Learning Deformable Object Manipulation Using Task-Level Iterative Learning Control

Krishna Suresh and Chris Atkeson
Carnegie Mellon University
Email: ksuresh2,cga@andrew.cmu.edu
https://flying-knots.github.io

***Abstract*—Dynamic manipulation of deformable objects is challenging for humans and robots because they have infinite degrees of freedom and exhibit underactuated dynamics. We introduce a Task-Level Iterative Learning Control method for dynamic manipulation of deformable objects. We demonstrate this method on a non-planar rope manipulation task called the flying knot. Using a single human demonstration and a simplified rope model, the method learns directly on hardware without reliance on large amounts of demonstration data or massive amounts of simulation. At each iteration, the algorithm constructs a local inverse model of the robot and rope by solving a quadratic program to propagate task-space errors into action updates. We evaluate performance across 7 different kinds of ropes, including chain, latex surgical tubing, and braided and twisted ropes, ranging in thicknesses of 7–25mm and densities of 0.013–0.5 kg/m. Learning achieves a 100% success rate within 10 trials on all ropes. Furthermore, the method can successfully transfer between most rope types in approximately 2–5 trials. https://flying-knots.github.io**

I. Introduction

Dynamic manipulation of deformable objects is a challenging domain for both robots and humans. Deformable objects such as ropes and cloth have many unactuated degrees of freedom, and are expensive to model accurately. We enable a robot to learn the dynamic task of tying a *flying knot* as seen in Fig. 1. This task is performed on a rope by executing a one-handed upward and twisting motion to form a loop and then arcing to strike near the end of the rope to flip it into the loop and completing a knot (Fig. 1).

We adapt Iterative Learning Control (ILC) to improve a command trajectory through real-world trials. ILC is a powerful tool for enabling a robot to improve its performance on hardware, as introduced by Arimoto et al. [5], but we find that typical ILC formulations fail in deformable-object manipulation. ILC uses an approximate system model to convert task errors from real trials into command corrections; however, when ILC is learning to track rope target motions, equal weighting of task errors causes learning to fail.

We leverage the underutilized Task-Level ILC [2] with two key features to enable success on deformable objects:

- **Critical point objective:** Instead of weighting trajectory errors equally throughout the task, we focus the robot's attention on the error in achieving a single rope configuration in the trajectory. This improves learning performance and enables the transfer of the task across large parameter variations.

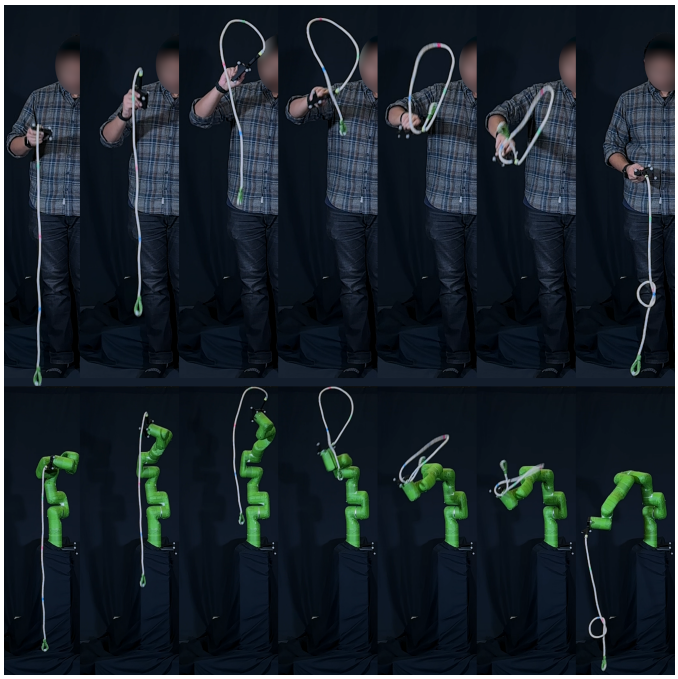

Fig. 1. Stages of a flying knot by both a human and a robot over 0.56s

- **Task-level learning:** Our approach corrects the trajectory of state variables of the manipulated object, which goes beyond the usual ILC approach of correcting the robot trajectory.

We demonstrate the effectiveness of Task-Level ILC for deformable object manipulation. We show that learning is adaptable to variations in rope dynamics, demonstration types, and the system model. Overall, the key contributions of this work include the following:

- We explore Task-Level ILC for deformable object manipulation, demonstrating that learning typically requires a small number of trials (< 10 trials) and transfer across different manipulated objects.
- We develop a new approach to ILC for robots, which focuses attention on *critical points* of the error history.

II. The Flying Knot Task

For this case study, we considered a variety of one dimensional object manipulations, including whipping, lassoing, throwing a rope to a target, fly casting, and attaching a rope to

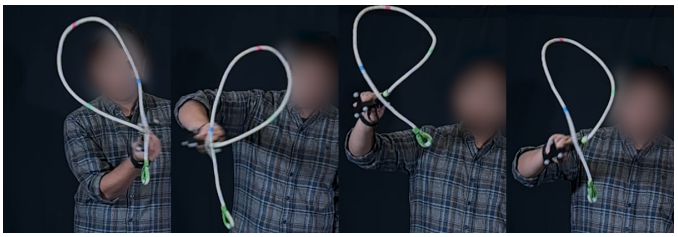

Fig. 2. **Critical point** for the flying knot across 4 demonstration variations. The shape of the rope at collision is used for the learning objective.

a distant object, such as a cleat or tree branch by manipulating one end. The flying knot task was chosen because it is impressive but achievable by humans and fast robots.

There are several types of flying knots. We focus on the *Overhand Knot*, in which one hand manipulates the end of a rope to tie an overhand knot, without changing the grip. As seen in Fig. 1, the flying knot consists of multiple stages. The rope starts hanging from the hand. The hand is moved upward and twisted to form a loop. The rope collides with itself, and the end of the rope is flipped through the loop. The knot tightens as the rope falls. Often, a weight is added to the end of the rope.

A. Flying Knot Objective

While a successful flying knot is a well-defined task, the measure of whether a failed attempt is closer or further away from success is unclear. There are many key moments in the execution of a flying knot: loop creation, rope shape at collision, rope end passing through the loop, etc. In instructional materials for the flying knot, instructors often refer to the *strike point* as a crucial step, as seen in Fig. 2. At least one rope self-collision occurs about halfway through task execution. While this moment in flying knot execution is not an exact predictor of task success, correctly matching this shape will likely lead to a knot.

We refer to this point in the rope motion as a *critical point*. We provide the robot with a single demonstration of the flying knot for an example of the rope state at the *critical point*. The flying knot task can be performed in several ways. The variations in demonstration share the same qualitative topological contact event, as visualized in Fig. 2, allowing the use of a single type of *critical point* for learning.

III. Related Work

A. Iterative Learning Control

Iterative learning control is a command-update algorithm in which the system iteratively executes a feed-forward command in the real world and maps task errors into feed-forward command corrections. ILC leverages a system's repeatability to accumulate corrections across trials. Model-based ILC maps task errors to command updates with an *inverse model* [4].

Classical Iterative Learning Control (ILC) studies how to improve performance on tasks that repeat over trials by updating a feedforward command based on errors. Early work formalized the idea of "bettering" robot operation via iterative error-based updates [5]. In robotics, ILC-style trajectory learning through practice was demonstrated on manipulators using explicit (possibly imperfect) linear models and approximate inversion [6, 4], and subsequent work analyzed convergence and design choices for robot manipulators and other repetitive systems [20, 23].

Optimization-based ILC or Norm-Optimal ILC formulations showed how to incorporate constraints and compute command corrections via an optimization objective. Schoellig et al. [28] demonstrated aggressive quadrotor trajectory tracking by leveraging convex programs to reduce errors, and Ratcliffe et al. [26] applies norm-optimal methods for real-time control of industrial gantry robots.

Event-based ILC methods focus learning on a subset of the task, such as the terminal states[12], point-to-point [10], time windows [32], and event triggers [22]. In general, placing learning attention at key moments of the task has worked in many domains, including non-robotics industrial domains such as CNC machine tools, wafer-stage motion systems, injection molding, induction motor drives, automotive braking/vehicle control, rapid thermal processing, and semi-batch chemical reactors [7].

Modern learning methods have recently been combined with ILC: using approximate simulators to propose local policy improvements [1], framing ILC through regret minimization [3], and explaining when ILC can outperform planning with a misspecified model [33]. Additional work introduces ILC-style model-gradient updates in deep off-policy reinforcement learning to improve sample efficiency [15].

In our work, we leverage optimization-based and event-based techniques to attend learning to the *critical points* of the task.

B. Deformable Object Manipulation

Quasi-static deformable object manipulation, such as the handling of rope, cable, and cloth, has been addressed in a number of works [27, 40] where many approaches focus on domains with strong perception and geometric priors [30] and imitation/representation learning for rope manipulation [25, 17, 19].

