

Iterative Residual Policy

for Goal-Conditioned Dynamic Manipulation of Deformable Objects

Cheng Chi¹, Benjamin Burchfiel², Eric Cousineau², Siyuan Feng², Shuran Song¹
¹ Columbia University ² Toyota Research Institute

<https://irp.cs.columbia.edu>

Abstract—This paper tackles the task of goal-conditioned dynamic manipulation of deformable objects. This task is highly challenging due to its complex dynamics (introduced by object deformation and high-speed action) and strict task requirements (defined by a precise goal specification). To address these challenges, we present Iterative Residual Policy (IRP), a general learning framework applicable to repeatable tasks with complex dynamics. IRP learns an implicit policy via delta dynamics – instead of modeling the entire dynamical system and inferring actions from that model, IRP learns delta dynamics that predict the effects of delta action on the previously-observed trajectory. When combined with adaptive action sampling, the system can quickly optimize its actions online to reach a specified goal. We demonstrate the effectiveness of IRP on two tasks: whipping a rope to hit a target point and swinging a cloth to reach a target pose. Despite being trained only in simulation on a fixed robot setup, IRP is able to efficiently generalize to noisy real-world dynamics, new objects with unseen physical properties, and even different robot hardware embodiments, demonstrating its excellent generalization capability relative to alternative approaches.

I. INTRODUCTION

We study the task of goal-conditioned dynamic manipulation of deformable objects, examples of which include whipping a target point with the tip of a rope or spreading a cloth into a target pose (Fig. 1). Humans show remarkable ability to not only perform these tasks with high precision (e.g., aiming a lasso or setting a table cloth), but also to quickly transfer these manipulation skills to new objects with very few attempts. For robots, however, this has historically been a highly challenging problem due to several factors:

- **Complex dynamics:** Unlike quasi-static manipulation, dynamic manipulations (often with high-speed actions) often lead to complex and non-linear effects. These dynamical processes are difficult to model, and pose significant challenges for system identification and state estimation. As a result, it is often infeasible to apply classical algorithms, such as optimal control, when an accurate and performant forward model of the system is prohibitively difficult to obtain.
- **Complex object properties:** Unlike rigid objects, deformable objects exhibit dynamics that are influenced by numerous hard-to-estimate factors such as nonlinear anisotropic stiffness, friction, density distribution, and changing aerodynamics. Not only is it challenging to estimate these properties, it is hard to map them to the parameter used in common simulators using simplified physics models.

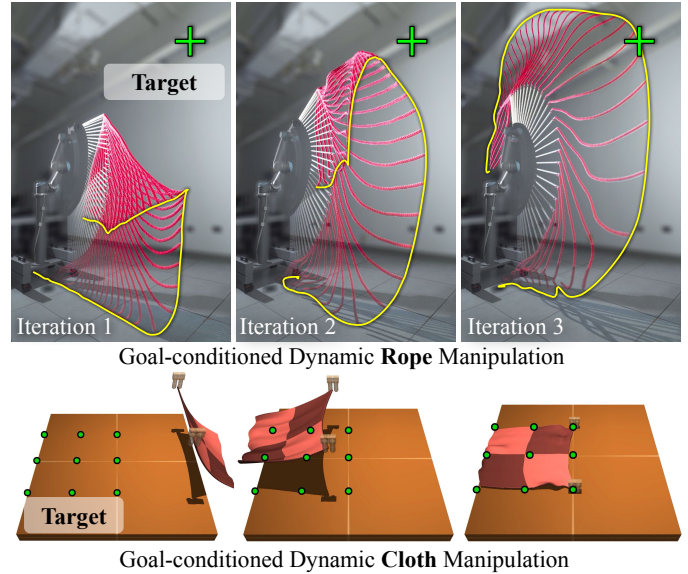


Fig. 1: **Goal-conditioned dynamic manipulation of deformable objects.** Our method achieves high accuracy by iteratively adjusting actions using visual feedback and a learned delta dynamics network. Top: Rope manipulation task. Goal configuration is defined by a target tip position (green cross). Bottom: Cloth manipulation task. Goal configuration is defined by target keypoint locations (green dots).

- **Precise goal conditions:** Due to these challenges, prior work that applies dynamic manipulation to deformable objects generally studies tasks with low precision requirements such as cloth unfolding [6] or cable vaulting [31]. In contrast, the tasks studied in this paper require precise actions in order to reach the specified goal conditions — for example hitting a goal location with the tip of a whip or reaching a goal configurations defined by keypoints on a cloth.

To address these challenges, we present **Iterative Residual Policy (IRP)**, a general formulation for goal-conditioned dynamic manipulation, that learns how to iteratively improve its actions to achieve a given goal condition using visual feedback. This formulation highlights two key features:

- **Iterative action refinement:** Instead of directly inferring the optimal action from the goal, IRP starts with a best guess action and iteratively refines it to move toward the goal. For tasks with repeatable dynamics, this iterative approach allows the system to achieve high precision and be robust against errors from action, observation, and model prediction.
- **Delta dynamics:** Within each iteration, IRP learns to

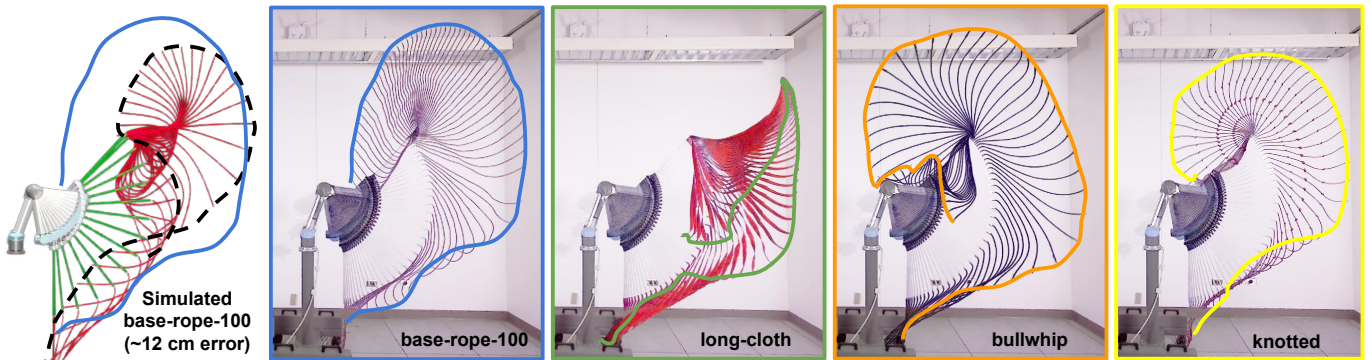


Fig. 2: **Challenges.** Here we show the different rope trajectories under the **same** robot action. Due to different physical properties (e.g., aerodynamic, mass distribution or stiffness), the resulting trajectories varies drastically. Left plot shows the trajectory overlay between the real-world (solid curve) and simulated result (dashed curve) both for base-rope-100. The offset between them indicates the sim2real gap even with the accurate rope parameters. These examples highlight the importance of online adaptation that is necessary for both sim2real transfer and generalization to novel ropes with unknown parameters.

predict an updated trajectory from an observed trajectory with small action perturbations. Compared to modeling the entire dynamical system (action in, trajectory out), this “delta dynamics” (especially its general direction) is much easier to predict and generalize to new objects, thereby, enabling quick online adaption.

