Iterative residual policy: For goal-conditioned dynamic manipulation of deformable objects

Cheng Chi1, Benjamin Burchfield2, Eric Cousineau2, Siyuan Feng2 and Shuran Song1

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.

Keywords
Deformable object manipulation, dynamics, vision for manipulation, deep learning

Received 18 December 2022; Revised 24 July 2023; Accepted 10 August 2023
Senior Editor: Jose-Luis Blanco
Associate Editor: Dylan Shell

1. 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 (Figure 1). Humans show a remarkable ability not only to perform these tasks with high precision (e.g., aiming a lasso or setting a tablecloth) but also to quickly transfer these manipulation skills to new objects with very few attempts. For robots, however, this has been a 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, but it is also hard to map them to the parameter used in common simulators using simplified physics models.

- **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 Ha and Song (2021) or cable vaulting Zhang et al. (2021). 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 goal configurations defined by keypoints on a cloth. Figure 2.

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 learn to 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 being 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 (1.8 cm and 2.6 cm, respectively).

2. 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 Berenson (2013); Nair et al. (2017); McConachie et al. (2020); Pathak et al. (2018); Seita et al. (2021); Sundaresan et al. (2020); Hoque et al. (2020); Schulman et al. (2016). Previous work studied tasks such as rope knot-tying Sundaresan et al. (2020) and fabric folding Hoque et al. (2020). Early methods attempt to transfer demonstration trajectories to unseen configurations Schulman et al. (2016). Berenson (2013) proposed to approximate the Jacobian of the deformable object as a function of its geometry. Recently, deep learning has been used extensively Seita et al. (2021); Sundaresan et al. (2020); Hoque et al. (2020) 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 have 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 Mason and Lynch (1993); Zeng et al. (2019); Colomé and Torras (2018); Colomé et al. (2015); Ruggiero et al. (2018); Yamakawa et al. (2011); Kawaharazuka et al. (2019). 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. (2020) proposes a system to dynamically fold and place the cloth in various goal configurations. However, thousands of interactions are required for every unseen object instance. Leveraging visual feedback and domain randomization, Hietala et al. (2021) 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 a wide variety of goals, making it a step closer to real-world applications.

**Trajectory optimization** uses analytical models Yamakawa et al. (2011) or numerical simulation Li et al. (2015); Nah et al. (2020); Yoshida et al. (2015); Jin et al. (2021); Tabata et al. (2021) to generate mostly open-loop solutions for deformable object manipulation. Once a model has been designed, or automatically identified Tabata et al. (2021), the optimization problem can be solved using methods such as direct collocation Jin et al. (2021), single shooting Li et al. (2015) or black-box optimization Nah et al. (2020). 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 Moore et al. (1993); Bristow et al. (2006). 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 cannot be directly applied. In addition, existing ILC algorithms often assume a 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 their complexity.

3. 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 (tablecloths). 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.

3.1. Rope whipping

The task is to hit a target location in the air with the tip of a rope attached to a UR5 robot (Figure 1 first row). The range of target locations far exceeds 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 50 cm 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 2 and 3, 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 $256 \times 2$ image as observation input. The distance metric $D(T_i, g)$ for this task is defined as the minimum distance from any point on the trajectory $T_i$ to the goal location $g$.

### 3.2. Cloth placement

The task is to place a square cloth on the table that reaches the goal configuration specified by nine keypoints on the cloth (shown in Figure 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 three via-points, evenly spaced temporally, in the Y-Z plane, Figure 3. The start point and the Z coordinate for the endpoint 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 four dimensional action space. The trajectory for all nine keypoints on the cloth $T_i$ is used as input observation to the algorithm. The distance metric $D(T_i, g)$ for this task is defined as the mean keypoint distance between the cloth’s final configuration and target configuration.