Dynamic deformable object manipulation focuses on faster tasks, such as whipping [24], cloth flinging [9, 38], and dynamic rope knotting [37]. Iterative Residual Policy proposes online action refinement via a learned residual dynamics model [11]. Related works study learning for whip targeting, combining structured motion primitives with online adaptation or reinforcement learning [36, 35].

Recent work has further explored dynamic cable and cloth behaviors with self-supervised data collection. Real2Sim2Real learning enables planar robot casting of cables by fitting simulators from physical rollouts [21], and subsequent systems learn to dynamically manipulate fixed-endpoint cables via arcing motions [41] or free-end cables in planar settings [34].

In Ha and Song [16], Chi et al. [11], and Zhang et al. [41], system models are learned with large-scale simulated data, but

our use of Model-Based ILC eliminates the need for large-scale simulated data collection.

In these works, commands are executed as feed-forward trajectories and demonstrate the repeatability of dynamic deformable object manipulation. Similarly, we leverage feedforward trajectories in the flying knot task.

IV. Method: Task-Level Iterative Learning Control

A. Iterative Learning Control

Iterative Learning Control (ILC) is a data-efficient learning method for repetitive tasks. ILC maps task error trajectories $\tilde{\mathbf{x}}(t)$ to command trajectory updates $\mathbf{\Delta u}(t)$ to iteratively refine a full sequence of commands. ILC can improve task performance beyond methods that generate commands solely from a system model, since it can correct for unmodeled dynamics. ILC is often applied to robot trajectory-tracking problems to minimize tracking errors caused by unmodeled system dynamics (e.g., friction, cogging torques, and gear backlash and play).

The command trajectory $\mathbf{u}(t)$ that ILC corrects is a function of task phase (or time) rather than a control policy, which is a function of state. A time-dependent feedforward command has N parameters, where N is the number of samples. A feedback control policy typically has several orders of magnitude more parameters.

Model-based ILC uses a system model $\mathcal{M}$ which maps commands to predictions of task performance. This system model may be inaccurate, but its gradient information accelerates learning, since knowledge of the command update direction need not be obtained from the real system. ILC inverts the system model, $\mathcal{M}^{-1}$, to map the real system's task errors into estimated command errors, which are subtracted from the current command. An overview of ILC is seen in Fig. 3. If this system model is perfect, then all errors would be removed in one step. Even when the system model is inaccurate, if the command update direction (gradient) has a component in the required command update direction, command updates accumulate and iteratively reduce task errors.

B. Task-Level Iterative Learning Control

ILC is often used for robot trajectory tracking. In trajectory tracking, the objective and the command are related by the robot dynamics, which are often modeled accurately. Trajectory-tracking ILC usually weights errors along the trajectory equally.

In Task-Level ILC, task errors are similarly mapped to feedforward robot command improvements. In this work, task errors are defined as deviations in the state variables of a deformable manipulated object. In addition, we explore the use of *critical point* objectives, which focus on the error at *critical points* along the trajectory.

Algorithm 1 summarizes Task-Level ILC.

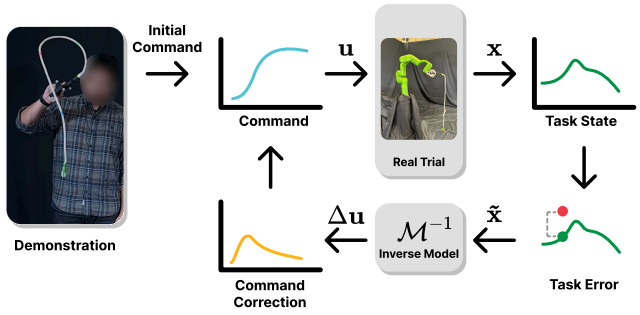

Fig. 3. **Task-Level Iterative Learning Control System:** A demonstration is converted to an initial command. The command trajectory $\mathbf{u}(t)$ is executed on the real system, and the resulting trajectory $\mathbf{x}(t)$ is measured. The task error $\tilde{\mathbf{x}}(t)$ at the *critical point* is mapped through the inverse model $\mathcal{M}^{-1}$ to command trajectory corrections $\mathbf{\Delta u}(t)$, which are applied to the current feedforward command, closing the learning loop.

Algorithm 1 Task-Level Iterative Learning Control (Task-Level ILC)

Inputs: initial command $\mathbf{u}_0(t)$; reference critical-point state $\mathbf{x}^{\text{demo}}(t_c)$; *critical point* t_c; inverse model $\mathcal{M}^{-1}$; K iterations
for $k = 0$ **to** K **do**
  $\mathbf{x}_k(t) \leftarrow \texttt{Trial}(\mathbf{u}_k(t))$
  Critical-point error $\tilde{\mathbf{x}}_k(t) \leftarrow \mathbf{x}_k(t_c) - \mathbf{x}^{\text{demo}}(t_c)$
  Command update $\Delta\mathbf{u}_k(t) \leftarrow \mathcal{M}^{-1}(\tilde{\mathbf{x}}_k(t))$
  Update $\mathbf{u}_{k+1}(t) \leftarrow \mathbf{u}_k(t) - \Delta\mathbf{u}_k(t)$
end for

C. Critical Point Objective

The *critical point* is defined as a key moment in the task at which all task error reduction is targeted. For the flying knot task, we select the state of the rope at collision time t_c in the demonstration as the *critical point* as described in Section II-A. Task-Level ILC attempts to generate a feedforward trajectory before the collision to match the rope state of the demonstration only at the moment of collision. Therefore, rope-tracking errors before and after the *critical point* are ignored as seen in Fig. 4

Eliminating task errors from the objective for times after contact also simplifies the modeling problem, as we only need to model free rope motion rather than the more complex contact physics between rope or finger and the end of the rope.

D. Command Parameterization

We parameterize the feed-forward robot command as 10 Bezier curves (7 joints + 3 base translation dimensions) with 8 knot points equally spaced in time: $\mathbf{u}(t) \in \mathbb{R}^{10\times 8}$. The base translation dimensions ensure translation invariance of the task objective, with constraints that prevent the robot base location from moving Eq. (3c). The fixed total execution time of the command T is defined by the total length of the demonstration hand motion. Given $\mathbf{u}(t)$ and T, a Bezier curve is computed using the Bezier function $\mathcal{B}$. This spline generation of the robot's desired trajectory command is described in more detail

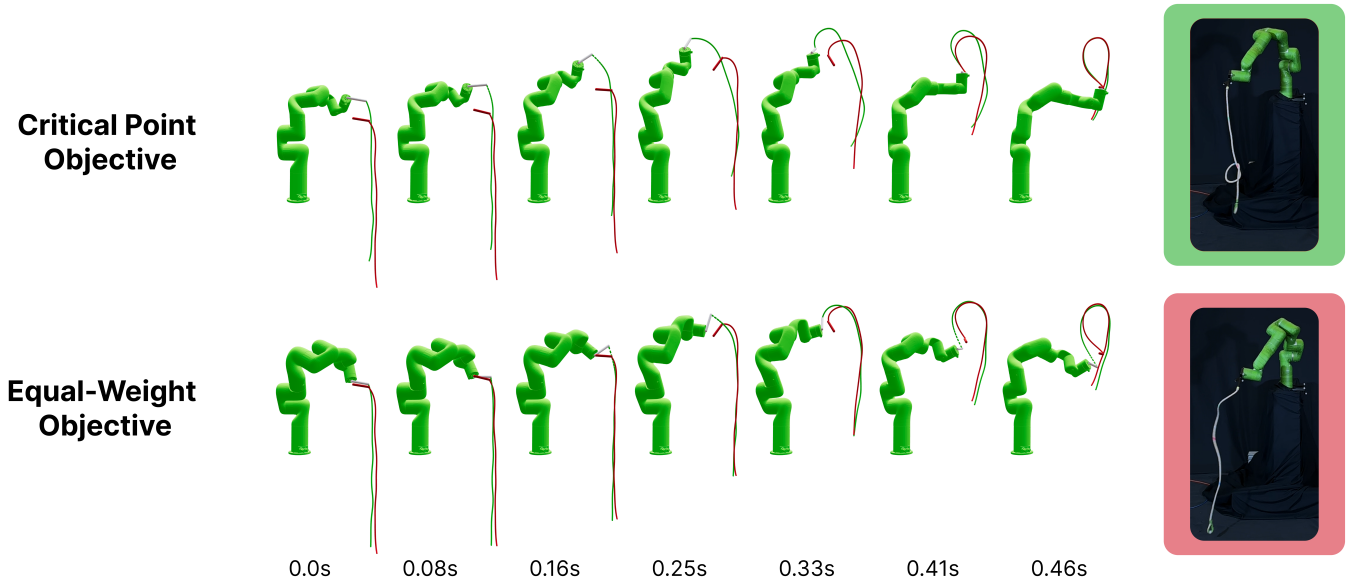

Fig. 4. **Critical point vs equal-weighted objective learned commands.** Each row is a real trial after 8 iterations with the corresponding objective. The green rope is the measured state from the real trial, and the red rope is the goal rope from the demonstration. The rope state at 0.46s is the *critical point*. The *critical point* objective trial results in a successful flying knot, while the equal-weighted objective trial results in a failure.

in Appendix Section D.