Our primary contribution is the Iterative Residual Policy (IRP), a general learning framework for goal-conditioned dynamic manipulation. Despite only trained using simulation data, IRP can be directly applied to real-world hardware. Furthermore, IRP demonstrates impressive generalization capability to unseen object instances, out-of-distribution rope parameters, unmodeled physical effects, and varying robot embodiments. Our experiments show that IRP can achieve pixel level accuracy for a wide variety of goals in both simulation and real robot environments (1.8cm and 2.6cm respectively).

II. RELATED WORK

In this section, we will focus on summarizing relevant prior methods for deformable object manipulation.

Goal-conditioned manipulation is particularly difficult for deformable objects due to their high degree-of-freedom and under-actuation [19, 20, 23, 24, 8, 22]. Previous work studied tasks such as rope knot-tying [24] and fabric folding [8]. Early methods attempt to transfer demonstration trajectories to unseen configurations [22]. Recently, deep learning has been used extensively [23, 24, 8] to enable policy generalization to unseen objects or multiple tasks. However, these methods only consider quasi-static actions, where action effects are inferred mostly from object geometry alone. While quasi-static manipulations has shown to be sufficient to solve many tasks, the resulting systems are often slow and inefficient.

Dynamic manipulation takes advantage of momentum (in addition to kinematic, static and quasi-static mechanisms) to increase load capacity or enable motions to have manipulands extend outside of the robot’s nominal reach range [16, 30, 4, 5, 21, 28, 11]. Deformable objects pose a unique generalization challenge for dynamic manipulation policies due to complex physical properties and a particularly large sim2real gap. Using reinforcement learning, Jangir et al. [9] proposes

a system to dynamically fold and place cloth in various goal configurations. However, thousands of interactions are required for training. Similarly, Lim et al.[13] closes the sim2real gap by algorithmically tuning physics simulation parameters with real-world observations. But this process needs to be repeated for every unseen object instance. Leveraging visual feedback and domain randomization, Hietala et al [7] is able to successfully transfer a manipulation policy trained in simulation to a real robotic setup. However, the policy is limited to a narrow range of goals such as folding clothes exactly in half. In comparison, our method is able to generalize to both unseen real-world object instances as well as wide variety of goals, making it a step closer to real-world applications.

Trajectory optimization uses analytical models [28] or numerical simulation [12, 18, 29, 10, 25] to generate mostly open-loop solutions for deformable object manipulation. Once a model has been designed, or automatically identified[25], the optimization problem can be solved using methods such as direct collocation [10], single shooting [12] or black-box optimization [18]. These methods are generally capable of handling a wide range of goals and generalizing to many object instances, as long as a model is available for each new object. However, direct execution of optimal trajectory solutions on hardware (i.e., the OptSim method described later) does not work well for our tasks due to the large sim2real gap. In contrast, our method bridges this gap using feedback from previous trajectories without an explicit dynamics model.

Iterative Learning Control (ILC) leverages feedback from previous iterations to improve the accuracy of a repeatable system [17, 1]. Most ILC algorithms tackle the task of trajectory tracking control, compensating for repeated disturbances against a reference trajectory. However, when the objective function is non-convex and difficult to optimize (like the two tasks we investigate in this paper), the reference trajectory is hard to obtain. Hence, ILC can not be directly applied. In addition, existing ILC algorithms often assumes known control direction or Jacobian which is difficult to obtain for our task. In contrast, our method is able to iteratively solve these tasks despite its complexity.

III. PROBLEM SETUP

We use two domains as examples of general dynamic manipulation tasks, one featuring 1-D deformable objects (ropes) and the other focusing on 2-D deformable objects (table cloths). For both tasks, the physical parameters of the objects are not known to the algorithm a priori. During testing, a single policy is used for each task and evaluated across a diverse set of manipulands and robot embodiments, demonstrating our method’s strong generalization capability to objects significantly outside of its training distribution.

Rope whipping. The task is to hit a target location in the air with the tip of a rope attached to a UR5 robot (Fig 1 first row). The range of target locations far exceed the robot’s reach range, requiring the robot to swing the rope dynamically. To reach sufficient speed in practice, we extended the UR5’s end-effector with a 50cm long wooden stick. We use a parametric action primitive to describe the robot’s movement. The action space for the whipping primitive is $a = (v, J_2, J_3)$, where $J_2 \in [90, -30]^\circ$ and $J_3 \in [-110, -290]^\circ$ are the target angles for joints two and three, respectively, and $v \in [1.0, 3.14]$ rad/s is the maximum permissible angular velocity across all joints.

This action primitive considers only 2D movement in the Y-Z plane, where target locations are defined in-plane — it is sufficient for this task, since out-of-plane goals could be reached simply by rotating the robot’s base joint. The trajectory of the rope tip T_i is tracked and rasterized to a 256^2 image as observation input. The distance metric $\mathcal{D}(T_i, g)$ for this task is defined as the minimum distance from any point on the trajectory T_i to the goal location g .

Cloth placement. The task is to place a square cloth on the table that reaches the goal configuration specified by 9 keypoints on the cloth (shown in Fig. 1 bottom row). Again, the goal configurations are further than the arm’s maximum reach range, requiring dynamic actions. We assume that the cloth is gripped by two grippers that move in sync and consider the case where the grippers only move within the Y-Z plane, since out-of-plane goals can be reached by horizontally translating the trajectory. Gripper trajectory is parameterized as a cubic spline with 3 via-points, evenly spaced temporally, in the Y-Z plane, Fig. 4. The start point and the Z coordinate for the end point are fixed. The remaining 3 degrees of freedom — Y,Z locations for the second via-point and the Y locations of the third via-point— as well as the total duration of the action, constitute the 4 dimensional action space. The trajectory for all 9 keypoints on the cloth T_i is used as input observation to the algorithm. The distance metric $\mathcal{D}(T_i, g)$ for this task is defined as the mean keypoint distance between the cloth’s final configuration and target configuration.