#### Algorithm 1 Iterative Residual Policy

1. **Input:** Goal position $g$
2. **Initialize** action $a_0 \in \mathbb{R}^{N_a}$ \(\triangleright 4.2\) Initial action
3. **while** $i < \text{max\_step}$
4. \hspace{1em} $T_i = \text{robot\_execution}(a_i)$
5. \hspace{1em} $d_i = D(T_i, g)$ \(\triangleright 3\) Defined for each task
6. \hspace{1em} **if** $d_i < d_{stop}$ **then**
7. \hspace{2em} break;
8. **end if
9. \hspace{1em} $\delta a_i^{0:N_r} = \text{sample\_action}(d_i)$ \(\triangleright 4.2\) Delta action
10. \hspace{1em} $T_i^{0:N_r} = \text{delta\_dynamics}(\delta a_i^{0:N_r}, T_i)$ \(\triangleright 4.1\)
11. \hspace{1em} $j^* = \arg \min_j \{D(T_i^{j}, g)\}$
12. \hspace{1em} $a_{i+1} = a_i + \delta a_i^{j*}$
13. **end while

### 4. 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, a larger force will make the rope tip swing higher. Although the prediction is not perfect, people often adjust their actions in the right direction and quickly drive down their errors 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 $\delta a_i$, and predict the resulting trajectory $T_i$ 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 Figure 4.

4.1. Delta dynamics network

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

4.1.1. Trajectory representation. To spatially represent the trajectory \( T_i \), we rasterize it into a \( 256 \times 256 \) 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 nine keypoints is stacked channel-wise. Due to small differences in the 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.

4.1.2. Action representation. We broadcast the \( N_a \) dimensional delta action \( \delta a \) 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.

4.1.3. Network and loss. We use DeepLabV3+ Chen et al. (2018) for our network architecture. During training, we uniformly select actions as \( a_i \) and sample \( \delta a_i \) with a Gaussian distribution (SD = 0.125). The resulting trajectories in the simulation are used as supervision. The network is trained with Binary Cross Entropy Loss and the AdamW optimizer Loshchilov and Hutter (2019) with a learning rate of 0.001 and weight decay of \( 1 \times 10^{-6} \).

4.2. Interactive action sampling and selection

4.2.1. Initial action. In the rope whipping task, the initial action \( a_0 \) is selected using the best average action for all training ropes 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.

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

4.2.3. Action selection. After the network predicts the raw trajectories image for all delta action sample \( \delta a \), we threshold the prediction with \( t = 0.2 \), creating a set of binary trajectory image \( \tilde{T}_i \). Then, the delta action \( \delta a_i \) associated with smallest distance \( D(\tilde{T}_i, g) \) is selected to compute next action: \( a_{i+1} = a_i + \delta a_i \). 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 the simulation, the optimization executes for exactly 16 iterations. The algorithm is summarized as pseudocode in Algorithm 1.

4.3. Training data generation

4.3.1. Rope whipping. For training and validation, we built a simulation environment in MuJoCo Todorov et al. (2012). 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 Figure 5 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 and collision with the floor. 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.

4.3.2 Cloth placement. Similarly, the training data for cloth is collected from a MuJoCo simulation environment. The square cloth is simulated as a 132 grid of capsules, skinned as a cloth for visualization. We again vary 2 parameters: size (ranged between 0.4 and 0.6 m) and density (ranged between 0.2 and 1.4 kg/m²). For each cloth parameter, we sample speed and three via-point parameters from an 8 × 163 grid. In total, the dataset contains 131 thousand examples.

4.4 Real world system setup

We conducted real-world experiments for the rope whipping task with a 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 2.4 m away from the robot and calibrated the homography between the image plane and the robot Y-Z plane with manually labeled point correspondences.

4.5 Rope keypoint tracking system

To track the tip of the rope from videos, we repurposed the DeepLabV3+ semantic segmentation architecture for keypoint detection. We manually labeled the rope tip locations for 400 images for training and validation. We train the model with a per-pixel BCE loss using a Gaussian-blurred one-hot-pixel heat map as the training target. After each robot action, each frame in the 720p video is fed through the keypoint detection model to predict a sequence of keypoints. We then generate the 256x256 tip trajectory image by drawing keypoints connected by line segments. Our tracking model has less than a 1.3 cm keypoint error on the validation set. The keypoint tracking pipeline is illustrated in Figure 6.