E. Robot Model

We model the robot as a kinematic chain parametrized by robot joint angles q. Because the robot's controller is driven by commanding desired robot joint angles, any desired trajectory $\mathbf{q}_d(t)$ that satisfies the robot's joint position, velocity, and acceleration limits is trivially modeled with the robot motion matching the command, $q_d = q$. The desired trajectory is executed by the control system provided by the robot manufacturer, which mostly consists of joint-level PD servos. Joint trajectories $\mathbf{q}(t)$ are mapped through the robot's forward kinematics $\mathcal{K}(\mathbf{q}(t))$ to determine the trajectory of the hand fingertip. While the real robot has more elements that can be modeled (e.g., link dynamics, actuation model), we find that this simple model is sufficient for ILC to improve the command. The robot model does not need to account for rope motion since the robot has joint actuators with a 100:1 gear ratio, so the rope's impact on the robot's dynamics is negligible.

F. Rope Model

We model the rope as a 3D serial chain of point masses m connected by fixed-distance constraints of length l. Each joint in the rope has a bending stiffness k and a damping coefficient b. Since ropes are approximately self-similar, we use a single m, k, b parameter for all but the last rope links and joints. The ropes we evaluate with have a weight attached to the end, which is approximated with a different end link mass m_e. Our approximate rope model has far fewer degrees of freedom than the true rope dynamics, with only $N = 11$ links, which has proven adequate in this work. The unit scaling of these parameters is with respect to m, which we set to 1, resulting in only 5 model parameters (k, b, m_e, l, N).

TABLE I
ROPE MODEL PARAMETERS

Parameter	Value
Stiffness	1×10^5
Damping	50
End Mass	5
Simulation Timestep	0.005
# Links	11
Link Length	0.1m

The unit scaling of stiffness, damping, and end mass is defined with respect to a link mass of 1.

The first point on the rope model is kinematically driven by the robot model's fingertip trajectory. We can define the rope dynamics model as a function f, which simulates the rope state given the initial rope configuration $\mathbf{z}_0$ and a position trajectory of the first link (full derivation found in Appendix Section C). This makes the overall system model, as seen in Fig. 5, the following:

$$\mathcal{M}(\mathbf{u}(t), \mathbf{z}_0) = f(\mathcal{K}(\mathcal{B}(\mathbf{u}(t), T)), \mathbf{z}_0) = \hat{\mathbf{x}}(t) \qquad (1)$$

where $\hat{\mathbf{x}}$ is the model prediction of the rope state. To simulate the rope dynamics, we implement a maximal coordinate variational integrator dynamics model from Brüdigam and Manchester [8].

G. Optimization-Based Inverse Model

Previous model-based ILC methods typically use an inverse model $\mathcal{M}^{-1}$ that equally weights all parts of the trajectory, which is well-suited for trajectory tracking but is insufficient for Task-Level ILC. Furthermore, task constraints such as robot joint dynamics limits are not typically handled by these

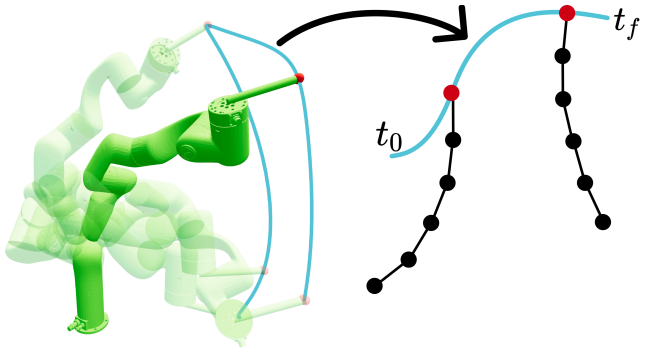

Fig. 5. **Left:** Kinematic robot model at various stages of a command. The opaque robot is at the point of contact. **Right:** Graphical representation of the point mass rope model. The robot's fingertip trajectory kinematically drives the first red dot. The remaining links are bound by distance constraints, and each joint has stiffness and damping. Together, the robot and rope models define our system dynamics model. All 3D visualization are created using Viser [39].

inverse models. Methods for an optimization-based inverse model were introduced for trajectory tracking in Schoellig et al. [28] and Schöllig and D'Andrea [29]. A Quadratic Program (QP) is formulated to minimize a quadratic task objective while satisfying linear constraints. The system model is imposed as a linearized dynamics constraint.

Based on the measured robot motion, we simulate the rope dynamics and linearize the system model $\mathcal{M}$ about the current simulated rope trajectory $\hat{\mathbf{x}}_k(t)$ and current command $\mathbf{u}_k(t)$:

$$\mathbf{M} = \frac{\partial \mathcal{M}}{\partial \mathbf{u}(t)}\bigg|_{(\hat{\mathbf{x}}_k(t), \mathbf{u}_k(t))} \tag{2}$$

We define the quadratic optimization objective $\mathbf{Q}$ to be a state-error-tracking cost applied only at the *critical point* t_c of the demonstration. A control cost $\mathbf{R}$ penalizes updates to the 7 arm joints at each knot point for all t.

This formulation allows explicit inclusion of linear robot dynamics constraints $\mathbf{J}_p, \mathbf{J}_v, \mathbf{J}_a, \mathbf{J}_\tau$ which are linearizations of the robot joint positions, velocities, accelerations, and torque about the current command applied for all t. q_0 is the initial configuration of the robot. $\mathcal{T}$ is a prediction of the torque at each joint via a rigid-body dynamics model.

The commanded trajectory continues to move after the *critical point*; however, there is no objective for the robot's behavior during this phase, as rope measurement is unavailable. We apply a linearized quadratic tracking cost to the fingertip trajectory at $t > t_c$ to match the demonstrator's follow-through motion: $\mathbf{Q_{ft}}$, which is fully derived in Appendix Section A2. In our QP, we solve for a command update $\mathbf{\Delta u}(t)$ and state prediction $\Delta \mathbf{x}(t)$ related by the linearized dynamics constraint from Eq. (2). Our full inverse model QP is formulated as follows:

$$\begin{aligned}
\mathcal{M}^{-1}(\tilde{\mathbf{x}}_k(t)) = \min_{\substack{\Delta\mathbf{u}(t)\\ \Delta\mathbf{x}(t)}} & \left\|\Delta\mathbf{x}(t_c) - \tilde{\mathbf{x}}_k(t_c)\right\|^2_{\mathbf{Q}} && \text{(3a)}\\
& + \sum_{t\in[t_c,T]} \|\Delta\mathbf{u}(t)\|^2_{\mathbf{Q_{ft}},\mathbf{b_{ft}}} \\
& + \sum_{t\in[0,T]} \|\Delta\mathbf{u}(t)\|^2_{\mathbf{R}} \\
\text{s.t.} \quad & \Delta\mathbf{x}(t) = \mathbf{M}\,\Delta\mathbf{u}(t) && \text{(3b)}\\
& \Delta\mathbf{u}_{\text{base}}(t) = 0 && \text{(3c)}\\
& \mathbf{q}_{\min} \le \mathbf{J}_p\Delta\mathbf{u}(t) + \mathcal{B}(\mathbf{u}(t)) \le \mathbf{q}_{\max} && \text{(3d)}\\
& \dot{\mathbf{q}}_{\min} \le \mathbf{J}_v\Delta\mathbf{u}(t) + \dot{\mathcal{B}}(\mathbf{u}(t)) \le \dot{\mathbf{q}}_{\max} && \text{(3e)}\\
& \ddot{\mathbf{q}}_{\min} \le \mathbf{J}_a\Delta\mathbf{u}(t) + \ddot{\mathcal{B}}(\mathbf{u}(t)) \le \ddot{\mathbf{q}}_{\max} && \text{(3f)}\\
& \tau_{\min} \le \mathbf{J}_\tau\Delta\mathbf{u}(t) + \mathcal{T}(\mathbf{u}(t)) \le \tau_{\max} && \text{(3g)}
\end{aligned}$$

A detailed list of all cost and constraint parameters can be found in Appendix Section A1.

We formulate this QP using the Drake optimization toolbox [31] and leverage the Clarabel Solver [14] to solve for the command update.

H. Demonstration

We provide the learning system with a single demonstration of the flying knot. For a given demonstration, we track the full hand trajectory and the trajectory of the rope up to collision. We use motion capture to track 11 markers in 3D, which correspond to the joints in the rope model. The collision point in the demonstration is manually annotated, then a search procedure is performed to find the start and end time of the demonstration as described in Appendix Section E. $\mathbf{h}(t)$ is the 3D pose trajectory of the hand, $\mathbf{x}^{\text{demo}}(t)$ is the rope trajectory, and t_c is the collision point.