IV. APPROACH

When attempting to hit a target with an unfamiliar rope, humans will generally not succeed on their first try. However, using physical intuition, people can usually **predict the effect of adjusting their actions**; when swinging a rope, for example, larger force will make the rope tip swing higher. Although the prediction is not perfect, people often adjust their actions in

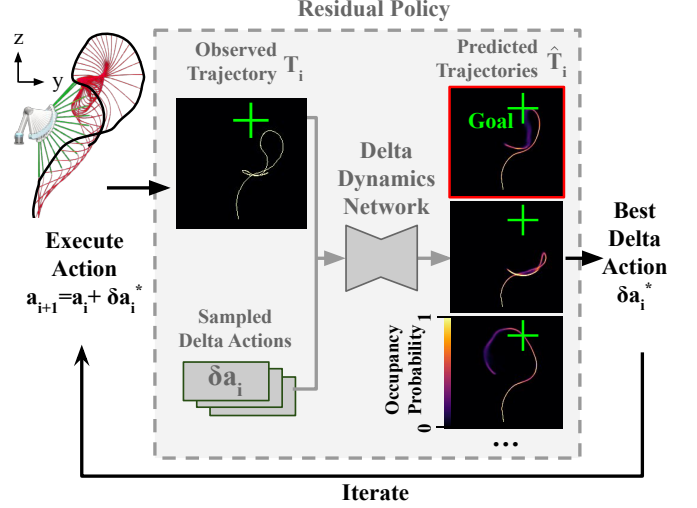


Fig. 3: **Iterative Residual Policy.** During each action execution, the tip trajectory is recorded as an image. For each sampled delta action δa , the Delta Dynamics Network predicts trajectories resulting from applying each delta action. The delta action that is predicted to minimize distance to the target is selected for execution during the next iteration.

Algorithm 1 Iterative Residual Policy

- 1: Input: Goal position g
- 2: Initialize action $a_0 \in \mathbb{R}^{N_a}$ \triangleright §IV-B Initial action
- 3: **while** $i < \text{max_step}$ **do**
- 4: $T_i = \text{robot_execution}(a_i)$
- 5: $d_i = \mathcal{D}(T_i, g)$ \triangleright §III Defined for each task
- 6: **if** $d_i < d_{stop}$ **then**
- 7: **break**;
- 8: **end if**
- 9: $\delta a_i^{0:N_s} = \text{sample_action}(d_i)$ \triangleright §IV-B Delta action
- 10: $\hat{T}_i^{0:N_s} = \text{delta_dynamics}(\delta a_i^{0:N_s}, T_i)$ \triangleright §IV-A
- 11: $j^* = \arg \min_j (\mathcal{D}(\hat{T}_i^j, g))$
- 12: $a_{i+1} = a_i + \delta a_i^{j^*}$
- 13: **end while**

the right direction and quickly drive down their error after a few interactions.

Building on this intuition, we propose **Iterative Residual Policy** for goal-conditioned dynamic manipulation of deformable objects. Given an observation of trajectory T_i by action a_i , we sample a set of potential delta actions δa_i and predict the resulting trajectory \hat{T}_i for each delta action. We then evaluate each of the predicted trajectories based on their minimal distance to the goal d_i , and greedily select the optimal delta action $\delta a_i^{j^*}$ to execute in the next iteration. Trained using simulation data alone, this method is able to achieve high accuracy and generalize to out-of-distribution objects on our real-world robot setup. The following sections provide details for the key algorithm components and design decisions.

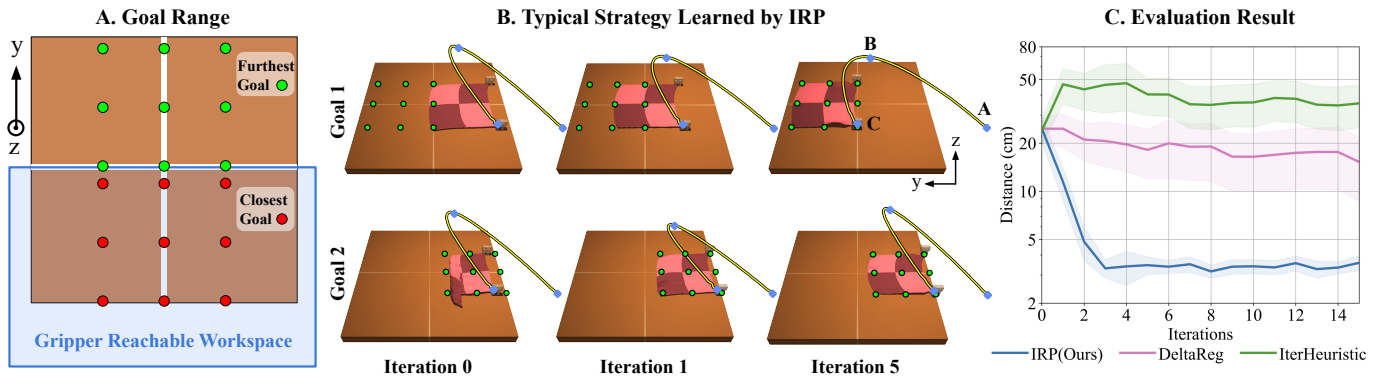


Fig. 4: **Cloth placing task** A) Some goals are placed outside the arm’s reach range, therefore requiring dynamic actions. B) Top row: our algorithm adjusts the trajectory to reach a far-away goal. Bottom row: our algorithm reduces action velocity to eliminate undesired folding. C) Mean distance to goal keypoints averages across all test cloth and goals, shown with a 95% confidence interval. Our method yields significantly better performance compared to the [deltaReg] baseline. Note that due to the stretching of the cloth and thickness of collision capsules, it is not possible to achieve 0 error.

A. Delta Dynamics Network

The delta dynamics network takes in an observed trajectory T_i and a delta action δa_i^j as input and predicts the trajectory \hat{T}_i^j after the delta action is applied. This network is used at every iteration for action optimization.

Trajectory representation. To spatially represent the trajectory T_i , we rasterize it into a 256×256 image projected onto the Y-Z plane. T_i covers $\pm 3m$ area from the robot base joint. The pixel values correspond to occupancy probability; the observed trajectories have a binary pixel and the predicted trajectories are real-valued: $p \in [0, 1]$. For the cloth placing task, the trajectory image of all 9 keypoints are stacked channel-wise. Due to small differences in initial condition, hardware precision, ambient airflow as well as other disturbances, the trajectory is not always the same for each action. Moreover, different segments of the trajectory exhibit different sensitivity to noise.

Action representation. We broadcast the N_a dimensional delta action δa_i into a N_a channel image and then concatenate them with the trajectory images before feeding them into the network. This spatially aligned representation allows us to use a standard CNN architecture designed for image segmentation while ensuring the action information is available in the receptive field of every neuron.

