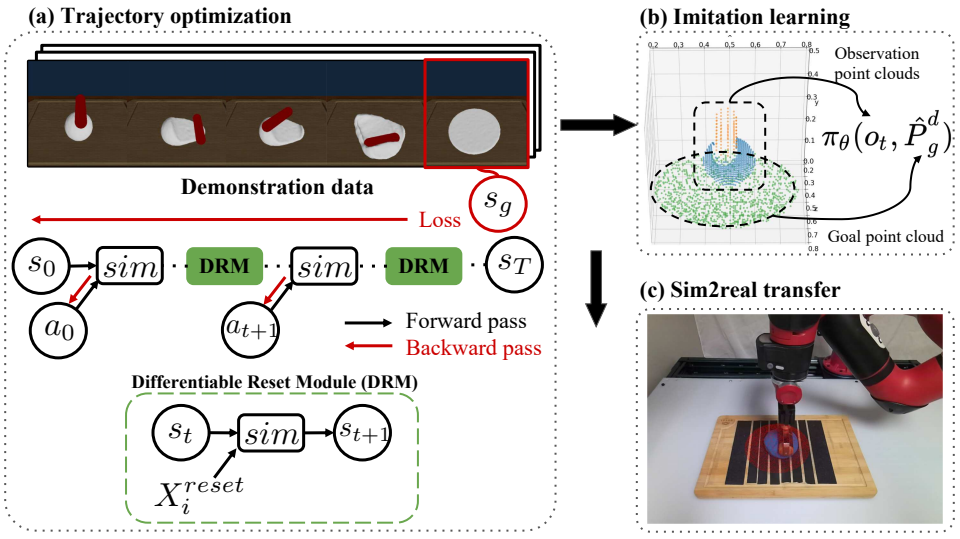


Fig. 2: An overview of our method. (a) Our trajectory optimizer implements a differentiable module that resets the tool to avoid local optima and allows gradient information to propagate through the entire trajectory. (b) We perform imitation learning on the demonstration data generated by the trajectory optimizer. Our policy takes segmented point cloud observations as input. (c) We use the learned policy to control a sawyer robot to perform closed-loop rolling in the real world.

cause later rolls to break the dough. As such, this seemingly easy task actually requires joint optimization of a multi-stage motion. To tackle these issues, DRM samples a few reset poses for the roller to be after each stage and differentiate through the reset motion to joinly optimize the multi-stage motion. Finding these reset poses is often easy and can significantly improve the performance of GBTO. We give another such example in cooking: when seasoning a pizza, the optimal motion requires tilting the shaker multiple times, each at a different location, instead of dumping all seasonings in one spot.

Finally, we use our trajectory optimizer in combination with a differentiable simulator to efficiently perform gradient-based optimization and generate high-quality demonstrations, as similarly done in [19]. We can then train a policy via imitation learning with partial point clouds as input. Our point cloud based policy is robust to occlusion in the scene and can effectively flatten a piece of dough in a smooth and closed-loop fashion. To summarize, in this paper:

- We propose a DRM in GBTO that optimizes a multistage trajectory end-to-end and avoids local optima.
- To the best of our knowledge, we introduce the first system for closed-loop dough manipulation in the real world. Our system operates directly from partial point cloud and does not require any feature engineering or action primitives.

II. RELATED WORK

A. Model-based approaches for dough manipulation

Many works use a model-based approach for dough manipulation. Tokumoto et al. [34] proposes an analytical dynamics model for flattening a piece of dough under a specific action primitive. The rolling primitive is a straight line motion of the roller with a fixed height and angle, where the height is predefined at test time, so the task becomes a simple 1D optimization over the rolling direction. Moreover, the analytical dynamics model is tailored to the rolling primitive, which makes it unusable for any other action parameterization or tasks. Recent works [2, 5, 13, 18, 23] also use action primitives for their tasks but learn a dynamics model in a data-driven fashion to improve generalization. However, many rely heavily on feature engineering for the state of the dough: Kim et al. [13] uses the set of distances between the boundary points and the center of the dough, Matl et al. [23] uses a bounding box for the dough, and Calinon et al. and Figueroa et al. [2, 5] fit an ellipse on the 2D view of the dough. These manually defined state spaces have limited representation power and cannot easily transfer to other tasks. Among them, DPI-Net [18] learns a particle dynamics model that functions over point clouds and does not require a manually specified state representation. However, DPI-Net requires full point cloud observation acquired by moving the robot arm out of the scene after each action, which significantly slows down their execution. Furthermore, DPI-Net uses a derivative free planner that also suffers from the local optima in our task. To compare with DPI-Net, we evaluate a CEM planner with ground-truth dynamics and reward in simulation. We also compare with a "heuristic" baseline that uses a similar action primitive described in [2, 13, 23, 34] in the real world to show the advantage of a more flexible action space. Last, Figueroa et al. [5] does not specify a fixed rolling primitive; instead, they automatically discover a set of dynamical systems (DS) from human demonstration, which they later use to perform rolling. Despite the advantages of DS in contact-rich interactions [26, 30], a good dynamical system for rolling is difficult to learn, and as a result, they require a significant number of rolls (between 12 and 71) to achieve the goal. In contrast to these works, we do not simplify the control by specifying a rolling primitive, nor do we manually define a compact state representation. These choices make the dough flattening problem significantly harder but meanwhile make our approach applicable to a general type of manipulation.