5 Evaluation

5.1 Rope whipping task setup

5.1.1 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 2.3 cm in our trajectory image encoding and the real-world tracking accuracy is around 1.3 cm. Table 1

5.1.2 Test cases. To test the model’s generalization capability, we used the following testing ropes:

- Simulated ropes with parameters interpolates training rope parameters (Figure 5 green).
- Simulated ropes with parameters extrapolates beyond training rope parameters (Figure 5 blue).
- Real-world ropes with different material, length, and mass distribution (Figure 7(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.
Figure 6. Rope Keypoint Tracking System. The DeepLabV3 tracking model maps an observed RGB image into a heatmap for the tip location of the rope, from which the keypoint coordinate can be obtained by taking argmax over all pixels. The tip trajectory image is obtained by running the keypoint model for each frame in the video and drawing line segments between consecutive keypoints.

Table 1. Rope experiments in simulation. Error in cm for different methods at iteration 3, 9, and 15 from the rope simulation experiment. Same as Figure 5 middle and right.

| Iteration | Interpolation | Extrapolation | Interpolation | Extrapolation |
|-----------|----------------|----------------|----------------|----------------|
| IRP-img   | 1.4            | 5.5            | 2.0            | 3.2            |
| IRP-coord | 0.8            | 9.0            | 1.1            | 2.2            |
| SysID     | 10.4           | 24.8           | 9.5            | 24.8           |
| SysID-GT  | 6.5            | 25.9           | 6.3            | 25.9           |
| IterHeur  | 1.8            | 14.6           | 2.3            | 15.1           |
| DeltaReg  | 47.1           | 11.9           | 21.3           | 10.3           |
| ConstSigma| 47.1           | 5.1            | 21.3           | 6.0            |
| IterLinear| 53.5           | 17.9           | 26.3           | 19.4           |

Figure 7. Real-world test cases. (a) Real-world testing ropes are used to test our algorithm’s generalization to untrained material, length, mass distribution, and aerodynamics. (b) By changing the length of the last link, the mapping from action parameters to effective swing speed and end-effector trajectory is changed simultaneously. (c) Table for key properties and parameters for each real-world test case.
to density ratio; hence, experiences much larger aerodynamic effects (unmodeled in simulation) than to other ropes.

5.1.3. Robot hardware embodiment. We validate the system’s generalization capability to different robot hardware embodiments by sampling the robot’s last link length from three different lengths, [elink-x] where x = 40 cm, 50 cm, 60 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. Figure 8 shows the example results for this experiment, where IRP converges to three different actions for the same goal.

5.1.4. 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.

- **Iterative control with heuristic action direction [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 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.

5.1.5. Rope trajectory representations. We evaluate the performance impact of two trajectory representations: image and coordinate sequence in simulation. The image representation is a 256x256 occupancy grid, with the occupancy probability being the value for each pixel. We convert the occupancy probability image during inference into a binary occupancy grid using a manually selected threshold, from which the distance to the goal can be computed. In contrast, the coordinate sequence representation is a 400x2 matrix, corresponding to the 2D coordinates of the rope tip trajectory in 400 equally-spaced timesteps. The last coordinate is repeated if the experiment terminates earlier than 400 timesteps.

5.1.6. Delta trajectory prediction accuracy. To evaluate the prediction accuracy of our delta dynamics model independently from the overall system performance, we sampled 10,000 rope-action pairs from both interpolation and extrapolation rope parameters. For each rope-action pair, we sample delta actions with a standard deviation of 0.05 or 0.125 and compute the Chamfer distance between the model prediction and the ground truth trajectory data with the delta action applied. The result is shown in Table 2.

5.2. Experimental results and analysis

5.2.1. Benefits of learning delta 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 it to other alternatives for modeling the system.

Compared to the system identification method \([\text{SysID, SysID GT}]\), 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 \([\text{DeltaReg}]\), IRP better models the multimodal aspect of the solution space. Oftentimes there are multiple actions that could reach the same goal, where \([\text{DeltaReg}]\) tends to predict the mean of all valid solutions (due to its MSE objective), which is likely to be an invalid solution.

Compared to \([\text{IterHeuristic}]\), 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 \([\text{IterHeuristic}]\) 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 the apex this assumption is incorrect and therefore, leads to incorrect action prediction. Moreover, around the apex point, the rope trajectory quickly switches between several modes, causing \([\text{IterHeuristic}]\) to get stuck in oscillating local minima (Figure 9 first row). Moreover, the step size gain of this heuristic method also needs to be carefully tuned to prevent the algorithm from diverging. It is especially true when the action is close to the limit or the rope trajectory is changing between distinct modes (Figure 9 first row). In these cases, the heuristic policy often moves in the incorrect direction or causes little change to the trajectory, resulting in poor performance.