Network and loss. We use DeepLabV3+ [2] for our network architecture. During training, we uniformly select actions as a_i and sample δa_i with a Gaussian distribution (SD=0.125). The resulting trajectories in simulation are used as supervision. The network is trained with Binary Cross Entropy Loss and the AdamW optimizer [15] with learning rate of 0.001 and weight decay of 1×10^{-6} .

B. Interactive Action Sampling and Selection

Initial action. In the rope whipping task, the initial action a_0 is selected using the best average action for all training rope given a specific goal. This action can be computed efficiently with our offline training dataset. The same action is used for a goal regardless of the rope parameters. In the cloth placement

task, a constant initial action is used for all goals to ensure all keypoint trajectories are observable in the first iteration.

Delta action. Since each dimension of the action space has been scaled to between 0 and 1, we sample $N_s = 128$ delta actions δa_i for each iteration from a spherical Gaussian distribution with 0 mean and standard deviation σ . To reduce overshooting and accelerate convergence, we select $\sigma = 0.5 \times d_i$, where $d_i = \mathcal{D}(T_i, g)$.

Action selection. After the network predicts the raw trajectories image for all delta action sample $\delta a_i^{0:N}$, we threshold the prediction with $t = 0.2$, creating a set of binary trajectory image $\hat{T}_i^{0:N}$. Then, the delta action $\delta a_i^j^*$ associated with smallest distance $\mathcal{D}(\hat{T}_i^j, g)$ is selected to compute next action: $a_{i+1} = a_i + \delta a_i^j^*$. In real-world experiments, the optimization stops when the policy reaches $d_{\text{stop}} = 2cm$ error or maximum iteration $\text{max_step} = 10$ is reached. In simulation, the optimization executes for exactly 16 iterations. The algorithm is summarized as pseudocode in Algorithm 1.

C. Training Data Generation

Rope whipping. For training and validation, we built a simulation environment in MuJoCo [26]. The rope is simulated using 25 linked capsules and to generate different ropes, varying two parameters: length and density. The range of both parameters as well as the training/testing split is shown in Fig. 6 left. The rope parameters are sampled from a 12^2 grid. For each rope, we simulate 50^3 actions within the set box constraint for speed $[1, 3.14]$ rad/s, joint 2 $\in [90, -30]^\circ$ and joint 3 $\in [-110, -290]^\circ$. In addition, each action was repeated 3 times with different random perturbations on the initial state to capture the stochastic nature of the trajectory. The entire dataset contains 54 million trajectories.

Clearly this model does not capture all the variations in real-world ropes, and the simulation itself presents different dynamics from the real world due to unmodeled factors such as aerodynamics, collision with the floor, etc. Despite the significant sim-to-real gap, we will later show that our model trained on these simulated ropes generalizes very well in the real world with significantly out-of-distribution ropes.

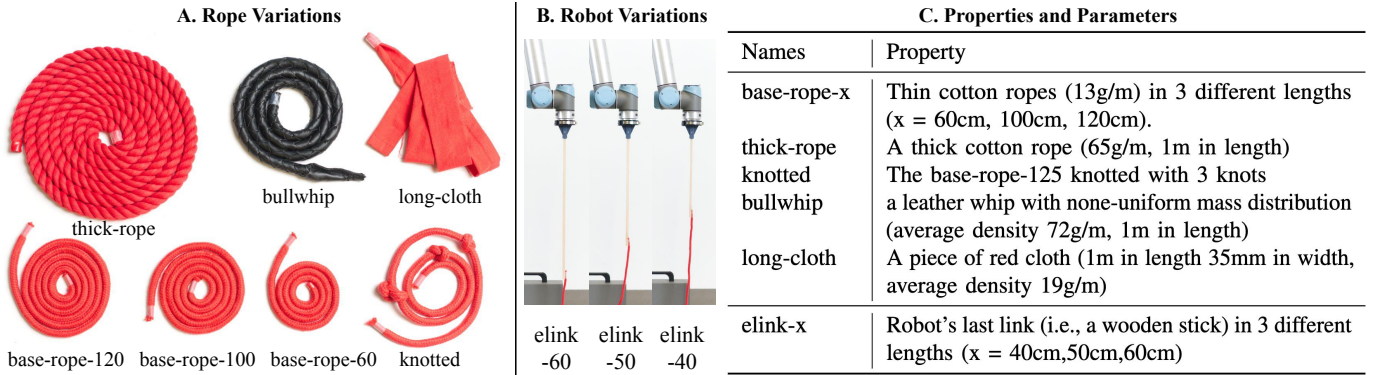


Fig. 5: **Real-world test cases.** A) Real-world testing ropes used to test our algorithm’s generalization to untrained material, length, mass distribution and aerodynamics. B) By changing the the length of last link, the mapping from action parameters to effective swing speed and end-effector trajectory are changed simultaneously. C) Table for key properties and parameters for each real world test case.

Cloth placement. Similarly, the training data for cloth is collected from a MuJoCo simulation environment. The square cloth is simulated as a 13^2 grid of capsules, skinned as a cloth for visualization. We again vary 2 parameters: size (ranged between 0.4 to 0.6 m) and density (ranged between 0.2 to 1.4 kg/m^2). For each cloth parameter, we sample speed and 3 via-point parameters from a 8×16^3 grid. In total, the dataset contains 131 thousand examples.

D. Real world system setup.

We conducted real-world experiments for the rope whipping task with an UR5-CB3 robot augmented with a wooden extension. We used a single RGB camera (Stereolab ZED 2i) running at 720p 60fps for tracking the tip of the rope. Due to the limited resolution and large motion range, stereo tracking does not provide sufficient depth accuracy and reliability. We therefore assume that the rope tip only moves in a 2D plane, allowing us to reduce the perception problem to that of 2D tracking. We placed the camera 2.4m away from the robot, and calibrated the homography between the image plane and the robot Y-Z plane with manually labeled point correspondences. We trained a keypoint detection model based on DeepLabV3+ with 400 hand labeled images. Note that we use a higher image resolution (720p instead of 256p) for tracking results with less than 1.3cm tracking error in the validation set.

V. EVALUATION

In our experiments we seek to validate the system’s generalization capability to 1) novel object physical parameters 2) real-world dynamics, and 3) robot hardware embodiment. In all following experiments, both in simulation and the real-world, we use the same model trained with simulation training ropes generated in Sec. IV-C. The cloth placing task is evaluated in simulation only and is discussed in Sec. VI

A. Rope Whipping Task

Metrics. We measure the performance of these algorithms using the minimum distance to the goal (in cm) at each step. The faster that an algorithm reaches a certain distance the better. We allow the maximum iteration to be 16 in simulation and 10

in real-world experiments. Note that the pixel size is around 2.3cm in our trajectory image encoding and the real-world tracking accuracy is around 1.3cm.