I. Initial Guess

Given a demonstration of the flying knot, we initialize learning with $\mathbf{u}_0$ to track the hand fingertip trajectory $h(t)$. We solve the following trajectory optimization problem to minimize fingertip tracking error while satisfying the joint dynamics limits:

$$\begin{aligned}
\min_{\mathbf{u}(t)} \quad & \sum_{t_i\in[0,T]} \left\|\mathbf{e}_h(t_i)\right\|^2_{\mathbf{W}_h} + w_j \sum_{t_i\in[0,T]} \left\|\ddot{\mathbf{q}}(t_i)\right\|^2 && \text{(4)}\\
\text{s.t.} \quad & \mathbf{q}(t) = \mathcal{B}(\mathbf{u}(t), T)\\
& \mathbf{q}_{\min} \le \mathbf{q}(t) \le \mathbf{q}_{\max}\\
& \dot{\mathbf{q}}_{\min} \le \dot{\mathbf{q}}(t) \le \dot{\mathbf{q}}_{\max}\\
& \ddot{\mathbf{q}}_{\min} \le \ddot{\mathbf{q}}(t) \le \ddot{\mathbf{q}}_{\max}\\
& \dot{\mathbf{q}}(0) = \dot{\mathbf{q}}(T) = \mathbf{0}\\
& z_{\text{tip}}(\mathbf{q}(0)) \ge z_{\min}
\end{aligned}$$

A detailed explanation of the hand-tracking objective $\mathbf{e}_h$, weighting $\mathbf{W}_h$, and constraints is provided in Section F. We then formulate this trajectory optimization problem using the Drake toolbox [31] and solve using the SNOPT nonlinear solver [13].

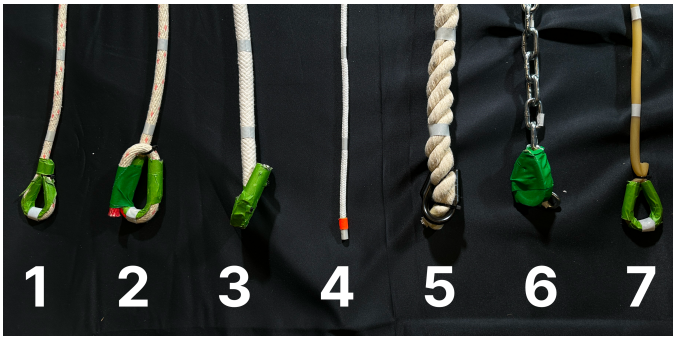

Fig. 6. 7 different rope types used for evaluation, including 4 braided ropes of various thickness and mechanical properties, a large twisted rope, a chain, and latex surgical tubing (very elastic); ranging in thicknesses of 7–25 mm and densities of 0.013–0.5 kg/m. Each rope has a mass affixed to aid in the formation of the knot.

V. Evaluation

We evaluate the learning algorithm's performance across 7 rope types and provide 4 variations of the flying knot demonstration. We define a successful flying knot as achieving a topological knot in a rope hanging down at the end of motion (e.g. a knot being tied then untied during motion is a failure). Given a flying-knot demonstration, we compare the success of the Task-Level ILC system against two common approaches: direct tracking of the human demonstrator (hand motion) and ILC learning of task progression (rope motion). The resulting commands are tested for robustness with 40 trials. We then evaluate the transfer of a learned command across rope types and, finally, assess the system's learning robustness with respect to model parameters. Any successful command is 100% successful, but continued learning can degrade commands, therefore learning should stop at the first success.

A. Experiment Setup

We conduct flying knot experiments using the xArm 7 robot arm. Both command execution and robot motion measurement occur at 250Hz. Commands that exceed the robot's joint dynamics limits (position, velocity, acceleration, torque) result in mid-motion faults and task failure.

Each rope is equipped with 11 retroreflective tape markers evenly spaced along its length. Markers are tracked with a Vicon Motion Capture system at 200Hz. Tracking of rope markers is often obscured once the rope contacts the hand. Hand position is also tracked using 4 motion-capture markers. All ropes are 1.1 meters, matching the same length as in the demonstration. The rope is always started in a static state hanging down from the fingertip, though some types of ropes have curved resting states.

B. Demonstration Hand Tracking

For each demonstration type, we apply our initial-guess generation procedure, as described in Section IV-I, to assess how well the robot performs the task solely by following the demonstrated hand motion. In every rope-demonstration type for rope 1, the robot is unable to tie the flying knot when attempting to follow the demonstration trajectory. However, for certain ropes, such as 4 and 7, following the demonstration nearly achieves a flying knot.

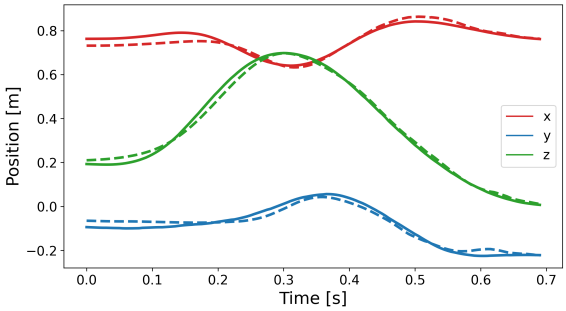

Fig. 7. **Initial command fingertip trajectory**: positions of the demonstration fingertip trajectory are dashed, and the robot commanded initial attempt to track the demonstration are solid. The robot fails to track the demonstration motion due to kinematic and dynamic constraints.

Generating a command to exactly track the demonstrated hand motion is not possible due to the limits of the robot's joint ranges, maximum velocities, accelerations, and torques, as well as the differences between human and robot morphology. An example demonstrated fingertip trajectory and the corresponding initial attempt by the robot are visualized in Fig. 7. While the robot can capture the demonstrator's overall motion, it cannot match it exactly, resulting in failures when attempting to tie the flying knot.

C. Weighted Error Objectives

We compare the performance of the learning algorithm under two objectives: the *critical point* objective (ours) and the equal-weighting objective. The *critical point* objective, which focuses on errors at one point in the trajectory, is described in Section IV-C. The equal-weighting objective weighs rope tracking errors equally along the pre-collision trajectory and is commonly found in trajectory tracking and behavior cloning.

As seen in Fig. 9, the critical-point objective aligns the rope state at a single point, at the collision, allowing for different rope motion beforehand, and results in the flying knot success.

Overall, we find that the *critical point* objective is crucial to task success. The equal weighting objective increases errors at the critical point, compared to the critical point objective, to reduce errors at earlier parts of the trajectory that matter less for task success. This error different at the *critical point* for equally-weighted learning causes failure.

D. Learned Command Success Rate

Once a successful trajectory is learned, the robot has a 100% success rate with the resulting learned command. We evaluated this success rate by repeating 40 trials of a learned command and observing no failures. Executions of the successful command are performed immediately after learning, and longer-term changes to the system dynamics, such as rope wear or breaking in (changes to stiffness and damping), and robot motor temperature, might require additional learning iterations.

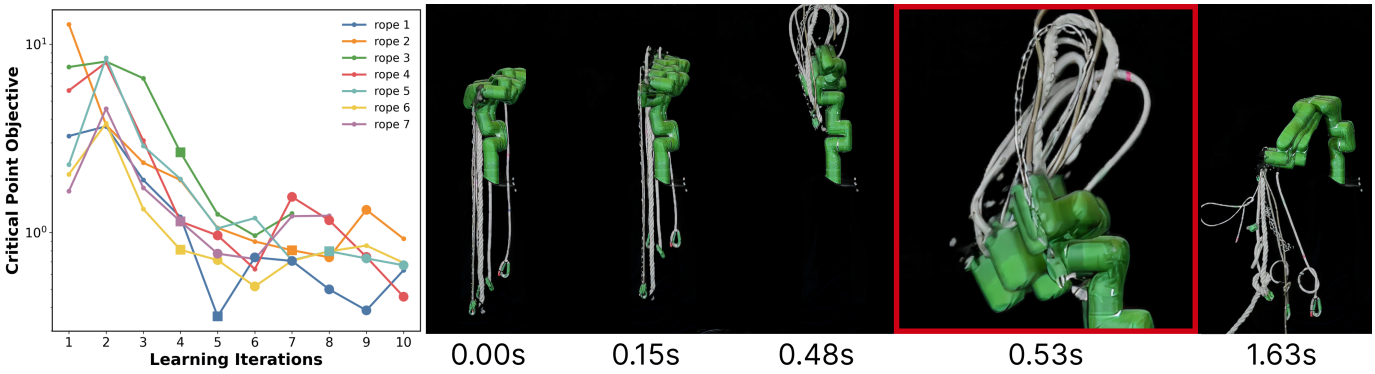

Fig. 8. **Left:** *Critical point* objective for 7 rope types over 10 iterations (rope 3 and 7 end early due to marker tracking failures). Squares represent the first successful flying knot, and every large solid dot represents a subsequent successful knot. **Right:** Real successful flying knot execution on 7 rope types overlayed. Red highlighted frame is the *critical point*.