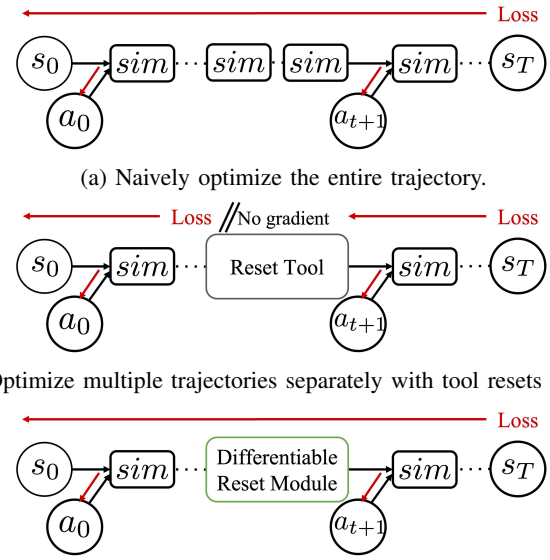
B. Model-free deformable object manipulation

As opposed to learning a dynamics model, another way is to learn a policy that directly outputs the actions. Wu et al. [36] and Matas et al. [22] learn a policy via Reinforcement Learning (RL). However, in our experiments with a Soft Actor Critic (SAC) [7] agent, we discover that RL with high dimensional input results in unmeaningful behavior. We include it as a baseline. Seita et al. [31] and Lin et al. [19] perform Imitation Learning (IL) with either an algorithmic supervisor or a trajectory optimizer that has access to privileged state information. Comparing to Seita et al. [31], which uses a algorithmic supervisor to a specific problem, our approach is more general. Similar to Lin et al. [19], we train our policy via Behavioral Cloning (BC) from the demonstrations generated by a trajectory optimizer, but our point cloud representation as policy input allows us to perform sim2real transfer with no fine-tuning.

C. Differentiable simulators for trajectory optimization

As deformable object manipulation becomes increasingly popular, the need for training data results in many high-quality simulators [8, 11, 12, 21]. We choose PlasticineLab [12], which uses Material Point Method [10] to model elastoplastic material. Built on top of the DiffTaichi system [11], it supports differentiable physics that allows us to perform gradient-based trajectory optimization.

There exists many prior works on non-convex trajectory optimization. Differential Flatness [24] exploits the fact that sometimes a dynamical system's state and control variables can be represented purely as a function of the system's output and its derivatives, and thus the state and control variables can be solved efficiently. Iterative LQR [17] performs a linear and quadratic approximation to the system's dynamics and cost function and iteratively refines the trajectory. However, with a large state space, finding a differentially flat system or approximating a nonlinear dynamics are both difficult. On the other hand, the differentiable simulator has the ability to compute the gradient information of the next state with respect to the previous state and action. Thus, using the backpropagation technique in neural network training, one can optimize a trajectory within the differentiable simulator with gradient descent [12]. However, as pointed out in Li et al. [16], this type of trajectory optimization suffers from local optima in slightly more complicated tasks, and they mitigate this issue via contact point discovery. Our approach is orthogonal to [16] and can be used in combination with contact point discovery to find better reset poses. Most relevant to our work is DiffSkill [19], which divides a long-horizon problem into simpler subproblems and leverages learned action primitives for long-horizon planning. However, DiffSkill uses more complex planning method and has only been shown to work in simulation with RGB-D images, which makes it hard to transfer to the real world. In contrast, our method uses simple reset poses to get around local optima and inputs partial point cloud for the policy, enabling easier sim2real transfer.



(a) Naively optimize the entire trajectory.

(b) Optimize multiple trajectories separately with tool resets in b/w.

(c) Optimize the entire trajectory with differentiable reset modules.

Fig. 3: Comparison of different trajectory optimization methods discussed in IV-A. (3a) suffers from local optima; Gradient in (3b) cannot propagate from one trajectory to another; (3c) allow us to optimize a multistage trajectory.

III. Problem Formulation

We formulate our dough manipulation task into a Partially Observable Markov Decision Process (POMDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, observation space Ω, deterministic transition dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and mapping from states to observations $O : \mathcal{S} \rightarrow \Omega$. In our simulation, we have access to the ground truth state information s, which includes the ground-truth dough particles $P^d \subseteq \mathcal{R}^3$ as well as the pose of the tool $X^t \in SE(3)$. The robot observation $o_t := \hat{P}_t^d \oplus P_t^t$ is the partial point cloud of the dough captured by a depth camera ($\hat{P}_t^d$) concatenated with the ground-truth point cloud of the tool (P_t^t), which we obtain from our prior knowledge of the tool shape and the end-effector position. The robot also observes the target dough point cloud $\hat{P}_g^d$ specified by a human. We assume that our transition function $\mathcal{T}$ is differentiable, which can be achieved using either a learned dynamics model [18] or a differentiable simulator, such as PlasticineLab [12].

