

Learning Dynamic Rope 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—We introduce a Task-Level Iterative Learning Control method for dynamic manipulation of ropes. 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 inverts a 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 7–25 mm 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 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, completing a knot.

We adapt Iterative Learning Control (ILC) [7] to improve a command trajectory through real-world trials. ILC is a powerful tool for enabling a robot to improve its performance, as introduced by Arimoto et al. [5]. We find that typical ILC formulations fail in deformable-object manipulation. Model-based iterative learning control uses an approximate system model to convert task errors from real trials into command corrections. When ILC is learning to track rope target motions, however, equal weighting of task errors causes learning to fail.

Task-Level ILC uses feedback of task states in addition to or instead of robot states during learning [2]. We leverage Task-Level ILC with two key features to enable success on dynamic rope manipulation:

- **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 state in the trajectory. This improves learning performance and enables the transfer of the task across large parameter variations.
- **Object trajectory learning:** Our approach corrects the trajectory of state variables of the manipulated object,

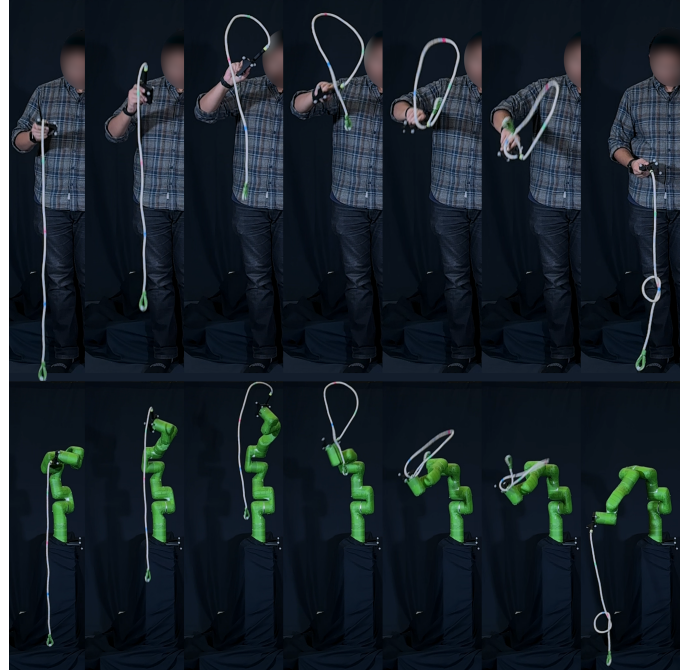


Fig. 1. Stages of a flying knot by both a human and an xArm 7 robot arm over 0.56s

which goes beyond the usual ILC approach of correcting the robot trajectory.

We demonstrate the effectiveness of Task-Level ILC for dynamic rope manipulation. We show that learning adapts to variations in rope dynamics, demonstrations, and the system model. Overall, the key contributions of this work include the following:

- We extend ILC to refining the trajectories of unactuated degrees of freedom of manipulated objects.
- We introduce Task-Level ILC for dynamic rope manipulation, demonstrating that learning typically requires a small number of real robot trials (< 10) and transfers across different ropes.
- We develop a new approach to ILC, 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

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, and has been previously explored in robotics by Yamakawa et al. [41].

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 to make the task easier.

A. Flying Knot Objective

In this case study, we develop the idea that it is useful to define a proxy goal for learning dynamic tasks, which we refer to as a “*critical point*”. The use of a proxy goal can simplify the learning task by making it easier to evaluate task success. Often the conclusion of a dynamic task results in a crash or a state where error information is lost. To our surprise, paying attention to errors equally throughout task execution is often not as useful as focusing on a particular point or phase of the task. The selected proxy goal can be based on an event such as the initiation of a particular contact, on some measure of task phase, or wall clock time from some trigger event. There are many such moments in the execution of a flying knot: the initial state, the first instant the loop is formed, the maximum outward swing or height of the rope end or any other part of the rope, the rope-rope collision, the rope end passing through the loop, or the final knotted state.