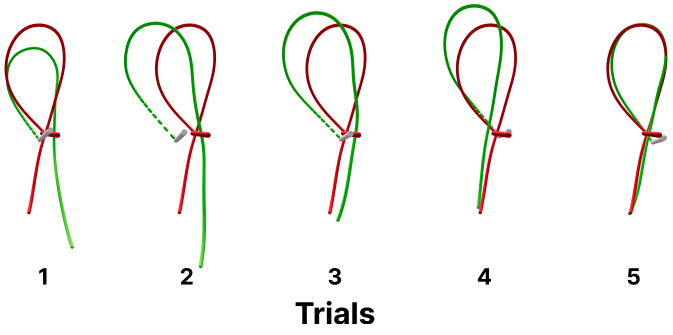

Fig. 9. **Rope configuration at the *critical point* over learning iterations:** Real rope (green) and goal rope(red) states for 5 iterations of Task-Level ILC where trial 5 resulted in a flying knot.

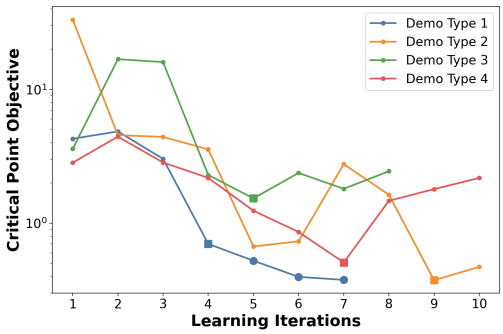

Fig. 10. Learning cost over trials for 4 demonstration types. Large solid dots represent times when the flying knot succeeded, and squares are the first successful command. Learning on Demo Types 1 and 3 is truncated due to rope tracking failures.

E. Learning Across Rope Types

We evaluate the learning of the flying knot across 7 different ropes (as shown in Fig. 6) to demonstrate the robustness of Task-Level ILC to varying system dynamics. A detailed list of parameters of each rope can be found in Appendix Section B. In general, ropes span a thickness of 7-25mm, a density of 0.013-0.5 kg/m, and are composed of materials such as cotton, polyester, steel, and latex. All ropes have a mass fixed to the end with masses selected to allow the human demonstrator to execute the flying knot.

Task-Level ILC successfully learns the flying knot on all 7 rope types within 10 trials. The successful flying knot commands are shown in Fig. 8. Each command starts the rope in a different initial state and performs the upward twisting motion at different speeds. Once the critical moment is reached, all ropes form the same loop shape and allow the knot to be formed. The differences in learned commands show that variation in the rope dynamics requires variation in the command. As seen in Fig. 8, the learning algorithm succeeds in fewer trials for some rope types (3, 6, 7) and requires more iterations for others (2, 5), but does not exceed 10 trials on any rope. Overall, the learning process reduces the cost of the *critical point* objective across trials; however, because we are not performing a line search on the real-system objective, the cost can increase after a poor command update, as with rope 4. Furthermore, repeated updates after success can lead to failure, as in trial 9 of rope 2. It is unclear whether additional learning after success leads to convergence to a successful command or to failure, as both outcomes occasionally occur.

F. Learning Across Demonstration Variation

For rope 1, we test whether variations in the demonstration impact learning with the 4 different flying knot demonstration types seen in Fig. 2. The total demonstration time lengths vary, ranging from 0.69s to 1.04s. The learning algorithm learns the flying knot in all 4 cases. Fig. 10 shows the learning progression, with Demo Type 2 requiring the most trials.

G. Transfer of Learned Command

Given a demonstration on rope 1 and a successful command on all 7 ropes, we attempt to transfer the learned commands between rope types. The successful command on rope A initiates learning on rope B and has a maximum of 10 trials to improve. A grid of the number of trials required to adapt the command to rope B is shown in Fig. 11. For the majority of transfers, the number of trials is larger than 1, showing that the dynamics of the ropes are different enough to require different commands. For rope 7 (9mm latex), no additional learning is

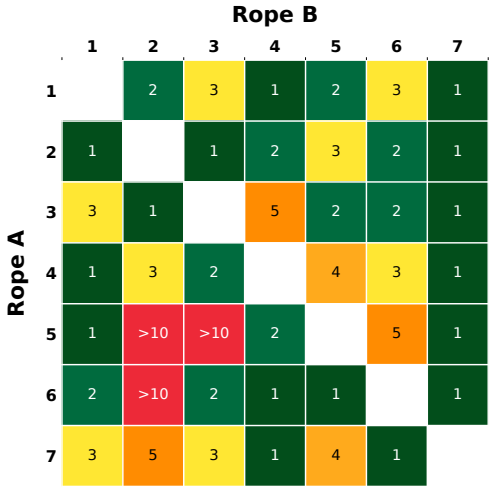

Fig. 11. Number of trials to transfer a successful command from rope A to rope B. Red cells are for transfer that did not succeed within 10 trials.

required, indicating that rope 7 is more robust to command variations. Additionally, learning fails when commands from ropes 5 and 6 are transferred to ropes 2 and 3. In all three cases, learning iteratively reduces the objective but requires larger adjustments and cannot fully correct the command within the 10 allotted trials.

H. Sensitivity to Model Parameters

ILC can make command corrections in a sample-efficient manner by leveraging a system model. We evaluate the effects of changes to the model parameters by learning on rope 1 with a single demonstration, while varying the system model stiffness k and end mass m_e. These parameters were selected for their dominant impact on the model dynamics. In Table II, we evaluate learning performance with different rope model parameters (bold represents the default parameters used across all experiments). For order-of-magnitude changes in the model parameters, learning can still succeed. There are cases in which learning fails, such as when stiffness is too low, in which a single command update places the trials in an unrecoverable execution regime. Similarly, for runs with low end-mass values, updates after a successful knot quickly drive the command away from success.

VI. Discussion

A. Learning on the Real System

Task-Level ILC enables task learning on real systems in a sample-efficient manner, using approximate system models to convert real-system errors into command updates. In simulation-based learning methods, such as policy learning on a simulated system model and Sim2Real transfer with domain randomization, issues often arise due to gaps between the system model and the real system. Techniques for model learning attempt to overcome this by adapting the model used for policy optimization with real data, but suffer from limitations of the modeling structure. Domain randomization builds robustness into the policy by optimizing over a range of models. Policy optimization with domain randomization is a worst-case robust control design procedure, where performance is degraded in all situations relative to the performance with an accurate model for that situation.

Learning directly on the real system eliminates these issues by allowing the real system to be captured in the learning process. The models used for learning do not need to accurately predict future system states, as illustrated by the large discrepancies between the model predictions and the real rope, yet learning still occurs. These approximate models still provide sufficiently accurate gradient information. As shown in Section V-H, the learning process is robust to large variations in the parameters, eliminating challenges in accurate model parameter estimation and domain-randomization.

TABLE II
Effect of Model Parameters on Learning Performance.

Stiffness	End Mass	Trials Until Success
$\mathbf{1 \times 10^5}$	**5**	**4**
1×10^5	0.005	6
1×10^5	0.05	8
1×10^2	5	> 10
1×10^3	5	4
1×10^4	5	4

B. Critical Points for Task-Level Learning

In evaluating the learning system on the flying knot task, we demonstrate that a *critical point* enables learning, robustness, and transfer. In Section V-C, we show that an equally weighted objective failed to achieve the task because errors at the *critical point* are larger than with a *critical point objective* (as visualized in Fig. 4).

Would critical point objectives improve Behavior Cloning (BC)? BC achieves tasks not by iteratively improving performance but instead by capturing the task distribution with a range of demonstrations. Direct teleoperation of the robot is often required for BC to ensure real system dynamics are captured in demonstrations. In BC, task transfer often struggles across domains and robots. Hu et al. [18] addresses some of these challenges by leveraging an instructor to guide learning toward subgoals. One can view the highlighted subgoals as critical points.

C. Command Learning

For tying a flying knot, we are learning a feedforward command. Repeated execution of commands results in reproducible behavior. By learning a feedforward command, the learning problem is reduced to searching for commands as a function of time ($\mathbf{u}(t)$) rather than a general feedback policy ($\mathbf{u}(\mathbf{x})$). This reduction in the dimensionality of the learning problem comes at a cost of the approach being unable to handle unstable systems or situations with large random perturbations. ILC can be applied to unstable systems that have been stabilized, as in Schoellig et al. [28].

Command learning can also suffer from increased high-frequency corrections, since mechanical systems with energy loss are inherently low-pass filters; inverse models of these

systems become high-pass filters. Therefore, learning amplifies high-frequency modeling inaccuracies. Since the robot can only execute smooth, continuous motions, we chose to parametrize the command trajectory using a small number of degrees of freedom. The drawback of this approach is that we cannot correct true high-frequency command errors.

Manipulation of deformable objects is well-suited for command learning, as errors are often repeatable, and with energy loss, the system dynamics are usually stable.

D. Limitations and Future Work

While our learning system is robust to changes in rope dynamics, different demonstrations, and model parameters, four main failure modes can degrade learning. First, because the system model does not account for collisions, rope collisions with the robot body or the rope handle can cause the inverse model to apply unrecoverable command corrections. Second, the selected *critical point* of the rope collision is sometimes insufficient for fully completing the flying knot, as the rope may tie the knot and execute a follow-through motion, which unties the knot before the rope settles. Similarly, lighter ropes can get caught on the finger surface and be unable to tighten the knot. Third, marker tracking is only accurate up to rope collision, and since we set the *critical point* to be at a fixed time point from the start of motion, if the rope contact happens early, rope state estimation may fail. Lastly, learning can diverge and get stuck in a local minima. Learning divergence occurs more frequently when task errors are too large at the *critical point*, and the model predictions are sufficiently inaccurate.