Test Cases. To test the system’s ability to generalize to different rope parameters, we used the following testing ropes:

- Simulated ropes with parameters in the **interpolation** regime relative to the training rope parameters (Fig. 6 green).
- Simulated ropes with parameters in the **extrapolation** regime relative to the training rope parameters (Fig. 6 blue).
- **Real-world ropes** with different material, length, and mass distribution (Fig. 5 C). We modeled the simulation rope after the [base-rope], therefore all other ropes are significantly out-of-distribution. In particular, the [bullwhip], and [knotted-rope] have a non-uniform mass distribution and the [long-cloth] has a larger surface area to density ratio, hence, experiences much larger aerodynamic effects (unmodeled in simulation) than to other ropes.

Robot hardware embodiment. We validate the system’s generalization capability to different robot hardware embodiments by sampling the robot’s last link length from 3 different lengths, [elink-x] where $x = 40\text{cm}, 50\text{cm}, 60\text{cm}$. By changing robot’s last link, we are effectively changing the mapping between the robot’s action parameter and its physical embodiment, which require the system to adapt according to the new trajectory observations. Fig. 7 shows the example results for this experiment, where IRP converges to 3 different actions for the same goal.

Algorithm comparisons. To validate the major design decisions and their impacts on performance, we compare with following alternative approaches:

- **System identification (SysID, SysID_GT):** This method first identifies key system parameters and then infers the action using these parameters. We give this baseline a head start by providing the true varying parameters in simulation: length and density. To ensure fairness, we use 16 fixed system identification actions to collect observations. Another 4-layer MLP is trained to regress the optimal action from estimated rope parameters. [SysID_GT] is a variant of this baseline, where we assume the ground truth rope parameters are given.

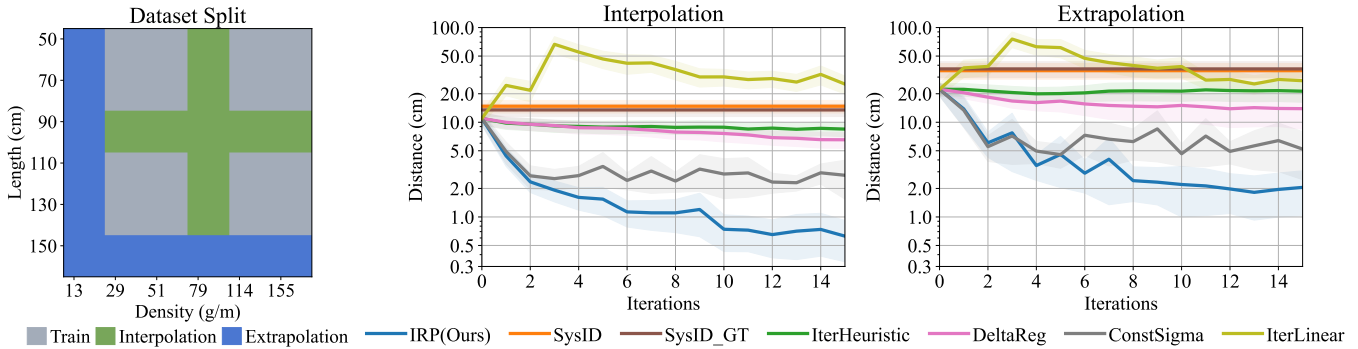


Fig. 6: **Rope Experiments in Simulation.** Left: 12×12 grid of rope parameters used in simulation experiment (grey: training, orange: interpolation, blue: extrapolation). Middle and Right: Distance to goal in cm averaged across 5 ropes and 25 goals for each method, shown with 95% confidence interval. The result of single-step methods (SysID, SysID_GT) are shown as lines for visualization. Testing rope with parameters that extrapolate beyond training ropes parameters (middle) typically result in higher distance errors for most of the alternative approaches. However, IRP is able to achieve low distance errors for both scenarios (0.4cm and 1.5cm respectively).

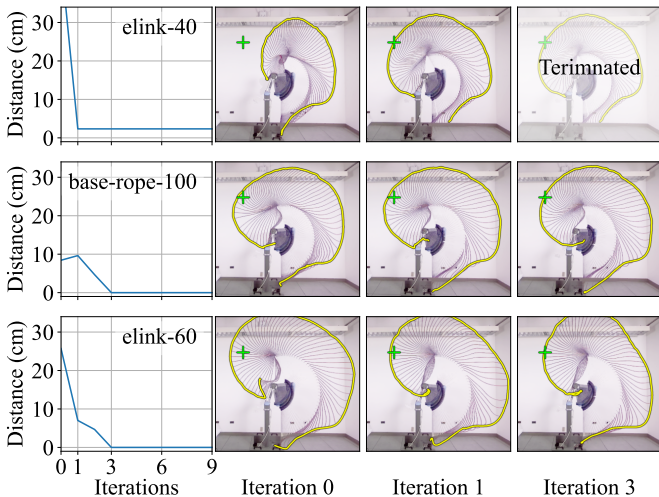


Fig. 7: **Adaptation to robot embodiment.** We varies the robot’s last link with 3 different lengths (40,50,60cm), which effectively changes the mapping between the actions and their effects. As expected, the robot with a different link than training (elink-40/-60) has a higher initial error. However, for all variations, the system is able to reach the goal by adjusting its action according to the visual feedback.

- **Iterative control with heuristic (iterHeuristic):** A heuristic strategy that increases both the speed and amplitude of the action if the trajectory intersects with the line segment between the goal and the origin (the goal is outside the trajectory) and decreases otherwise. If the trajectory does not intersect with the ray from the origin to the goal, the algorithm terminates to prevent increasing error.
- **Iterative control with linear model [iterLinear]:** Instead of leveraging the residual dynamics model learned from offline data, a linear model of the plant is fitted on the trajectories observed so far, updated at each step. The action is adjusted at each step by minimizing the shortest distance to the goal using the linear model.
- **Direct delta action regression [DeltaReg].** Instead of using sampled delta actions as input, this method directly regresses the optimal delta action for the goal and the observed trajectory. The model is trained with MSE loss. We test

rope	OptSim	AVG	il-3	il-9	IRP-3	IRP-9
base-rope-100	21.6	15.6	31.5	13.4	3.2	1.3
base-rope-120	14.4	16.5	51.9	22.5	1.9	1.9
base-rope-60	14.5	19.9	28.4	28.0	8.9	5.5
knotted	14.2	8.3	17.1	9.2	2.6	2.6
thick-rope	11.9	19.7	29.2	10.7	11.7	3.2
long-cloth	15.8	59.6	72.2	16.8	14.0	1.9
bullwhip	17.0	28.7	50.5	8.4	9.0	1.9
elink-40	16.0	28.4	77.5	29.3	13.3	6.1
elink-60	11.9	14.6	30.5	18.5	5.4	3.8

TABLE I: **Realworld Evaluation.** Distance to goal in cm for 7 unseen real world ropes and 2 robot hardware embodiments. [long-cloth] and [bullwhip] are particularly out-of-distribution. Our method significantly outperforms the other approaches we evaluated, demonstrating surprisingly strong generalization to unseen rope parameters, action definitions and simulation/reality divergence. IL: iterLinear. X-3, X-9 error measured in iteration 3 and 9.

it with the same number of iterations.