We chose the moment the rope collides with itself as the critical point in this case study. We were heavily influenced by instructional materials which emphasized this collision state or “strike point” as a crucial step, as seen in Fig. 2. The overhand flying knot task can be performed in several ways, but the variations share the same qualitative topological contact event, as visualized in Fig. 2, allowing the use of a single definition of *critical point*. We also chose this collision point because it is easy to detect and measure and occurs before our rope tracking system gets confused by the overlap of the rope during knot formation. While this moment in flying knot execution is not a perfect predictor of task success, correctly matching this state will likely lead to a knot, so we selected this contact initiation and the corresponding rope shape and velocity as the *critical point*.

We provide the robot with a single demonstration of the flying knot, with the *critical point* manually annotated, for an example of the rope state at the *critical point*, to provide a goal for learning the task.

III. RELATED WORK

A. Iterative Learning Control

Iterative learning control is a command update algorithm in which the system iteratively executes a feedforward command in the real world and maps errors into feedforward command

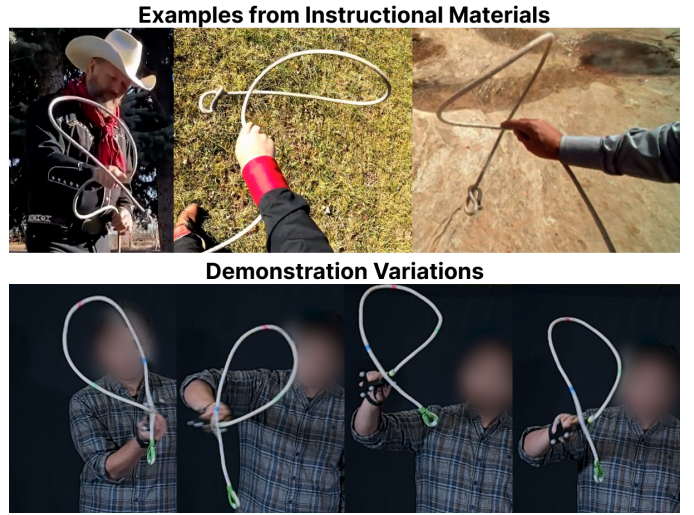


Fig. 2. **Critical point** for the flying knot shown in instructional videos [24, 8]. The bottom row is the *critical point* state across 4 demonstration variations. The shape of the rope at collision is used as the learning objective.

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 robot trajectory errors. Early work formalized the idea of “bettering” robot operation via iterative error-based updates [5]. In robotics, model-based ILC-style trajectory learning through practice was demonstrated on manipulators using nonlinear rigid-body robot models [6, 4], and subsequent work analyzed convergence and design choices for robot manipulators and other repetitive systems [21, 25].

Optimization-based ILC or Norm-Optimal ILC formulations showed how to incorporate constraints and compute command corrections via an optimization objective. Schoellig et al. [30] demonstrated aggressive quadrotor trajectory tracking by leveraging convex programs to reduce errors, and Ratcliffe et al. [28] 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 [13], point-to-point [11], time windows [35], and event triggers [23]. 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 [36]. Additional work introduces ILC-style model-gradient updates in deep off-policy reinforcement learning to improve sample efficiency [16]. In our work, we leverage optimization-based and event-based techniques to

focus learning on the *critical points* of the task.

B. Deformable Object Manipulation

Quasi-static deformable object manipulation, such as the handling of rope, hair, and cloth, has been addressed in a number of works [29, 44, 47] where many approaches focus on domains with strong perception and geometric priors [32, 45] and imitation/representation learning for rope manipulation [27, 18, 20].

Dynamic deformable object manipulation focuses on faster tasks, such as whipping [26], cloth flinging [10, 42], and dynamic rope knotting [41]. Iterative Residual Policy proposes online action refinement via a learned residual dynamics model [12]. Related works study learning for whip targeting, combining structured motion primitives with online adaptation or reinforcement learning [40, 39].

Recent work has further explored dynamic cable and cloth behaviors with self-supervised data collection. Real2Sim2Real learning enabled planar robot casting of cables by fitting models to physical rollouts [22], and subsequent systems learned to dynamically manipulate fixed-endpoint cables via arcing motions [46] or free-end cables in planar settings [38].

In Ha and Song [17], Chi et al. [12], and Zhang et al. [46], approximate 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 feedforward trajectories and demonstrate the repeatability of dynamic deformable object manipulation. Similarly, we leverage feedforward trajectories in the flying knot task.