IV. Method

Our pipeline consists of three main components: generating demonstrations via a novel trajectory optimizer, training a point-cloud based policy, and transferring the policy to the real world. For our trajectory optimizer, we leverage a reset module that avoids local optima and allows the gradient to back-propagate from one end of an episode to the other. We then use the demonstrations generated by the trajectory optimizer to train a policy that takes in partial point clouds as input. Last, we transfer the policy by mapping the action space from simulation to the real world. We discuss each of the components below in detail.

A. Demonstrations from non-local trajectory optimization

To generate expert demonstrations for rolling, we leverage a gradient-based trajectory optimization (GBTO) with the differentiable simulator. We write our trajectory optimization in Direct Shooting form [33] with all the constraints captured by the dynamics of the environment. Thus, given the initial state s_0, the objective is:

$$\min_{\substack{a_1,\ldots,a_T \\ \text{where } s_{t+1}=\mathcal{T}(s_t,a_t)}} L_{traj} = \sum_{t=0}^{T} L_t^{roll} + \lambda L_t^{contact} \tag{1}$$

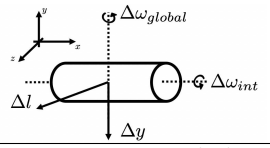
where L^{roll} measures the Earth Mover's Distance (D_{EMD}) [15] between the current and target dough shapes. $L^{contact}$ measures the distance from the tool to the closest particle to encourage the tool to approach the dough and λ weighs the different loss terms. This is the same objective used for dough manipulation in prior work DiffSkill [19].

With differentiable dynamics, we can back-propagate the gradient of the loss w.r.t the actions as follows:

$$\frac{\partial L_{traj}}{\partial a_t} = \sum_{t'=t+1}^{T} \frac{\partial \left(L_t^{roll} + \lambda L_t^{contact}\right)}{\partial s_{t'}} \frac{\partial s_{t'}}{\partial a_t} \tag{2}$$

where L_t^{roll} and $L_t^{contact}$ are differentiable w.r.t. s_t. A straightforward approach used by many others [12] is to use gradient descent to directly update the actions, as shown in Fig. 3a. However, gradient descent is a local optimization method, and the resulting action sequence will fall into a local optimum; specifically, the contact loss $L_t^{contact}$ will cause the tool to stay in close contact with the dough, which will cause the policy to perform only one rolling action (even if multiple rolling actions are needed to achieve the goal). Instead of using complex planning methods [19], we notice that a simpler approach is to divide an episode into multiple rolling trajectories with tool resets in between. Each trajectory has its own tool initialization and would be optimized separately w.r.t. L_{traj}. These reset poses break out of the local optima resulted from a fixed initialization of the tool. However, as mentioned earlier, flatting a piece of dough requires joint optimization of multiple rolls. As shown in Fig. 3b, simply dividing up an episode will break the gradient flow from one rolling trajectory to another, which will lead to a suboptimal solution.

Ideally, we would like to reset the tool in between different rolls and jointly optimize these rolls. To do so, we propose a novel trajectory optimization method that implements a differentiable reset module (DRM), as illustrated in Fig. 3c. The DRM can replace any action within a trajectory with a tool reset. We then can pick a set of timesteps $\{t_1^{reset}, ..., t_k^{reset}\}$ for DRM to move tool to a set of reset poses $\{X_1^{reset}, ..., X_k^{reset}\}$. Effectively, the timesteps separated by the DRMs are the different stages of rolling, and the DRMs provide multiple initializations of the tool in a trajectory. During trajectory optimization, we keep the rest of the gradients the same but do not back-propagate the gradient from the reset poses, i.e. we set $\frac{\partial L_{traj}}{\partial X_i^{reset}} = 0, \forall i \in \{1, ..., k\}$. For the dough flattening task, we find that it is easy to get around the local optima by specifying a few reset poses inspired by a human rolling dough. In general, our method is also useful to other tasks where it can be easy to specify a few intermediate reset poses. Using this approach, we transform a single-stage, fixed-initialization trajectory into a multistage, multi-initialization trajectory where all stages are optimized jointly. Our optimizer avoids local optima and produces synergistic behaviors between multiple rolls, as later shown in Section V. A detailed illustration of our trajectory optimizer is shown in Fig. 2.



Action	Description
Δy	displacement of roller height
Δl	displacement along rolling direction
$\Delta\omega_{int}$	internal rotation of the roller
$\Delta\omega_{global}$	rotation around the vertical axis

Fig. 4: Action space of the simulation environment.