- **No adaptive action sampling [ConstSigma].** An ablation of our method which uses a fixed sigma for action sampling.
- **Average action in simulation (AVG):** The action that minimizes the average distance to goal in the training set, regardless of rope parameters, this is also the action we used to initialize our method at step one.
- **Optimal control in simulation (OptSim):** This method estimates optimal actions in simulation with measured rope parameters from a real-world (i.e., length, density and width). This method represents an oracle for model-based control using our simulator. It is used only in real-world experiments to quantify the sim2real gap.

B. Experimental Analysis

Benefits of learning residual dynamics.

We hypothesize that this formulation eases the learning process and maintains the flexibility of representing complex non-linear dynamics. To validate this hypothesis, we compared to other alternatives for modeling the system.

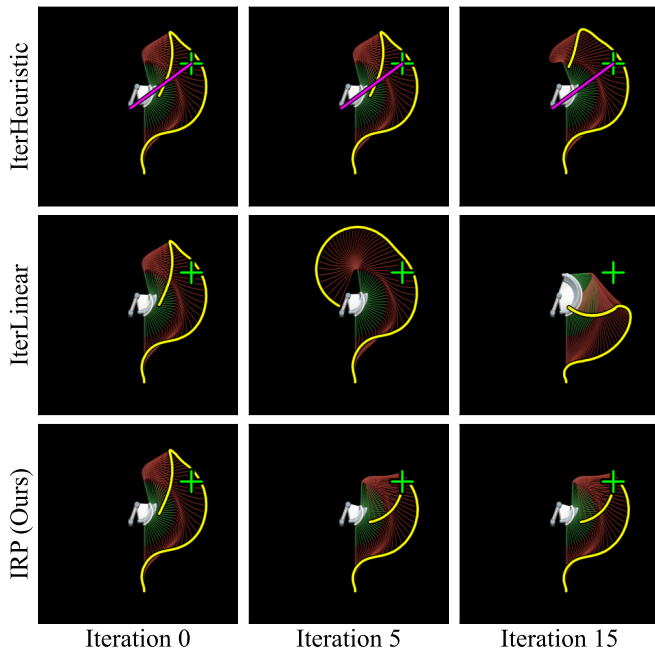


Fig. 8: **Comparison.** In this example, the rope trajectory switches between several discrete modes. As a result, IterHeuristic gets stuck in a sub-optimal orbit and iterLinear failed to approximate the switching behavior. In contrast, our method accurately predicts the switching behavior and reaches the goal.

Compared to the system identification method [SysID, SysID_GT], IRP can better handle unmodeled physical parameters presented in real-world ropes (e.g., aerodynamics or stiffness), which is captured in the input trajectory but not in the predefined system parameters (e.g., length and density). As a result, the algorithm can adapt to real-world ropes much better than SysID methods.

Compared to [DeltaReg], IRP better models the multi-modal aspect of the solution space. Oftentimes there are multiple actions could reach the same goal, where [DeltaReg] tends to predict the mean of all valid solution (due to its MSE objective), which is likely to be an invalid solution.

Compared to [IterHeuristic], our learned IRP model can capture complexity in the rope dynamics, such as mode switching when crossing key velocity thresholds, that the heuristic approach cannot capture. The [IterHeuristic] policy essentially assumes that the tip of the rope swings in a full circle, with its radius controlled by the action’s energy. However, when the energy is insufficient for the tip to swing past apex this assumption is incorrect, and therefore, lead to incorrect action prediction. Moreover, around the apex point, the rope trajectory quickly switches between several modes, causing [IterHeuristic] to get stuck in oscillating local minima (Fig. 8 first row).

Finally, compared to [iterLinear]: our IRP model is able to converge in fewer steps with lower distance. In many cases, the rope trajectories switch between several discrete modes for different actions in a highly non-linear fashion, which cannot be easily captured by the linear model (Fig. 8 second row). In contrast, our model is able to learn a non-linear model from

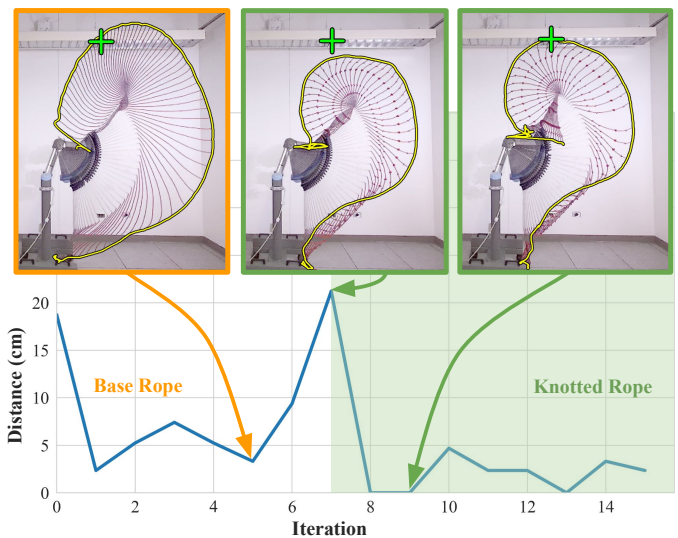


Fig. 9: **Online adaptation to system changes.** In this experiment, the robot first interacts with [base-rope-100]. At step 6, the rope is knotted. We plot the distance to goal with respect to the steps. While the tied knots significantly change the rope’s length, density and mass distribution, the system is able to quickly adjust to the new system dynamics and achieve low errors.

the diverse set of training trajectories.

Benefits of iterative action refinements. With its iterative refinements, IRP can consistently improve its performance from the best average action. On average, the error drops 94% on interpolation experiments and 91% drop on extrapolation experiments by using the additional trajectory observations. Moreover, the comparison with [ConstSigma] shows that the adaptive action sampling method (described in Sec. IV-B) can further improve the action samples and prevent overshooting around the optimal action — demonstrated by higher error and variance for [ConstSigma] in Fig. 6 C after step 4.

Comparison to optimal control in simulation. By computing the action using exhaustive search with manually measured parameters, the performance of [OptSim] represents an *oracle* for trajectory optimization using our MuJoCo simulation environment, and it should achieve perfect result in the simulated environments. However, due to the instability of the dynamical system and unmodeled effects such as aerodynamics, [OptSim] still results in higher error compared to IRP in our real-world experiment (Fig 10, Tab. I). This result demonstrates the importance of online adaptation for closing sim2real gaps, especially for complex dynamical systems such as the dynamic manipulation of deformable objects.