Avenues for future research include autonomous selection of *critical points* from demonstrations or instructional materials, identification of model structures for a given task, and learning from lower-fidelity demonstrations/measurements. Specifying the *critical point* as the rope collision point is a crucial input to our learning approach, and the autonomous identification of these *critical points* will enable a robot to learn efficiently. Potential selection criteria for *critical points* could include: human instruction, changes in contact state; dynamical extrema (position, velocity, acceleration, jerk, curvature, etc.); closest/furthest approach points; dynamic bifurcations/branch points; and dynamical funneling points.

VII. Conclusion

We demonstrate the success of Task-Level Iterative Learning Control for deformable-object manipulation by learning to tie various flying knots in less than 10 trials. We introduce the notion of a critical-point objective and demonstrate task-level learning of unactuated system dynamics. Given a single demonstration of the task, our learning system learns to tie a flying knot across 7 different rope types and 4 demonstration variations. We show that a learned command achieves a 100% success rate and transfers across most rope types in 2-5 trials.

References

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 1–8, 2006.

[2] E.W. Aboaf, C.G. Atkeson, and D.J. Reinkensmeyer. Task-level robot learning. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1309–1310 vol.2, 1988. doi: 10.1109/ROBOT.1988.12245.

[3] Naman Agarwal, Elad Hazan, Anirudha Majumdar, and Karan Singh. A regret minimization approach to iterative learning control. In *Proceedings of the 38th International Conference on Machine Learning*, pages 100–109. PMLR. URL https://proceedings.mlr.press/v139/agarwal21b.html.

[4] Chae H. An, Christopher G. Atkeson, and John Hollerbach. *Model-Based Control of a Robot Manipulator*. Artificial Intelligence Series. MIT Press. ISBN 978-0-262-51157-5.

[5] Suguru Arimoto, Sadao Kawamura, and Fumio Miyazaki. Bettering operation of robots by learning. *Journal of Robotic Systems*, 1(2):123–140, 1984.

[6] C. Atkeson and J. McIntyre. Robot trajectory learning through practice. In *1986 IEEE International Conference on Robotics and Automation Proceedings*, volume 3, pages 1737–1742. URL https://ieeexplore.ieee.org/document/1087423.

[7] 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.

[8] Jan Brüdigam and Zachary Manchester. Linear-time variational integrators in maximal coordinates. In Steven M. LaValle, Ming Lin, Timo Ojala, Dylan Shell, and Jingjin Yu, editors, *Algorithmic Foundations of Robotics XIV*, volume 17, pages 194–209. Springer International Publishing. doi: 10.1007/978-3-030-66723-8_12. URL http://link.springer.com/10.1007/978-3-030-66723-8_12. Series Title: Springer Proceedings in Advanced Robotics.

[9] Lawrence Yunliang Chen, Huang Huang, Ellen Novoseller, Daniel Seita, Jeffrey Ichnowski, Michael Laskey, Richard Cheng, Thomas Kollar, and Ken Goldberg. Efficiently learning single-arm fling motions to smooth garments. URL http://arxiv.org/abs/2206.08921.

[10] Yiyang Chen, Bing Chu, and Christopher T. Freeman. Point-to-point iterative learning control with optimal tracking time allocation. *IEEE Transactions on Control Systems Technology*, 26(5):1685–1698, 2018. doi: 10.1109/TCST.2017.2735358.

[11] Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Iterative residual policy: for goal-conditioned dynamic manipulation of deformable objects. URL http://arxiv.org/abs/2203.00663.

[12] Ronghu Chi, Danwei Wang, Frank L Lewis, Zhongsheng Hou, and Shangtai Jin. Adaptive terminal ilc for iteration-varying target points. *Asian Journal of Control*, 17(3):952–962, 2015.

[13] Philip E. Gill, Walter Murray, and Michael A. Saunders.

Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002. doi: 10.1137/S1052623499350013. URL https://doi.org/10.1137/S1052623499350013.

[14] Paul J. Goulart and Yuwen Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives, 2024. URL https://arxiv.org/abs/2405.12762.

[15] Swaminathan Gurumurthy, J. Zico Kolter, and Zachary Manchester. Deep off-policy iterative learning control. In *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, pages 639–652. PMLR. URL https://proceedings.mlr.press/v211/gurumurthy23b.html.

[16] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding, 2021. URL https://arxiv.org/abs/2105.03655.

[17] Eric Hannus, Tran Nguyen Le, David Blanco-Mulero, and Ville Kyrki. Dynamic manipulation of deformable objects using imitation learning with adaptation to hardware constraints. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12655–12662. IEEE, 2024.

[18] Zheyuan Hu, Robyn Wu, Naveen Enock, Jasmine Li, Riya Kadakia, Zackory Erickson, and Aviral Kumar. RaC: Robot learning for long-horizon tasks by scaling recovery and correction. URL http://arxiv.org/abs/2509.07953.

[19] Zixuan Huang, Xingyu Lin, and David Held. Self-supervised cloth reconstruction via action-conditioned cloth tracking. In *IEEE International Conference on Robotics and Automation (ICRA), 2023*, 2023.

[20] Tae-Yong Kuc, Kwanghee Nam, and J.S. Lee. An iterative learning control of robot manipulators. *IEEE Transactions on Robotics and Automation*, 7(6):835–842, 1991. doi: 10.1109/70.105392.

[21] Vincent Lim, Huang Huang, Lawrence Yunliang Chen, Jonathan Wang, Jeffrey Ichnowski, Daniel Seita, Michael Laskey, and Ken Goldberg. Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8282–8289. doi: 10.1109/ICRA46639.2022.9811651. URL https://ieeexplore.ieee.org/document/9811651.

[22] Na Lin, Ronghu Chi, Biao Huang, and Zhongsheng Hou. Event-triggered nonlinear iterative learning control. *IEEE Transactions on Neural Networks and Learning Systems*, 32(11):5118–5128, 2020.

[23] Kevin L Moore, Mohammed Dahleh, and SP Bhattacharyya. Iterative learning control: A survey and new results. *Journal of Robotic Systems*, 9(5):563–594, 1992.

[24] 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 Biomechatronics (BioRob)*, pages 685–691, Nov 2020. doi: 10.1109/BioRob49111.2020.9224399.

[25] 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.

[26] James D. Ratcliffe, Paul L. Lewin, Eric Rogers, Jari J. Hatonen, and David H. Owens. Norm-optimal iterative learning control applied to gantry robots for automation applications. *IEEE Transactions on Robotics*, 22(6):1303–1307, 2006. doi: 10.1109/TRO.2006.882927.

[27] Gautam Salhotra, I-Chun Arthur Liu, Marcus Dominguez-Kuhne, and Gaurav S. Sukhatme. Learning deformable object manipulation from expert demonstrations. *IEEE Robotics and Automation Letters*, 7(4):8775–8782, 2022. doi: 10.1109/LRA.2022.3187843.

[28] Angela P. Schoellig, Fabian L. Mueller, and Raffaello D'Andrea. Optimization-based iterative learning for precise quadrocopter trajectory tracking. 33(1):103–127. doi: 10.1007/s10514-012-9283-2. URL https://doi.org/10.1007/s10514-012-9283-2.

[29] Angela Schöllig and Raffaello D'Andrea. Optimization-based iterative learning control for trajectory tracking. In *2009 European Control Conference (ECC)*, pages 1505–1510, 2009. doi: 10.23919/ECC.2009.7074619.

[30] Te Tang, Changhao Wang, and Masayoshi Tomizuka. A framework for manipulating deformable linear objects by coherent point drift. *IEEE Robotics and Automation Letters*, 3(4):3426–3433, 2018. doi: 10.1109/LRA.2018.2852770. URL https://ieeexplore.ieee.org/document/8403315.

[31] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

[32] C.L. van Oosten, O.H. Bosgra, and B.G. Dijkstra. Reducing residual vibrations through iterative learning control, with application to a wafer stage. In *Proceedings of the 2004 American Control Conference*, volume 6, pages 5150–5155 vol.6, 2004. doi: 10.23919/ACC.2004.1384669.

[33] Anirudh Vemula, Wen Sun, Maxim Likhachev, and J. Andrew Bagnell. On the effectiveness of iterative learning control. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, pages 47–58. PMLR. URL https://proceedings.mlr.press/v168/vemula22a.html.

[34] Jonathan Wang, Huang Huang, Vincent Lim, Harry Zhang, Jeffrey Ichnowski, Daniel Seita, Yunliang Chen, and Ken Goldberg. Self-supervised learning of dynamic planar manipulation of free-end cables. URL http://arxiv.org/abs/2405.09581.

[35] Junyi Wang and Xiaofeng Xiong. A learning-based control framework for human-like whip targeting. In *2024 10th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*, pages 550–555. doi: 10.1109/BioRob60516.2024.10719935. URL https://ieeexplore.ieee.org/document/10719935/. ISSN: 2155-1782.