B. Point-cloud policy learning via imitation learning

We deploy the above trajectory optimization method to generate demonstration trajectories, which we will use to train a policy from partial point clouds, as described in this section. Specifically, we use the trajectory optimization method described above to create a demonstration dataset $\mathcal{D} = \{o^{(i)}, a^{(i)}, \hat{P}_g^{d^{(i)}}\}_i$, where i indexes over trajectories. We use this data to train a goal-conditioned policy with hindsight relabeling via imitation learning [19]. Our policy $\hat{a}_t = \pi_\theta(o_t, \hat{P}_g^d)$ takes in the partial point cloud of the dough and ground-truth point cloud of the tool o_t, as well as the partial point cloud of the target dough shape $\hat{P}_g^d$, and outputs the action $\hat{a}_t$. Using a point cloud as input to our policy can help minimize the sim2real gap when transferring the policy to the real world. We train our policy using standard behavior cloning (BC) with the following loss:

$$L_{BC} = E_{(o,a,\hat{P}_g^d)\sim\mathcal{D}} \left[\|a - \pi_\theta(o, \hat{P}_g^d)\|^2 \right] \tag{3}$$

Details on our policy architecture are discussed in Appendix A-A.

C. Sim2real transfer

To transfer the policy to the real world, we parameterize the action space to abstract away the low-level controller. Specifically, given the policy output $\hat{a}_t$ in simulation, we use the environment's forward dynamics $\mathcal{T}$ to compute the target pose of the tool $\hat{X}_{t+1}$ and then apply a transformation on the coordinate system to obtain the real world target pose $\hat{X}_{t+1}^{real}$. Then we solve for the target robot joint angles with inverse kinematics and perform position control. We obtain the segmented dough point cloud by color thresholding and obtain the tool pose from the proprioceptive information of the robot. We then sample points uniformly on the surface of the tool to obtain the tool point cloud. More details on our real world setup in detail can be found in Section V-D.

D. Implementation details

Our trajectory optimizer runs Adam Optimizer [14] for 1000 steps with a learning rate of 0.005. We use a single reset module with a total of 100 rolling timesteps for our task because results in Section V-C suggests this combination is the most effective. For other tasks, one can adjust the number of resets and time horizon accordingly. We use our trajectory optimizer to generate 150 demonstration trajectories uniformly sampled over 125 initial and target configurations with different sizes and locations of the dough. We then train our policy using behavior cloning and add Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.01)$ to the partial point clouds during training to prevent overfitting. Our policy consists of a standard PointNet++ [27] encoder followed by a MLP. Details on the policy architecture can be found in Appendix A-A.

V. EXPERIMENTS

Our experiments aim to demonstrate the effectiveness of our pipeline for dough manipulation. Specifically, we show that our trajectory optimizer outperforms the existing alternatives and produces high-quality demonstrations in simulation. We then evaluate our policy that uses partial point clouds as input, in both simulation and the real world.

A. Experiments setup

Task. We conduct our simulation experiments in PlasticineLab [12], which provides a differentiable simulator that can simulate elastoplastic materials such as dough. Given a dough in a spherical shape, our task is to use a cylindrical roller to flatten the dough into a circular shape. We actuate the roller in simulation using a 4-dimensional continuous action space, as described in Fig. 4. The actions input to the simulator aim to specify the infinitesimal rigid transformation of the roller at every simulation step. We vary the initial and target dough locations, initial roller location, as well as the volume of the dough. An example of a trajectory in our simulation environment is shown at the top of Fig. 2.

Evaluation metric. We use the normalized final Earth Mover's Distance (EMD) as our performance metric in simulation, defined as:

$$\frac{D_{EMD}(P_0^d, P_g^d) - D_{EMD}(P_T^d, P_g^d)}{D_{EMD}(P_0^d, P_g^d)} \tag{4}$$

where P_0^d, P_T^d, P_g^d are the ground-truth dough point clouds at initialization, final timestep, and the target, respectively.

Baselines. We consider several baselines in simulation. First, we compare our policy with a model-free RL baseline trained with partial point clouds as input: Soft Actor Critic (SAC) [7]. Both the actor and the critic in SAC use the same inputs and the same encoder as our method. We train the SAC agent for 1 million timesteps and average the performance over 4 random seeds. Second, we compare our trajectory optimizer with the Cross Entropy Method (CEM). Both optimizers (our optimizer and the CEM baseline) operate on the ground-truth state. Details on the hyperparameters used in our experiments are shown in Appendix A-B.

B. Simulation experiments

We evaluate all methods on 10 held-out configurations. The results are shown in Fig. 5a. First, we see that our trajectory optimizer ("Diff-Reset (Ours)") outperforms the CEM baseline by a wide margin, highlighting the advantage of our reset module and the gradient information. We also see that our policy ("Diff-Reset-BC (Ours)"), trained using behavior cloning on the demonstration data (generated from trajectory optimization), outperforms the SAC agent trained using model-free RL.

To further demonstrate the generalization power of our policy, we show the performance of our policy on a larger set of held-out configurations in Fig. 6. The convex hull of the training data is shown in dotted grey lines, and the evaluation configurations are circles whose colors correspond to the normalized performance. As the result suggests, our policy can extrapolate to unseen configurations, despite a few failure cases where the target size is too small compared to the dough size.