Online adaptation to system changes. In this experiment, we stress test IRP’ robustness against unexpected system changes during the interaction steps. We first ask the robot to interact with the [base-rope-100]. After step 6, we tie four knots on the rope and observe the system behavior. While the tied knots significantly changes the rope’s length, density and mass distribution, we observe that the system is able to quickly adjust to the new system dynamics and regain good performance (Fig. 9).

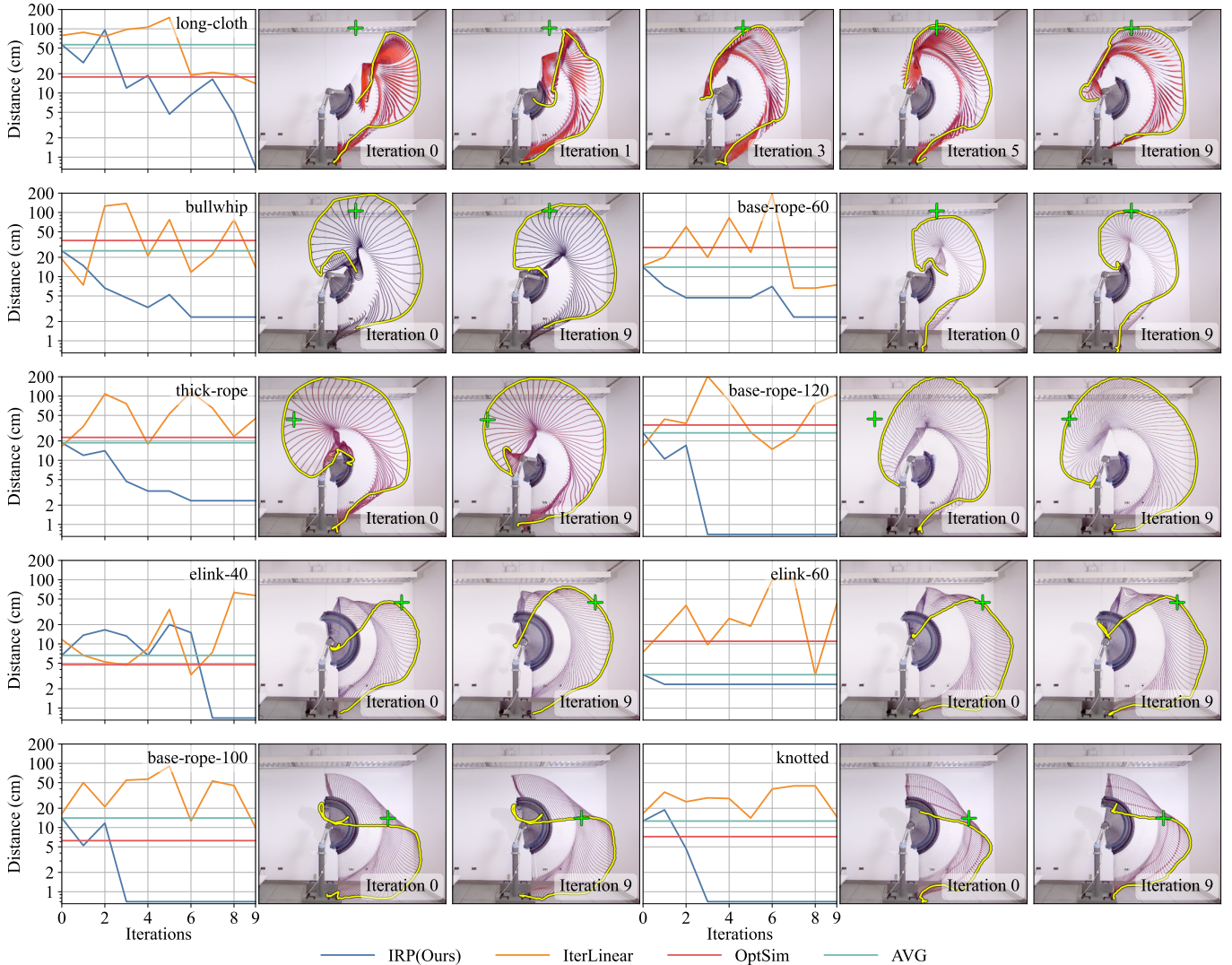


Fig. 10: **Real-world test results on different ropes.** Using the same network trained on simulation data only, our method is able to generalize to significantly out-of-distribution ropes and a wide range of goals. First row: step 0,1,3,5 and 9 for long-cloth test case. Second row: step 0, 9 for other test cases. OptSim is not able to achieve low error, indicating the large sim2real gap in this task. All distances are shown in **log scale**. iterLinear failed to find good solutions within 9 steps, since it uses online observations only and has limited model capacity. More results can be find in the supplemental video.

VI. EXTENSION TO CLOTH MANIPULATION

To demonstrate the generality of IRP’s, we applied the same method to a cloth placement task with minimal modifications.

Test cases. We randomly sampled five unseen cloth parameter pairs (size, density) as our testing cases. For each cloth sample, we uniformly sampled 11 goal configurations from the range illustrated in Fig. 4 A. Note that since test cloths all have different sizes, the specific target keypoint locations are adjusted accordingly.

Metrics. The performance is measured by the average distance to the goal (in cm) over all keypoints. Note that due to the stretching effects on the cloth and the thickness of MuJoCo’s capsule model for collision, it is not possible to reach zero error for a target configuration with perfectly square and flat keypoint locations.

Result. Fig 4 B and C show the qualitative and quantitative

results respectively. Fig 4 B shows that IRP can adjust the action to different landing locations and shape of the cloth (e.g. to avoid folding). We compared with [DeltaReg] and [IterHeuristic], two of the best performing baseline methods from the rope whipping task. We modified the [IterHeuristic] to increase all action parameters if the landing keypoints are closer than the goal, and decreases otherwise, with the proportional gain set to 0.5. On average, IRP achieves 3.5 cm error across 11 test goal configurations, yielding better performance and lower variance comparing to other methods.

VII. CONCLUSION

This paper presents a general learning framework for goal-conditioned dynamic manipulation: Iterative Residual Policy. Our extensive experiments in both simulation and the real-world demonstrate its adaptability to many aspects of the

system, including object parameters, real-world dynamics, and even robot hardware embodiment. However, our method is not without limitations, one of which is the assumption of action repeatability, which can not be guaranteed in many applications. We also assume full observability of the manipuland throughout the trajectory, which makes direct application of this approach difficult in highly cluttered scenarios. Future work could explore using different visual feedback [14, 3] to deal with non-repeatable action scenarios and perception techniques leveraging temporal cues to handle partial observability [27].