Finally, compared to \([\text{IterLinear}]\), our IRP model is able to converge in fewer steps with a 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 (Figure 9 s row). In contrast, our model is able to learn a non-linear model from the diverse set of training trajectories.

5.2.2. Benefits of iterative action refinements. With its iterative refinements, IRP can consistently improve its performance from the initial 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 \([\text{ConstSigma}]\) shows that the adaptive action sampling method (described in Section 4.2) can further improve the action samples and prevent overshooting around the optimal action—demonstrated by higher error and variance for \([\text{ConstSigma}]\) in Figure 5 C after step 4.

5.2.3. Comparison to optimal control in simulation. By computing the action using exhaustive search with manually measured parameters, the performance of \([\text{OptSim}]\) represents an oracle for trajectory optimization using our MuJoCo simulation environment, and it should achieve perfect results in the simulated environments. However, due to the instability of the dynamical system and unmodeled effects such as aerodynamics, \([\text{OptSim}]\) still results in higher error compared to IRP in our real-world experiment (Figure 10, Table 3). This result demonstrates the importance of online adaptation for closing sim2real gaps, especially for complex dynamical systems.

5.2.4. Online adaptation to system changes. In this experiment, we stress test IRP’s robustness against unexpected system changes during the interaction steps. We first ask the robot to interact with the \([\text{base-rope-100}]\). After step 6, we tie four knots on the rope and observe the system behavior. While the tied knots significantly change 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 (Figure 11).

5.2.5. The effect of trajectory representations. Comparing the image and coordinate sequence representation, we found that the image representation performs better in delta trajectory prediction accuracy (shown in Table 2) and overall system performance (shown in Figure 5). We hypothesize that image representation has the advantage of capturing uncertainty in training data. We observed that the tip movement becomes unpredictable toward the end of each trajectory. With the image representation, the delta dynamics network blurs the end portion of the trajectory image to capture uncertainty. However, the coordinate sequence representation will always have a large L2 loss at the end of each trajectory, biasing the network to focus less on predictable parts of the trajectory.

Table 3. Real world evaluation. Distance to goal in cm for seven unseen real world ropes and two 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 three and nine.

| Rope            | OptSim | AVG | II-3 | II-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   |

Bold values means the best (lowest error) for each row.

Figure 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 the long-cloth test case. Second row: step 0 and 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 nine steps since it uses online observations only and has limited model capacity. More results can be found in the supplemental video.
5.2.6. **Experiment with physical target object.** In most of our experiments, we define the target position in the form of a 2D pixel and assume no out-of-plane movement. To validate this assumption and test the system’s ability to hit physical targets we conduct another set of real-world experiments. In this experiment, we tested the robot to hit a small cup in mid-air supported by a tripod at three different target locations. The plastic toy cup used has a diameter of 4 cm and a length of 6 cm. As shown in Figure 12, all three targets are hit by the robot within eight steps, despite starting from an action that is quite far from optimal. There are cases where the tip of the rope overlaps with the target location in 2D but misses the cup (out-of-plane movement). However, since IRP will only make minor action adjustments in this case due to the adaptive delta action sampling mechanism, the rope hits the target cup eventually.

6. **Extension to cloth manipulation**

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

6.1. **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 Figure 3(a). Note that since test cloths all have different sizes, the specific target keypoint locations are adjusted accordingly.

6.2. **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 MuJo-Co’s capsule model for collision, it is not possible to reach zero error for a target configuration with perfectly square and flat keypoint locations.

6.3. **Result**

Figure 3(b) and (c) show the qualitative and quantitative results respectively. Figure 3(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.
7. Failure case analysis: Local minima

One prominent failure mode of IRP is converging to suboptimal local minima. Since the iterative improvement process of IRP only searches for local improvements to the previous action, IRP is susceptible to the same limitations of gradient descent when the loss landscape is non-convex. In practice, we mostly avoid this problem on the rope whipping task by selecting the initial action $a_0$ using the best average action for all training ropes, as described in Section 4.2. However, this could still limit the applicability of IRP to problems where the loss landscape has numerous local minima or where obtaining a reasonable prior estimate of initial action is impractical.