C. Ablation studies

We first investigate the effects of rolling timesteps and the number of reset modules used in our task. For a fixed horizon $T = 100$, we vary the number of reset modules from 0 to 3 and divide the trajectory into equal numbers of timesteps for rolling, i.e. $t_i^{reset} = \lfloor \frac{100}{N_{reset}} \rfloor \cdot i$. Table I shows the normalized performance of using different number of resets in a trajectory. Most trajectories with resets outperform the one without any reset, highlighting the importance of multiple initializations, and two-stage rolling with one reset in between performs the best for our task. We Also consider the following ablations

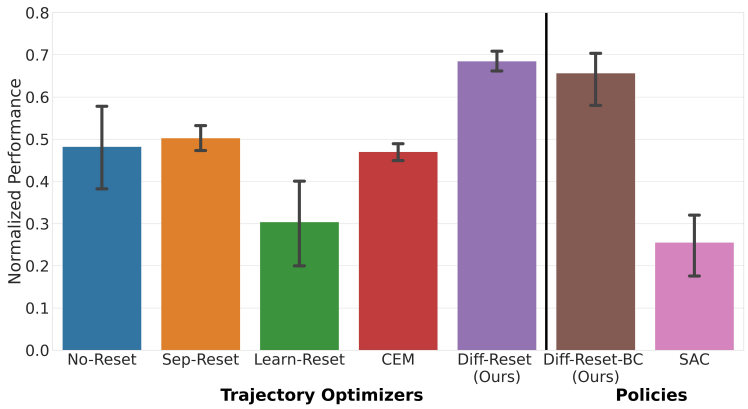
TABLE I: Effect of the number of resets in a fixed-horizon trajectory evaluated on 10 held-out configurations.

No-Reset	1-Reset	2-Reset	3-Reset
0.48 ± 0.15	$\mathbf{0.70 \pm 0.05}$	0.57 ± 0.05	0.42 ± 0.01

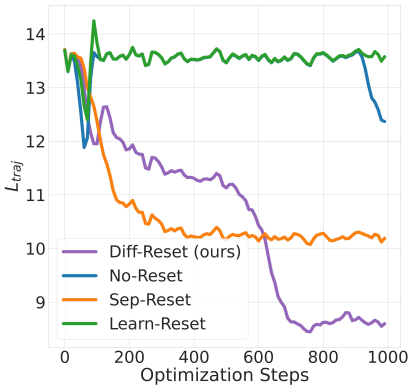
to our novel trajectory optimizer:

- No-Reset: optimize the whole trajectory without reset.
- Sep-Reset: optimize multiple rolling trajectories separately with a (non-differentiable) reset of tools in b/w.
- Learn-Reset: instead of using the reset module, try to learn the actions that moves the tool back via a reset loss $L_t^{reset} = D(X_t^{tool}, X_i^{reset})$.

We compare the performance of our method and the ablations over the held-out trajectories in Fig. 5a and include the loss curves of an example trajectory in Fig. 5b. As the result suggests, No-Reset (optimization without reset) gets stuck in local optima due to the complexity of our task. Sep-Reset (optimizing the rolling trajectories separately) results in the dough being broken in half, showing the importance of propagating gradient information from one trajectory to another. Learn-Reset tries to optimize for two loss functions, one for rolling and the other for resetting the roller. Since the two losses have conflicting gradient directions, one usually dominates the other, and the roller either ignores the reset action or doesn't performing rolling at all. Please see our website for visualizations of the baselines.



(a) Normalized performance on 10 held out configurations. The performance of the SAC agent is averaged over 4 random seeds.



(b) Loss L_{traj} of different trajectory optimizers over optimization steps on an example trajectory.

Fig. 5: Results of simulation experiments. (5a) shows the performances of our method comparing to all baselines and ablations. (5b) shows the trajectory loss of different trajectory optimizers over time.

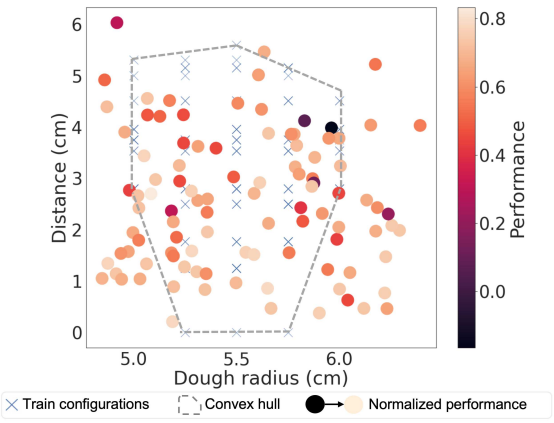


Fig. 6: Diff-Reset-BC Policy performance over different configurations of dough radius and target distance. Train configurations ('X') and the convex hull (grey dotted lines) for training data are overlayed. Our policy can effectively extrapolate to unseen configurations.