[36] Xiaofeng Xiong, Moses C. Nah, Aleksei Krotov, and Dagmar Sternad. Online impedance adaptation facilitates manipulating a whip. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9297–9302. doi: 10.1109/IROS51168.2021.9636663. URL https://ieeexplore.ieee.org/document/9636663.

[37] Yuji Yamakawa, Akio Namiki, and Masatoshi Ishikawa. Motion planning for dynamic knotting of a flexible rope with a high-speed robot arm. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 49–54, 2010. doi: 10.1109/IROS.2010.5651168.

[38] 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.

[39] Brent Yi, Chung Min Kim, Justin Kerr, Gina Wu, Rebecca Feng, Anthony Zhang, Jonas Kulhanek, Hongsuk Choi, Yi Ma, Matthew Tancik, and Angjoo Kanazawa. Viser: Imperative, web-based 3d visualization in python. *arXiv preprint arXiv:2507.22885*, 2025.

[40] Hang Yin, Anastasia Varava, and Danica Kragic. Modeling, learning, perception, and control methods for deformable object manipulation. *Science Robotics*, 6(54):eabd8803, 2021. doi: 10.1126/scirobotics.abd8803. URL https://www.science.org/doi/abs/10.1126/scirobotics.abd8803.

[41] 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. URL http://arxiv.org/abs/2011.04840.

TABLE III
INVERSE MODEL PARAMETERS

Parameter	Value
w_{control}	0.5
$w_{\text{critical pos}}$	25
$w_{\text{critical vel}}$	0.00375
w_{pc}	100
w_{vc}	0.1
w_{Rc}	5
w_{pft}	1
w_{vft}	0.1
w_{Rft}	0.1
$w_{\text{ft velocity}}$	0.5
$\mathbf{q}_{\min}$	$[\,-6.28, -1.8, -6.28, -0.19, -6.28, -1.69, -6.28\,]$
$\mathbf{q}_{\max}$	$[\,6.28, 1.9, 6.28, 3.92, 6.28, 3.14, 6.28\,]$
$\dot{\mathbf{q}}_{\min}$	$-[3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14]$
$\dot{\mathbf{q}}_{\max}$	$[3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14]$
$\ddot{\mathbf{q}}_{\min}$	$-[100, 100, 100, 100, 100, 100, 100]$
$\ddot{\mathbf{q}}_{\max}$	$[100, 100, 100, 100, 100, 100, 100]$
$\tau_{\min}$	$-[130, 130, 40, 40, 40, 20, 20]$
$\tau_{\max}$	$[130, 130, 40, 40, 40, 20, 20]$

APPENDIX

A. Inverse Model Additional Details

1) Inverse Model Parameters: We use the shorthand $\|\mathbf{a}\|^2_{\mathbf{W}} := \mathbf{a}^\top \mathbf{W}\mathbf{a}$. The critical-point objective weights the rope-marker position and velocity errors at t_c with a diagonal matrix

$$\mathbf{Q} := \operatorname{diag}(w_{\text{critical pos}}\mathbf{I}_{3N},\ w_{\text{critical vel}}\mathbf{I}_{3N})\,,$$

where N is the number of rope markers (links). In general, w is a diagonal cost element for a cost matrix. The control-update regularizer is $\|\mathbf{\Delta u}(t)\|^2_{\mathbf{R}}$ with a diagonal $\mathbf{R}$ applying w_{control} to the 7 arm-joint update variables. For the follow-through, we penalize the end-effector tracking error in Section F with a time-varying diagonal weight matrix

$$\mathbf{W}(t) := \operatorname{diag}\big(w_p(t)\mathbf{I}_3,\ w_R(t)\mathbf{I}_3,\ w_v(t)\mathbf{I}_3\big),$$

using $(w_p, w_R, w_v) = (w_{pc}, w_{Rc}, w_{vc})$ at $t = t_c$ and $(w_p, w_R, w_v) = (w_{pft}, w_{Rft}, w_{vft})$ for $t > t_c$.

The parameters $\mathbf{q}$ corresponds to the seven robot joint angles with minimum and maximum constratins emposed on the angle, velocity and acceleration. τ is the predicted joint torque as given by the inverse dynamics of a full robot model. All values used in our experiments are listed in Table III.

2) QP Hand Tracking Objective: For $t \in [t_c, T]$, we encourage the robot to match the demonstrator's follow-through motion by penalizing the end-effector tracking error. Let $\mathbf{e}_{ft}(t; \mathbf{u}(t))$ denote the end-effector error vector defined in Section F. This error depends nonlinearly on the command through the Bezier spline and forward kinematics. We linearize $\mathbf{e}_{ft}$ about the current command $\mathbf{u}_k(t)$ with respect to a small update $\mathbf{\Delta u}(t)$:

$$\mathbf{e}_{ft}\big(t; \mathbf{u}_k(t) + \mathbf{\Delta u}(t)\big) \approx \mathbf{e}_{ft}\big(t; \mathbf{u}_k(t)\big) + \mathbf{J}_u(t)\mathbf{\Delta u}(t), \quad (5)$$

$$\mathbf{J}_u(t) := \left.\frac{\partial \mathbf{e}_{ft}(t; \mathbf{u}(t))}{\partial \mathbf{u}(t)}\right|_{\mathbf{u}(t)=\mathbf{u}_k(t)}. \quad (6)$$

For notational simplicity, we write $\mathbf{e}_{ft}(t) := \mathbf{e}_{ft}(t; \mathbf{u}_k(t))$. We apply a weighted least-squares cost

$$\big\|\mathbf{e}_{ft}\big(t; \mathbf{u}_k(t) + \mathbf{\Delta u}(t)\big)\big\|^2_{\mathbf{W}(t)} \approx \big\|\mathbf{e}_{ft}(t) + \mathbf{J}_u(t)\mathbf{\Delta u}(t)\big\|^2_{\mathbf{W}(t)}. \quad (7)$$

Expanding and dropping the constant term $\|\mathbf{e}_{ft}(t)\|^2_{\mathbf{W}(t)}$ yields a quadratic function of the command update:

$$\|\mathbf{\Delta u}(t)\|^2_{\mathbf{Q_{ft}}(t),\mathbf{b_{ft}}(t)} := \mathbf{\Delta u}(t)^\top \mathbf{Q_{ft}}(t)\mathbf{\Delta u}(t) + \mathbf{b_{ft}}(t)^\top \mathbf{\Delta u}(t), \quad (8)$$

$$\mathbf{Q_{ft}}(t) = \mathbf{J}_u(t)^\top \mathbf{W}(t)\mathbf{J}_u(t), \quad (9)$$

$$\mathbf{b_{ft}}(t) = 2\,\mathbf{J}_u(t)^\top \mathbf{W}(t)\mathbf{e}_{ft}(t). \quad (10)$$

B. Rope Parameters

Rope types were selected to evaluate the algorithm robustness to diameter, material, and density. Weights were chosen to enable human demonstrator to perform the demonstration successfully. A full list of parameters for each rope type used in evaluation can be found in Table IV.

C. Rope Dynamics Model

We model the rope as a serial chain of N point masses in maximal coordinates and simulate it with the first-order variational integrator described by Brüdigam and Manchester [8]. In maximal coordinate dynamics formulations, each link of the rigid body mechanism is tracked in a global frame. Interactions between links such as joints or contacts are represented explicitly in an optimization problem solved at each timestep. Similarly, in our rope model, we do not track the joint angles between links but instead only track the global position and velocity of each link, then impose constraints to enforce the serial structure of the rope. Let $\mathbf{p}_k^{(i)} \in \mathbb{R}^3$ and $\mathbf{v}_k^{(i)} \in \mathbb{R}^3$ denote the position and velocity of link $i \in \{1, \dots, N\}$ at discrete timestep k. We stack all link positions and velocities as

$$\mathbf{p}_k := \begin{bmatrix} \mathbf{p}_k^{(1)\top} & \cdots & \mathbf{p}_k^{(N)\top} \end{bmatrix}^\top \in \mathbb{R}^{3N},$$

$$\mathbf{v}_k := \begin{bmatrix} \mathbf{v}_k^{(1)\top} & \cdots & \mathbf{v}_k^{(N)\top} \end{bmatrix}^\top \in \mathbb{R}^{3N},$$

and define the rope state (used throughout the paper) as $\mathbf{x}_k := \begin{bmatrix} \mathbf{p}_k^\top & \mathbf{v}_k^\top \end{bmatrix}^\top$. The integrator also introduces constraint multipliers $\boldsymbol{\lambda}_k$, and we denote the full maximal-coordinate variable by $\mathbf{z}_k := \begin{bmatrix} \mathbf{p}_k^\top & \mathbf{v}_k^\top & \boldsymbol{\lambda}_k^\top \end{bmatrix}^\top$. In Eq. (1), $\mathbf{z}_0$ denotes the initial rope configuration, with $\boldsymbol{\lambda}_0 = \mathbf{0}$.