To better illustrate this failure mode, we exaggerate the problem by intentionally selecting a bad initial action in the convergent basin of suboptimal local minima, shown in Figure 13. The loss landscape of this particular combination of goal and rope is visualized by enumerating the entire 3D action space as a $50^3$ grid in simulation. For each action, we measure the closest distance from the rope tip trajectory to the goal, which is then used to color each voxel in the loss landscape volume. Note that IRP’s local updates only explore a small portion of this suboptimal basin and do not discover the second mode containing the global minimum.

8. Robustness to delta dynamic error

To set the stage for future theoretical analysis of IRP and to examine the effects of misestimated delta dynamics on performance, we applied a simplified toy version of IRP on a 2D projectile dynamics system with quadratic drag. Because the true dynamics system is simple, we can apply the IRP algorithm with its delta-dynamics neural network model replaced with an explicit quadratic equation that does not consider aerodynamic drag.

The task is to accurately hit a 2D target with a projectile by adjusting the projectile’s initial velocity in the x-y plane (subject to a maximum speed constraint). The projectile’s trajectory is influenced by gravity and a quadratic aerodynamics drag term. This 2D point dynamics system can be expressed as ordinary differential equations

$$\begin{align*}
\dot{x} &= -ck\sqrt{x^2 + y^2} \\
\dot{y} &= -g - cy\sqrt{x^2 + y^2},
\end{align*}$$

where $g = 9.8$ is gravity acceleration and $c = c_d A \rho / m$ is the lumped drag acceleration coefficient that combines the effect of the common drag coefficient, reference area, air density, and the projectile mass. Given the initial velocity

$$\vec{v}_0 = (\hat{x}_0, \hat{y}_0) = \text{action}$$

We can solve the projectile trajectory as a simple initial value problem (represented as the function $env$ in Figure 14(a)).

The high-level overview of our simplified IRP for this task is illustrated as pseudocode in Figure 14(a). After executing the initial action and obtaining a trajectory, we fit a parabola from three points on the trajectory: the origin $(0,0)$, the closest point to the goal, and the point $\delta t$ before the closest point on the trajectory (allowing good approximation of the tangent around the closest point), obtaining $a^*, b^*, c^*$ from the best-fit equation $\hat{y} = ax^2 + bx + c$. We then obtain the “virtual action” $\vec{v}_0$ we would have taken to obtain this best-fit parabolic trajectory in a vacuum:
Figure 13. Typical Failure Case. Similar to gradient descent, IRP can converge to sub-optimal local minima due to bad initialization. By enumerating the entire 3D action space as a 50^3 grid in simulation, we can visualize the loss landscape of a particular rope and goal combination (lighter color is better). The sequence of actions executed by IRP is shown as a series of blue arrows. The optimal action computed via exhaustive search is shown as a yellow star.

Figure 14. Simplified IRP on Projectile Dynamics The task is to hit a 2D goal position (the red cross) with a projectile influenced by gravity and quadratic aerodynamic drag. The simplified IRP algorithm (a) with fixed explicit delta-dynamics is visualized in (b) as dashed lines. The simplified IRP algorithm quickly converges to the goal despite using an inaccurate dynamics model shown in (c). A baseline strategy that takes the optimal action according to the vacuum projectile model does not work, and exhibits error that increases with the drag coefficient, while the simplified IRP strategy maintains its low error despite decreasing model accuracy, shown in (d). All goals and trajectories from IRP as well as the baseline method are shown in (e).
\[ x_0^* = \sqrt{\frac{g}{2 \alpha}} \]
\[ y_0^* = b^* x_0 \]
\[ \bar{v}_0^* = (x_0^*, y_0^*). \]

Then, following the original IRP algorithm, we sample several delta actions \( \delta \bar{v}_0 \) from a Gaussian distribution whose standard deviation is proportional to the current error (trajectory distance to the goal). For each delta action, we compute the distance between the vacuum trajectory (parabolic) if the action \( \bar{v}_0^* + \delta \bar{v}_0 \) is taken and obtain the best delta action. Note that the use of a parabola to predict the updated trajectory given the current trajectory and a delta action is analogous to the delta dynamics neural network model used in the original IRP algorithm.