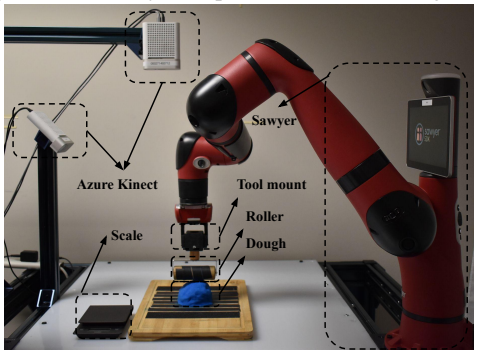


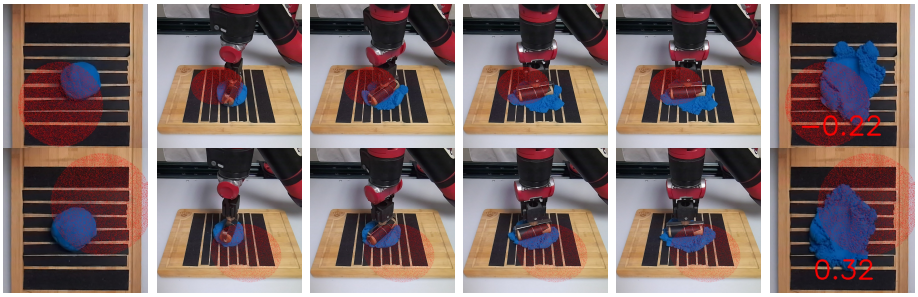
Fig. 7: Real world workspace setup.

D. Real world experiments

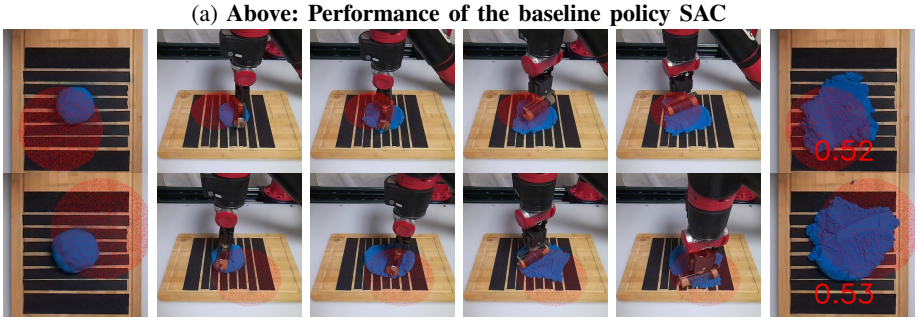
Our real world setup is shown in Fig. 7 which includes a Sawyer robot, two Azure Kinect cameras, a tool mount, a roller, and a cutting board. We use Kinetic Sand[1] as a proxy for "dough." The set of target dough configurations consists of 16 different locations, divided into 3 categories based on the distance between the goal location and the initial location. Since computing the volume of dough in the real world is difficult, we define each of the goal shape to be a flat disk with height zero on the top surface of the cutting board. This results in slight decrease in performance for all the methods because the thickness of the goal shape is not considered. The initial and target configurations of the dough are set in the following way: we reset the dough by manually rubbing it into a spherical shape and place it in the center of the cutting board. We then sample a goal from the predefined goal set. Last, we input to our policy observations from the side-view camera and execute the current policy until the episode completes. Using the above procedure, our closed-loop policy is able to achieve a control frequency of 2Hz and finish a trajectory in under 1 minute. To evaluate our method, we compute the normalized Chamfer Distance (CD) between the final point cloud and the goal (captured by the top-down camera), similar to Equation 4. We use Chamfer Distance (CD) rather than EMD for the real-world evaluation because EMD requires an equal number of points between the two point clouds, which does not occur for point clouds in the real world.

We consider three baselines and one ablation in our real world experiments. First, we compare to the SAC agent with the best-performing random seed. Second, we compare to a "Heuristic": we do a grid search over the rolling depth and rolling length from the initial point cloud observation and execute the same rolling primitive used in prior works [13, 34]. For a fair comparison, we repeat the rolling primitive twice with the tool reset used by our policy. Third, we compare with a human that observes the overlayed target in real time and is given unlimited time. We also compare to an ablation ("Open-loop") which is an open-loop policy trained on the same demonstration data as our method, but it takes in the initial observation o_0 and outputs the entire action sequence $\{\hat{a}_0, ..., \hat{a}_{T-1}\}$ in the episode as opposed the action in the next timestep. Table II shows the performance of all methods,

[1]https://kineticsand.com/



(a) **Above: Performance of the baseline policy SAC**



(b) **Above: Performance of Diff-Reset-BC (Ours)**

Fig. 8: Example rollouts of SAC (8a) and our policy (8b) with goal distance in $6 \sim 9$ cm. The top-down view of the initial observation and final observation is added to the start and end of the rollouts. The goal shape and normalized performance are overlayed in red.