The rope is driven by the robot fingertip position $\mathbf{p}_{\text{tip},k} \in \mathbb{R}^3$, which we treat as an input. We enforce inextensibility via fixed-distance constraints between adjacent links of the rope and between the fingertip and the first link:

$$g(\mathbf{p}_k, \mathbf{p}_{\text{tip},k}) := \begin{bmatrix} \|\mathbf{p}_k^{(1)} - \mathbf{p}_{\text{tip},k}\|^2 - l^2 \\ \|\mathbf{p}_k^{(2)} - \mathbf{p}_k^{(1)}\|^2 - l^2 \\ \vdots \\ \|\mathbf{p}_k^{(N)} - \mathbf{p}_k^{(N-1)}\|^2 - l^2 \end{bmatrix} = \mathbf{0}.$$

TABLE IV
LIST OF ROPE PARAMETERS

ID	Name	Material	Diameter [mm]	Density [kg/m]	End Weight [g]
1	#10 Sash Spot Cord	Cotton	9	0.040	18
2	#14 Spot Cord	Cotton	12	0.081	80
3	Soft Braided	Cotton	15	0.076	80
4	Shoe Lace	Cotton	7	0.014	5
5	Thick Twisted	Cotton	25	0.139	50
6	3/8" Chain	Steel	20	0.514	50
7	3/8" Surgical Tubing	Latex	9	0.026	18

Let $\mathbf{G}(\mathbf{p}_k, \mathbf{p}_{\text{tip},k}) := \frac{\partial g}{\partial \mathbf{p}}(\mathbf{p}_k, \mathbf{p}_{\text{tip},k}) \in \mathbb{R}^{N\times 3N}$ be the constraint Jacobian, and let $\boldsymbol{\lambda}_k \in \mathbb{R}^N$ be the corresponding Lagrange multipliers, so that constraint forces on the links are $\mathbf{G}^\top \boldsymbol{\lambda}_k$.

The mass matrix is block diagonal,

$$\mathbf{M}_{\text{mass}} := \text{diag}\big(m\mathbf{I}_{3(N-1)},\ m_e\mathbf{I}_3\big) \in \mathbb{R}^{3N\times 3N},$$

and we collect gravity and internal bending forces (stiffness k and damping b) into a single force vector $\mathbf{f}(\mathbf{p}, \mathbf{v}) \in \mathbb{R}^{3N}$. With these definitions, a single timestep of the variational integrator can be written as an implicit system in the unknowns $(\mathbf{p}_{k+1}, \mathbf{v}_{k+1}, \boldsymbol{\lambda}_{k+1})$:

$$\mathbf{0} = \mathbf{p}_{k+1} - \mathbf{p}_k - \Delta t\, \mathbf{v}_{k+1}, \tag{11}$$

$$\mathbf{0} = \mathbf{M}_{\text{mass}}(\mathbf{v}_{k+1} - \mathbf{v}_k) - \Delta t\, \mathbf{f}(\mathbf{p}_{k+1}, \mathbf{v}_{k+1}) - \Delta t\, \mathbf{G}(\mathbf{p}_{k+1}, \mathbf{p}_{\text{tip},k+1})^\top \boldsymbol{\lambda}_{k+1}, \tag{12}$$

$$\mathbf{0} = g(\mathbf{p}_{k+1}, \mathbf{p}_{\text{tip},k+1}). \tag{13}$$

We solve Eqs. (11) to (13) with Newton's method at each timestep to roll out the rope trajectory, which defines the simulator f used in Eq. (1).

To compute the local linear model used in the inverse-model QP, we differentiate an implicit system. Let $\tilde{\mathbf{z}}$ stack all per-timestep unknowns $(\mathbf{p}_k, \mathbf{v}_k, \boldsymbol{\lambda}_k)$ over the horizon and let $\tilde{\mathbf{u}}$ stack the driven fingertip positions $\mathbf{p}_{\text{tip},k}$. Concatenating Eqs. (11) to (13) over time yields an implicit residual $\tilde{f}(\tilde{\mathbf{z}}, \tilde{\mathbf{u}}) = \mathbf{0}$. For a solution $(\tilde{\mathbf{z}}^*, \tilde{\mathbf{u}}^*)$, the implicit function theorem gives

$$\frac{\partial \tilde{\mathbf{z}}^*}{\partial \tilde{\mathbf{u}}} = -\left(\frac{\partial \tilde{f}}{\partial \tilde{\mathbf{z}}}\right)^{-1} \frac{\partial \tilde{f}}{\partial \tilde{\mathbf{u}}}. \tag{14}$$

In practice we do not form the inverse explicitly and instead solve linear systems with the KKT Jacobian $\frac{\partial \tilde{f}}{\partial \tilde{\mathbf{z}}}$. We implement the residual and its derivatives in Drake using automatic differentiation [31], and apply the chain rule through the Bezier spline and forward kinematics to obtain the system-model linearization $\mathbf{M}$ in Eq. (2).

D. Command Parametrization

The Bezier trajectory $\mathcal{B}$ is parametrized by the 8 knot points $\mathbf{u}(t) \in \mathbb{R}^{10\times 8}$ equally spaced in the time interval from 0 to T. $\mathcal{B}$ is defined as follows, with b_i representing the Bernstein basis functions:

$$\mathcal{B}(\mathbf{u}(t), T) = \sum_{i=0}^{N} \mathbf{u}_i\, b_i^N\left(\frac{t}{T}\right), \quad t \in [0, T]$$

$$b_i^N(s) = \binom{N}{i} s^i (1-s)^{N-i}, \quad s \in [0, 1].$$

E. Demonstration Timing Selection

Given a full demonstration of the flying knot, we select a time window from t_0 to t_f to limit demonstration motion to between the start of hand motion and the end of the follow-through motion. We select these times given a manually annotated rope collision time t_c with the following search procedure:

1) Select a temporary range $\tilde{t}_0$ and $\tilde{t}_f$ that excludes noisy data from the human picking up and placing the rope on the floor.
2) Search for maximum hand velocity in the range $[\tilde{t}_0, \tilde{t}_f]$ and label as t_{peak}
3) Step backwards in time from t_{peak} and update $\tilde{t}_0$ to the first point when the hand velocity is near-zero (3% of velocity at t_{peak}).
4) Set $t_f = t_c + 35$ms as a fixed follow through time length.
5) Integrate along the hand path motion between $\tilde{t}_0$ and t_f, then set t_0 to the time when 5% of the total path length is traveled.

$h(t)$ is then defined as 3D pose trajectory of the hand between t_0 and t_f. $\mathbf{x}^{\text{demo}}(t)$ is the rope trajectory from t_0 to t_c. Each demonstration type has a different execution speed and overall time length.

TABLE V
DEMONSTRATION TRACKING PARAMETERS

Parameter	Value
$\mathbf{q}_{\min}$	$[-6.28, -1.8, -6.28, -0.19, -6.28, -1.69, -6.28]$
$\mathbf{q}_{\max}$	$[6.28, 1.9, 6.28, 3.92, 6.28, 3.14, 6.28]$
$\dot{\mathbf{q}}_{\min}$	$-[3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14]$
$\dot{\mathbf{q}}_{\max}$	$[3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14]$
$\ddot{\mathbf{q}}_{\min}$	$-[100, 100, 100, 100, 100, 100, 100]$
$\ddot{\mathbf{q}}_{\max}$	$[100, 100, 100, 100, 100, 100, 100]$
w_j	5.0×10^{-7}
w_p	10
w_R	0.2
w_v	0.5
$z_{\min}$	1.2

F. Demonstration Tracking

The first 7 rows of $\mathbf{q}(t)$ correspond to the robot joint trajectory and the remaining 3 rows correspond to the robot base location. $\mathbf{e}_h(t)$ is the end-effector tracking error with respect to the desired fingertip trajectory $h(t)$ weighted by the cost matrix $\mathbf{W}_h$:

$$\mathbf{e}_h(t) := \begin{bmatrix} \mathbf{p}_{\text{tip}}(t) - \mathbf{p}_h(t) \\ \text{Log}\big(\mathbf{R}_h(t)^\top \mathbf{R}_{\text{tip}}(t)\big) \\ \dot{\mathbf{p}}_{\text{tip}}(t) - \dot{\mathbf{p}}_h(t) \end{bmatrix}$$
$$\mathbf{W}_h := \text{diag}(w_p \mathbf{I}_3,\ w_R \mathbf{I}_3,\ w_v \mathbf{I}_3),$$

with $\mathbf{p}_{\text{tip}}(t)$, $\mathbf{R}_{\text{tip}}(t)$, and $\dot{\mathbf{p}}_{\text{tip}}(t)$ given by the robot forward kinematics $\mathcal{K}$. Values for w_p, w_R, w_v can be found in Table V. While the joint dynamics limits between Table V and Table III are the same. The overall objective in the demonstration tracking is equally tracking the hand motion without any critical point weighting, therefore, the cost terms are applied equally throughout the hand motion.

The z_{tip} constraint specifies that the initial configuration of the robot at t_0 must have enough distance for the rope to dangle without hitting the floor.