Algorithm 1 summarizes Task-Level ILC.

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

- 1: **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
 - 2: **for** $k = 0$ **to** K **do**
 - 3: $\mathbf{x}_k(t) \leftarrow \text{Trial}(\mathbf{u}_k(t))$
 - 4: Critical-point error $\tilde{\mathbf{x}}_k(t_c) \leftarrow \mathbf{x}_k(t_c) - \mathbf{x}^{\text{demo}}(t_c)$
 - 5: Command update $\Delta \mathbf{u}_k(t) \leftarrow \mathcal{M}^{-1}(\tilde{\mathbf{x}}_k(t_c))$
 - 6: Update $\mathbf{u}_{k+1}(t) \leftarrow \mathbf{u}_k(t) - \Delta \mathbf{u}_k(t)$
 - 7: **end for**
-

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 error trajectories $\tilde{\mathbf{x}}(t)$ to command trajectory updates $\Delta \mathbf{u}(t)$ to iteratively refine a full sequence of commands. ILC can improve 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).

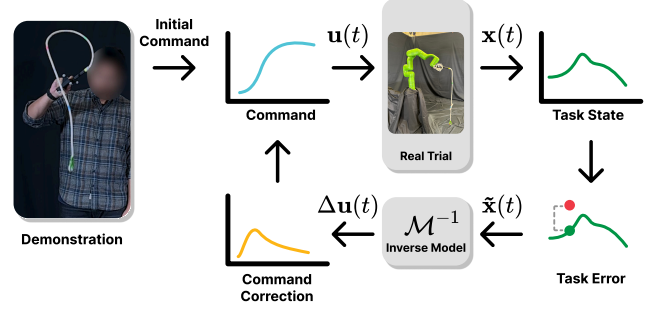


Fig. 3. **Task-Level Iterative Learning Control System:** A demonstration is converted to an initial command. The command trajectory $\mathbf{u}(t)$ is executed on the real system, and the resulting trajectory $\mathbf{x}(t)$ is measured. The task error $\tilde{\mathbf{x}}(t)$ at the *critical point*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

ILC is well-suited when several conditions hold: the task has a fixed goal across trials, the dynamics are repeatable enough that errors persist from one trial to the next, the system can be reset to a consistent initial condition, the outcome is measurable, and the initial trajectory is close enough to the desired trajectory for the particular update operator to converge [4]. Model-based ILC works well when the model is good enough to get the initial trajectory close enough to the desired trajectory, and for learning to converge [4].

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 $O(N)$ parameters, where N is the number of samples. A feedback control policy typically has several orders of magnitude more parameters. We further reduce the dimensionality of the action by representing the command trajectory with knot points of splines κ .

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 the system model is perfect, then all errors would be removed in one step (of course, the initial command would have been perfect and there would be no errors to correct). 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 robot 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