	$0 \sim 3$ cm	$3 \sim 6$ cm	$6 \sim 9$ cm
Human	0.72 ± 0.02	0.63 ± 0.10	0.67 ± 0.02
SAC	0.52 ± 0.04	0.41 ± 0.06	0.11 ± 0.20
Heuristic	0.43 ± 0.03	0.27 ± 0.25	0.19 ± 0.10
Open-loop	0.59 ± 0.04	0.26 ± 0.41	-0.08 ± 0.45
Diff-Reset-BC (ours)	$\mathbf{0.62 \pm 0.07}$	$\mathbf{0.45 \pm 0.06}$	$\mathbf{0.51 \pm 0.04}$

TABLE II: Rolling performance over different target distances. Each entry is evaluated on 4 sampled targets.

	Small(240g)	Medium(360g)	Large(480g)
Human	0.64 ± 0.01	0.72 ± 0.02	0.71 ± 0.01
SAC	$\mathbf{0.44 \pm 0.12}$	0.34 ± 0.21	0.0 ± 0.36
Heuristic	0.33 ± 0.10	0.31 ± 0.13	0.21 ± 0.38
Open-loop	0.35 ± 0.35	0.26 ± 0.41	0.29 ± 0.36
Diff-Reset-BC (ours)	$\mathbf{0.44 \pm 0.09}$	$\mathbf{0.53 \pm 0.09}$	$\mathbf{0.51 \pm 0.17}$

TABLE III: Rolling performance over different dough sizes. Each entry is evaluated on 4 sampled targets.

grouped based on the distance between the initial and final dough configurations. Our policy Diff-Reset-BC outperforms the non-human baselines and ablation in all configurations, and it has the largest improvement over the baselines when the goal is far from the initial location of the dough. Our human baseline is the most dexterous and discovers novel tool uses such as pushing the overextended dough back into the goal region. This highlights the fact that dough manipulation requires a complex action space. Example rollouts for the SAC agent and Diff-Reset-BC are shown in Fig. 8. Last, we demonstrate the robustness of our policy by varying the size of the dough. We consider 3 different sizes quantified by the weight of the dough, and we scale the radius of the target shape based on the initial dough size. Table III shows the quantitative results. Diff-Reset-BC outperforms the non-human baselines for all dough sizes and retains a high performance across different sizes. Although the SAC baseline performs on par with our method for small dough, its performance degrades quickly as the dough size increases. Finally, we perform paired samples t-tests [32] to statistically compare the performances of our closed-loop policy and each baseline. After applying Bonferroni correction [29] with $\alpha = 0.013$, we find significant differences in performance between Diff-Reset-BC (M=0.51, SD=0.11) and:

- SAC (M=0.30, SD=0.28); $t(19) = 3.6$, $p = 1\text{e}{-3}$
- Heuristic (M=0.30, SD=0.21); $t(19) = 5.6$, $p = 2\text{e}{-5}$
- Human (M=0.66, SD=0.08); $t(19) = -4.3$, $p = 3\text{e}{-4}$

as well as marginally significant differences between ours and Open-loop (M=0.28,SD=0.39); $t(19) = 2.7$, $p = 0.013$.

VI. CONCLUSION

We introduce a system for closed-loop dough manipulation from high dimensional inputs. Our novel gradient-based trajectory optimizer leverages a differentiable reset module and can optimize an entire multistage trajectory to avoid local optima. We use the trajectory optimizer to generate high-quality demonstration data in a differentiable simulator, which we later use to train a policy via Behavioral Cloning. Our policy trained on partial point cloud is directly transferred to the real world without any fine-tuning. In both simulation and real world experiments, we demonstrate that our method outperforms other approaches and generalizes to different dough sizes and configurations.

REFERENCES

[1] M. Bollini, S. Tellex, T. Thompson, N. Roy, and D. Rus, "Interpreting and executing recipes with a cooking robot," Jan. 2013.

[2] S. Calinon, T. Alizadeh, and D. G. Caldwell, "On improving the extrapolation capability of task-parameterized movement models," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 610–616.

[3] T. Chen, J. Xu, and P. Agrawal, "A system for general in-hand object re-orientation," *Conference on Robot Learning*, 2021.

[4] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[5] N. Figueroa, A. L. Pais Ureche, and A. Billard, "Learning complex sequential tasks from demonstration: A pizza dough rolling case study," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, ser. HRI '16, Christchurch, New Zealand: IEEE Press, 2016, pp. 611–612.

[6] H. Ha and S. Song, "Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding," in *Conference on Robot Learning*, PMLR, 2022, pp. 24–33.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2017.

[8] E. Heiden, M. Macklin, Y. S. Narang, D. Fox, A. Garg, and F. Ramos, "DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting," in *Proceedings of Robotics: Science and Systems*, Virtual, Jul. 2021.

[9] R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg, "Visuospatial foresight for multi-step, multi-task fabric manipulation," Jul. 2020.