Finally, the updated action for the next iteration is obtained by adding the delta action to the original action \( \bar{v}_0 + \delta \bar{v}_0 \). Note that since our internal parabolic model does not consider drag, the resulting trajectory from model prediction and the actual action and dynamics are significantly different, as shown in Figure 14(b). Yet, despite internal model inaccuracy, IRP can converge to the goal very quickly, shown in Figure 14(c).

To evaluate the effectiveness of our simplified IRP algorithm, we randomly select 25 goal positions within 80 m from the origin and test IRP with 50 drag coefficient values \( c \) ranging from 0 to 0.015. For comparison, we also evaluate a naive baseline method, named Model Prediction, where the action is directly computed by fitting a parabola through the origin and the goal. As expected, the average error of both methods is 0 when the drag coefficient is 0.0, in which case the internal dynamics model exactly equals the actual dynamics system. However, as shown in 14 (d), the error of baseline method Model Prediction increases as the internal model becomes more and more inaccurate, while the error of IRP remains close to 0.

9. Limitations and future work

The proposed system relies on several assumptions. First, the assumption of action repeatability can be hard to satisfy in safety-critical applications where a failed action might cause irreversible damage to the manipulated object. However, in these cases, IRP could still be used as part of the simulation training process in analogy to mental practice Clark (1960) to human motor learning. Second, we assume full observability of the manipuland throughout the trajectory, which makes direct application of this approach difficult in highly cluttered scenarios. Future works could explore perception techniques that leverage temporal cues to handle partial observability Xu et al. (2020). Third, our method only makes action adjustments at the end of each action execution. Therefore, disturbances in the system cannot be corrected responsively. Future works could consider using IRP inside a Model Predictive Control Camacho and Alba (2013) loop to increase control frequency.

10. 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. Experiments suggest that IRP can achieve high precision despite complex object dynamics and the inaccurate simulator used for training, provided that the robot’s actions are repeatable. We hope the proposed algorithm can spur more interest in better utilizing existing inaccurate physical simulators for bridging the sim2real gap.

Acknowledgements

We would like to thank Zhenjia Xu, Huy Ha, Dale McConachie, Naveen Kuppuswamy for their helpful feedback and fruitful discussions.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: 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.

ORCID iDs

Cheng Chi https://orcid.org/0000-0003-0319-0228
Benjamin Burchfiel https://orcid.org/0000-0001-7332-6712
Eric Cousineau https://orcid.org/0000-0002-5056-8046

References

Berenson D (2013) Manipulation of deformable objects without modeling and simulating deformation. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. Tokyo, Japan: IEEE, pp. 4525–4532. DOI: 10.1109/IROS.2013.6697007
Bristow D, Tharayil M and Alleyne A (2006) A survey of iterative learning control. IEEE Control Systems Magazine 26(3): 96–114. DOI: 10.1109/MCS.2006.1636313
Camacho EF and Alba CB (2013) Model Predictive Control. Springer science and business media. Available at: https://link.springer.com/book/10.1007/978-1-4471-3398-8
Xu Z, He Z, Wu J, et al. (2020) Learning 3D Dynamic Scene Representations for Robot Manipulation. *Conference on Robot Learning (CoRL)*.

Yamakawa Y, Namiki A and Ishikawa M (2011) 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*. Shanghai, China: IEEE, pp. 5486–5491. DOI:10.1109/ICRA.2011.5979606

Yoshida E, Ayusawa K, Ramirez-Alpizar IG, et al. (2015) Simulation-based optimal motion planning for deformable object. In: *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*. Lyon, France: IEEE. pp. 1–6. DOI:10.1109/ARSO.2015.7428219

Zeng A, Song S, Lee J, et al. (2019) Tossingbot: learning to throw arbitrary objects with residual physics.

Zhang H, Ichnowski J, Seita D, et al. (2021) Robots of the lost arc: self-supervised learning to dynamically manipulate fixed-endpoint cables. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi’an, China: IEEE, pp. 4560–4567.