ACKNOWLEDGEMENT

We would like to thank Zhenjia Xu, Huy Ha, Dale McConachie, Naveen Kuppaswamy for their helpful feedback and fruitful discussions. This work was supported by the Toyota Research Institute, NSF CMMI-2037101 and NSF IIS-2132519. We would like to thank Google for the UR5 robot hardware. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

REFERENCES

- [1] D.A. Bristow, M. Tharayil, and A.G. Alleyne. A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(3):96–114, 2006. doi: 10.1109/MCS.2006.1636313. 2
- [2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 833–851, Cham, 2018. Springer International Publishing. 4
- [3] Cheng Chi and Shuran Song. Garmentnets: Category-level pose estimation for garments via canonical space shape completion. In *The IEEE International Conference on Computer Vision (ICCV)*, 2021. 9
- [4] Adrià Colomé and Carme Torras. Dimensionality reduction for dynamic movement primitives and application to bimanual manipulation of clothes. *IEEE Transactions on Robotics*, 34(3):602–615, 2018. doi: 10.1109/TRO.2018.2808924. 2
- [5] Adrià Colomé, Antoni Planells, and Carme Torras. A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5649–5654, 2015. doi: 10.1109/ICRA.2015.7139990. 2
- [6] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. *arXiv preprint arXiv:2105.03655*, 2021. 1
- [7] Julius Hietala, David Blanco-Mulero, Gokhan Alcan, and Ville Kyrki. Closing the sim2real gap in dynamic cloth manipulation. *arXiv preprint arXiv:2109.04771*, 2021. 2
- [8] 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. 2
- [9] Rishabh Jangir, Guillem Alenyà, and Carme Torras. Dynamic cloth manipulation with deep reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4630–4636, 2020. doi: 10.1109/ICRA40945.2020.9196659. 2
- [10] Shiyu Jin, Diego Romeres, Arvind Raganathan, Devesh K Jha, and Masayoshi Tomizuka. Trajectory optimization for manipulation of deformable objects: Assembly of belt drive units. *arXiv preprint arXiv:2106.00898*, 2021. 2
- [11] Kento Kawaharazuka, Toru Ogawa, Juntaro Tamura, and Cota Nabeshima. Dynamic manipulation of flexible objects with torque sequence using a deep neural network. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2139–2145, 2019. doi: 10.1109/ICRA.2019.8793513. 2
- [12] Yinxiao Li, Yonghao Yue, Danfei Xu, Eitan Grinspun, and Peter K. Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6000–6006, 2015. doi: 10.1109/IROS.2015.7354231. 2
- [13] Vincent Lim, Huang Huang, Lawrence Yunliang Chen, Jonathan Wang, Jeffrey Ichnowski, Daniel Seita, Michael Laskey, and Ken Goldberg. Planar robot casting with real2sim2real self-supervised learning. *arXiv preprint arXiv:2111.04814*, 2021. 2
- [14] Xingyu Lin, Yufei Wang, Zixuan Huang, and David Held. Learning visible connectivity dynamics for cloth smoothing. In *Conference on Robot Learning*, 2021. 9
- [15] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 4
- [16] Matthew T. Mason and Kevin Lynch. Dynamic manipulation. In *Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 152 – 159, July 1993. 2
- [17] Kevin L. Moore, M. Johnson, and Michael J. Grizzle. *Iterative Learning Control for Deterministic Systems*. Springer-Verlag, Berlin, Heidelberg, 1993. ISBN 0387197079. 2
- [18] Moses C. Nah, Aleksei Krotov, Marta Russo, Dagmar Sternad, and Neville Hogan. Dynamic primitives facilitate manipulating a whip. In *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechanics (BioRob)*, pages 685–691, 2020. doi: 10.1109/BioRob49111.2020.9224399. 2
- [19] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2146–2153, 2017. doi: 10.1109/ICRA.2017.7989247. 2
- [20] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Fred Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell.

- Zero-shot visual imitation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2131–21313, 2018. doi: 10.1109/CVPRW.2018.00278. 2
- [21] Fabio Ruggiero, Vincenzo Lippiello, and Bruno Siciliano. Nonprehensile dynamic manipulation: A survey. *IEEE Robotics and Automation Letters*, 3(3):1711–1718, 2018. doi: 10.1109/LRA.2018.2801939. 2
- [22] John Schulman, Jonathan Ho, Cameron Lee, and Pieter Abbeel. *Learning from Demonstrations Through the Use of Non-rigid Registration*, pages 339–354. Springer International Publishing, Cham, 2016. ISBN 978-3-319-28872-7. doi: 10.1007/978-3-319-28872-7_20. URL https://doi.org/10.1007/978-3-319-28872-7_20. 2
- [23] Daniel Seita, Pete Florence, Jonathan Tompson, Erwin Coumans, Vikas Sindhwani, Ken Goldberg, and Andy Zeng. Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4568–4575, 2021. doi: 10.1109/ICRA48506.2021.9561391. 2
- [24] Priya Sundareshan, Jennifer Grannen, Brijen Thananjeyan, Ashwin Balakrishna, Michael Laskey, Kevin Stone, Joseph E. Gonzalez, and Ken Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9411–9418, 2020. doi: 10.1109/ICRA40945.2020.9197121. 2
- [25] Kenta Tabata, Hiroaki Seki, Tokuo Tsuji, and Tatsuhiro Hiramitsu. Casting manipulation of unknown string by robot arm. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 668–674, 2021. doi: 10.1109/IROS51168.2021.9635837. 2
- [26] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. 4
- [27] Zhenjia Xu, Zhanpeng He, Jiajun Wu, and Shuran Song. Learning 3d dynamic scene representations for robot manipulation. *arXiv preprint arXiv:2011.01968*, 2020. 9
- [28] Yuji Yamakawa, Akio Namiki, and Masatoshi Ishikawa. Motion planning for dynamic folding of a cloth with two high-speed robot hands and two high-speed sliders. In *2011 IEEE International Conference on Robotics and Automation*, pages 5486–5491, 2011. doi: 10.1109/ICRA.2011.5979606. 2
- [29] Eiichi Yoshida, Ko Ayusawa, Ixchel G. Ramirez-Alpizar, Kensuke Harada, Christian Duriez, and Abderrahmane Kheddar. Simulation-based optimal motion planning for deformable object. In *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 1–6, 2015. doi: 10.1109/ARSO.2015.7428219. 2
- [30] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. 2019. 2
- [31] Harry Zhang, Jeffrey Ichnowski, Daniel Seita, Jonathan Wang, Huang Huang, and Ken Goldberg. Robots of the lost arc: Self-supervised learning to dynamically manipulate fixed-endpoint cables. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4560–4567. IEEE, 2021. 1