[10] Y. Hu, Y. Fang, Z. Ge, Z. Qu, Y. Zhu, A. Pradhana, and C. Jiang, "A moving least squares material point method with displacement discontinuity and two-way rigid body coupling," *ACM Transactions on Graphics*, vol. 37, no. 4, p. 150, 2018.

[11] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," *ICLR*, 2020.

[12] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. B. Tenenbaum, and C. Gan, "Plasticinelab: A soft-body manipulation benchmark with differentiable physics," in *International Conference on Learning Representations*, 2021.

[13] J.-T. Kim, F. Ruggiero, V. Lippiello, and B. Siciliano, "Planning framework for robotic pizza dough stretching with a rolling pin," in *Robot Dynamic Manipulation*, Springer, 2022, pp. 229–253.

[14] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[15] E. Levina and P. Bickel, "The earth mover's distance is the mallows distance: Some insights from statistics," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, 2001, 251–256 vol.2.

[16] S. Li, Z. Huang, T. Du, H. Su, J. B. Tenenbaum, and C. Gan, "Contact points discovery for soft-body manipulations with differentiable physics," in *International Conference on Learning Representations*, 2022.

[17] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.," vol. 1, Jan. 2004, pp. 222–229.

[18] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," in *ICLR*, 2019.

[19] X. Lin, Z. Huang, Y. Li, D. Held, J. B. Tenenbaum, and C. Gan, "Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools," in *International Conference on Learning Representations*, 2022.

[20] X. Lin, Y. Wang, Z. Huang, and D. Held, "Learning visible connectivity dynamics for cloth smoothing," in *Conference on Robot Learning*, 2021.

[21] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," in *Conference on Robot Learning*, 2020.

[22] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," *ArXiv*, vol. abs/1806.07851, 2018.

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

[24] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *Proceedings of the 1995 ASME International Congress and Exposition*, 1995.

[25] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on Robot Learning*, PMLR, 2020, pp. 1101–1112.

[26] P. R. Pagilla and B. Yu, "Robotic Surface Finishing Processes: Modeling, Control, and Experiments," *Journal of Dynamic Systems, Measurement, and Control*, vol. 123, no. 1, pp. 93–102, Oct. 1999.

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[28] F. Ruggiero, J.-T. Kim, A. Gutierrez-Giles, A. C. Satici, A. Donaire, J. Cacace, L. R. Buonocore, G. A. Fontanelli, V. Lippiello, and B. Siciliano, "Nonprehensile manipulation control and task planning for deformable object manipulation: Results from the rodyman project," in *International Conference on Informatics in Control, Automation and Robotics*, Springer, 2018, pp. 76–100.

[29] G. Rupert Jr *et al.*, *Simultaneous statistical inference*. Springer Science & Business Media, 2012.

[30] S. S. M. Salehian and A. Billard, "A dynamical-system-based approach for controlling robotic manipulators during noncontact/contact transitions," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2738–2745, 2018.

[31] D. Seita, A. Ganapathi, R. Hoque, M. Hwang, E. Cen, A. K. Tanwani, A. Balakrishna, B. Thananjeyan, J. Ichnowski, N. Jamali, K. Yamane, S. Iba, J. Canny, and K. Goldberg, "Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[32] Student, "The probable error of a mean," *Biometrika*, pp. 1–25, 1908.

[33] R. Tedrake, *Underactuated Robotics, Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2022.

[34] S. Tokumoto and S. Hirai, "Deformation control of rheological food dough using a forming process model," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 2, 2002, pp. 1457–1464.

[35] Z. Wang, Y. Torigoe, and S. Hirai, "A prestressed soft gripper: Design, modeling, fabrication, and tests for food handling," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 1909–1916, 2017.

[36] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel, "Learning to Manipulate Deformable Objects without Demonstrations," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, Jul. 2020.

[37] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," *CoRL*, 2020.

APPENDIX A
EXPERIMENT DETAILS

A. Network Architectures

Our policy consists of a standard PointNet++ [27] encoder and a three-layer MLP with hidden dimensions $[1024, 512, 256]$ and ReLU activations. Before the points are inputted to the encoder, we use a one-hot encoding to differentiate points that belong to the tool v.s. points that belong to the current dough observation v.s. points that belong to the target dough shape. We use PyTorch Geometric's [4] implementation of PointNet++ and use the following modules in our encoder.

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

B. Hyperparameters

We perform a grid search over the different hyperparameters used in our SAC [7] and CEM baselines. The results are denoted in Table A-B. The numbers in a list denotes the values that we search over, and the bolded numbers are the ones used in generating the final results.

TABLE IV: Hyperparameters used in SAC and CEM.

Parameters	Values
α	0.2
Tune alpha	[**True**, False]
σ	0.1
lr	[3e-3, 3e-4, **3e-5**]
Batch size	[5, **10**]
γ	0.99
τ	0.005
$\lambda_{contact}$	[1e-3, **10**, 20]
Training steps	1000000

Parameters	Values
Horizon	[1, 5, **10**, 20]
Pop. size	[50, **100**]
Max iter	[**10**, 20]
Elites	10