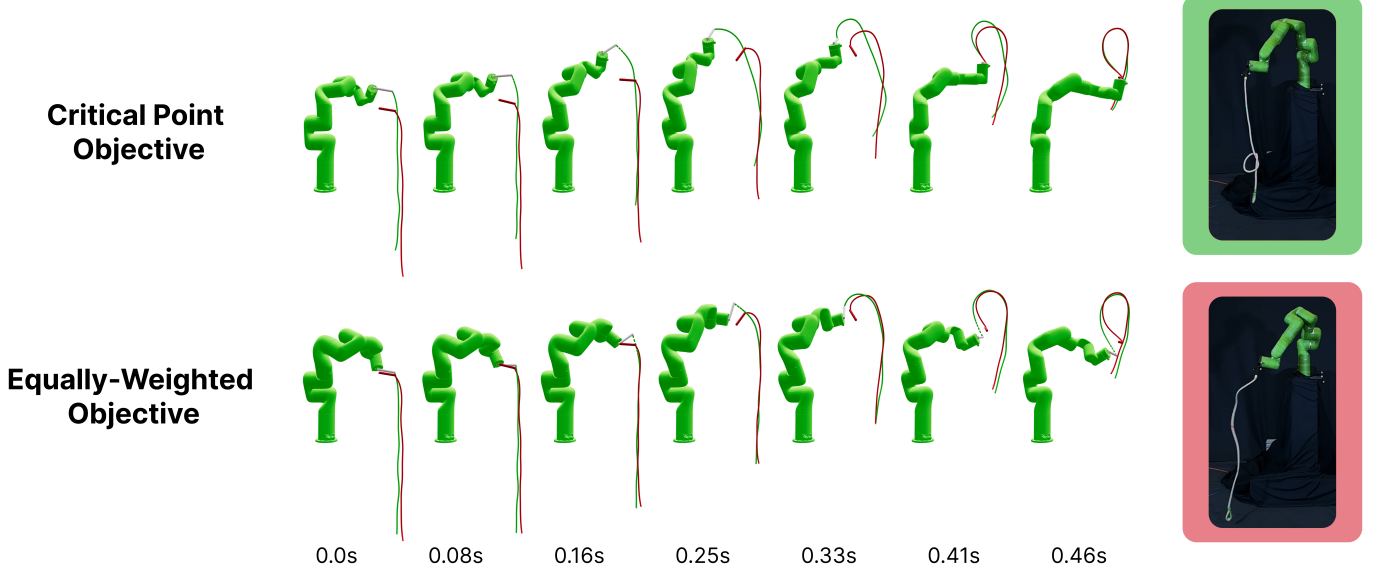


Fig. 4. **Critical point vs equally-weighted objective learned commands.** Each row is a real trial after 8 iterations with the corresponding objective. The green rope is the measured state from the real trial, and the red rope is the corresponding rope state from the demonstration. The rope state at 0.46s is the *critical point*. The *critical point* objective results in a successful flying knot, while the equally-weighted objective results in failure. During learning, the robot learns to deviate from the demonstration motion to reduce tracking error along the rope, which is why the demonstration rope handle position does not match the measured rope handle position.

of a deformable manipulated object from a goal trajectory. In addition, we explore the use of *critical point* objectives, which focus learning on the error at a *critical point* along the trajectory.

C. Critical Point Objective

The *critical point* is defined as a key moment in the task at which all task error reduction is targeted. Let t_c be the selected task phase and let $\mathbf{x}^{\text{demo}}(t_c)$ be the demonstrated task state. The corresponding objective minimizes the weighted error $\|\mathbf{x}(t_c) - \mathbf{x}^{\text{demo}}(t_c)\|_Q^2$ rather than the integral of tracking error over the full trajectory. For the flying knot task, we manually 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 during the collision.

D. Command Parameterization

We parameterize the feedforward robot command as 10 Bezier curves (7 joints + 3 base translation dimensions) with 8 knot points equally spaced in time: $\kappa \in \mathbb{R}^{10 \times 8}$. The base translation dimensions ensure translation invariance of the task objective, with constraints that prevent the robot base location from moving Eq. (2e). The fixed total execution time of the command (T) is defined by the total length of the demonstration hand motion. Given κ 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 in Appendix ??.

E. Robot Model

We model the robot as a kinematic chain parametrized by robot joint angles $\mathbf{q}$. 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 modeled with the robot motion matching the command, $\mathbf{q}_d = \mathbf{q}$. The desired trajectory is executed by the control system provided by the robot manufacturer, which mostly consists of joint-level PD servos. In reality, the robot motion does not exactly follow the commanded trajectory, but this error is compensated for indirectly during task-level ILC. Joint trajectories $\mathbf{q}(t)$ are mapped through the robot's forward kinematics $\mathcal{K}(\mathbf{q}(t))$ to determine the trajectory of the fingertip (a tube holding the rope). 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 set of m, k, b parameter for all but the last rope links and joints. All the ropes have a weight attached to the end, which is approximated with a different end link mass m_e . Our approximate rope model has fewer degrees of freedom

than the actual ropes, with only $N = 11$ links. 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).

The first point on the rope model is kinematically driven by the robot model's fingertip position and orientation 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 trajectory of the first link (full derivation found in Appendix ??). This makes the overall system model, as seen in Fig. 5, the following:

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

where $\hat{\mathbf{x}}$ is the model prediction of the rope state and κ are the command spline knot points. To simulate the rope dynamics, we implement a maximal coordinate variational integrator dynamics model from Brüdigam and Manchester [9]. The same fixed parameter set in Table I is used for all rope types; we do not fit model parameters to individual ropes.

G. Optimization-Based Inverse Model

Methods for an optimization-based inverse model were introduced for trajectory tracking in Schoellig et al. [30] and Schöllig and D'Andrea [31]. A Quadratic Program (QP) is formulated as a local inverse model to minimize a quadratic task objective while satisfying linear constraints by taking the measured critical-point error from the real trial and returns a command correction that is predicted to remove that error. 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. Our full inverse model QP is formulated as follows:

$$(\Delta\kappa_k^*, \Delta\mathbf{x}_k^*(t)) = \arg \min_{\substack{\Delta\kappa \\ \Delta\mathbf{x}(t)}} \|\Delta\mathbf{x}(t_c) - \tilde{\mathbf{x}}_k(t_c)\|_{\mathbf{Q}}^2 \quad (2a)$$

$$+ \sum_{t \in [t_c, T]} \|\Delta\kappa\|_{\mathbf{Q}_{ft}, \mathbf{b}_{ft}}^2 \quad (2b)$$

$$+ \sum_{t \in [0, T]} \|\Delta\kappa\|_{\mathbf{R}}^2 \quad (2c)$$

$$\text{s.t. } \Delta\mathbf{x}(t) = \mathbf{M} \Delta\kappa \quad (2d)$$

$$\Delta\kappa_{\text{base}} = 0 \quad (2e)$$

$$\mathbf{q}_{\min} \leq \mathbf{J}_p \Delta\kappa + \mathcal{B}(\kappa_k) \leq \mathbf{q}_{\max} \quad (2f)$$

$$\dot{\mathbf{q}}_{\min} \leq \mathbf{J}_v \Delta\kappa + \dot{\mathcal{B}}(\kappa_k) \leq \dot{\mathbf{q}}_{\max} \quad (2g)$$

$$\ddot{\mathbf{q}}_{\min} \leq \mathbf{J}_a \Delta\kappa + \ddot{\mathcal{B}}(\kappa_k) \leq \ddot{\mathbf{q}}_{\max} \quad (2h)$$

$$\tau_{\min} \leq \mathbf{J}_\tau \Delta\kappa + \mathcal{T}(\kappa_k) \leq \tau_{\max} \quad (2i)$$

The QP decision variables are the command correction $\Delta\kappa$ and the predicted state correction $\Delta\mathbf{x}(t)$. Based on the measured robot motion, we simulate the rope dynamics and linearize the system model $\mathcal{M}$ about the current simulated rope trajectory $\tilde{\mathbf{x}}_k(t)$ and current command κ_k :

$$\mathbf{M} = \left. \frac{\partial \mathcal{M}}{\partial \kappa} \right|_{(\tilde{\mathbf{x}}_k(t), \kappa_k)} \quad (3)$$

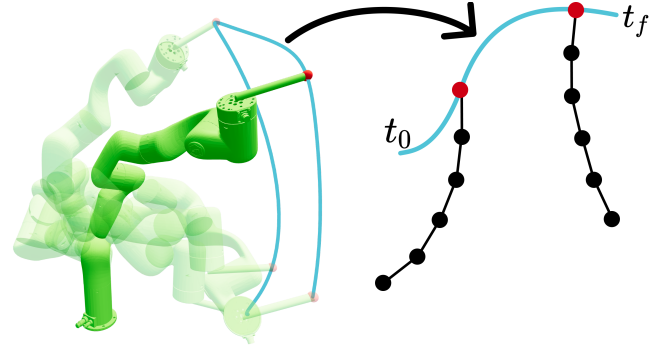


Fig. 5. **Left:** Kinematic robot model at various stages of a command. The opaque robot is at the point of contact. **Right:** Graphical representation of the point mass rope model. The robot's fingertip trajectory kinematically drives the first red dot. The remaining links are bound by distance constraints, and each joint has stiffness and damping. Together, the robot and rope models define our system dynamics model. All 3D visualizations are created using Viser [43].

The linearized dynamics constraint $\Delta\mathbf{x}(t) = \mathbf{M} \Delta\kappa$ (Eq. (2d)) enforces a model-based match between the command and trajectory corrections.

Since the ILC correction subtracts the returned correction from the current command, the first cost term penalizes the mismatch of the model-predicted state correction at t_c to match the measured real-system error $\tilde{\mathbf{x}}_k(t_c) = \mathbf{x}_k(t_c) - \mathbf{x}^{\text{demo}}(t_c)$. The term Eq. (2a) is the state-error-tracking cost applied only at the *critical point* t_c of the demonstration.

We apply a linearized quadratic tracking cost (Eq. (2b)) to the fingertip trajectory at $t > t_c$ to match the demonstrator's follow-through motion. The weighting matrices $\mathbf{Q}_{ft}, \mathbf{b}_{ft}$ are derived in Appendix ??.

A control cost (Eq. (2c)) penalizes updates to the 7 arm joints at each knot point for all t .

Constraint terms Eqs. (2f) to (2i) enforce position, velocity, acceleration, and torque limits on the commanded trajectory. Each has the form

$$\mathbf{x}_{\min} \leq \mathbf{J}_x \Delta\kappa + \mathbf{B}_x(\kappa_k) \leq \mathbf{x}_{\max},$$

where $\mathbf{B}_x(\kappa_k)$ evaluates the current spline-parameterized command (or its derivative for velocity and acceleration, or its torque prediction $\mathcal{T}$ from the rigid-body inverse dynamics), and $\mathbf{J}_x$ is the linearization of that mapping with respect to the command knot points. The total quantity is bounded so that the updated command stays within the robot's joint position, velocity, acceleration, and torque limits.

The base-translation constraint (Eq. (2e)) prevents the 3 base-translation degrees of freedom from varying across timesteps within a single command, so the robot base stays at a fixed location throughout each trial. ILC can still shift this fixed base location between iterations.

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

The inverse model returns the command component of the QP solution, $\mathcal{M}^{-1}(\tilde{\mathbf{x}}_k(t_c)) = \Delta\kappa_k^*$. After linearization, the objective is quadratic and the constraints are linear, so this inverse-model subproblem is a convex QP. We formulate the QP using the Drake optimization

TABLE I
ROPE MODEL PARAMETERS

Parameter	Value
Stiffness	1×10^5
Damping	50
Link Mass	1
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.

toolbox [33] and leverage the Clarabel Solver [15] to solve for the command update.

H. Demonstration

We provide the learning system with a single demonstration of the flying knot, performed on rope 1. When evaluating learning on other rope types, the same rope 1 demonstration is used as the starting point. For a given demonstration, we track the full hand trajectory and the trajectory of the rope up to collision. We use Vicon Vantage 16 motion capture [37] to track 11 markers in 3D, which correspond to the joints in the rope model. The collision point in the demonstration is manually annotated and used as the critical point t_c ; the algorithm does not discover this event autonomously. A search procedure is then performed to find the start and end time of the demonstration as described in Appendix ?? . $\mathbf{h}(t)$ 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 κ_0 to track the hand fingertip trajectory $\mathbf{h}(t)$. We solve the following trajectory optimization problem to minimize fingertip tracking error while satisfying the joint dynamics limits:

$$\begin{aligned}
\min_{\kappa} \quad & \sum_{t_i \in [0, T]} \|\mathbf{e}_h(t_i)\|_{\mathbf{W}_h}^2 + w_j \sum_{t_i \in [0, T]} \|\ddot{\mathbf{q}}(t_i)\|^2 \quad (4) \\
\text{s.t.} \quad & \mathbf{q}(t) = \mathcal{B}(\kappa, T) \\
& \mathbf{q}_{\min} \leq \mathbf{q}(t) \leq \mathbf{q}_{\max} \\
& \dot{\mathbf{q}}_{\min} \leq \dot{\mathbf{q}}(t) \leq \dot{\mathbf{q}}_{\max} \\
& \ddot{\mathbf{q}}_{\min} \leq \ddot{\mathbf{q}}(t) \leq \ddot{\mathbf{q}}_{\max} \\
& \dot{\mathbf{q}}(0) = \dot{\mathbf{q}}(T) = \mathbf{0} \\
& z_{\text{tip}}(\mathbf{q}_0) \geq z_{\min}
\end{aligned}$$

where $\mathbf{q}_0$ is the initial configuration of the robot. A detailed explanation of the hand-tracking objective $\mathbf{e}_h$, weighting $\mathbf{W}_h$, and constraints is provided in Appendix ?? . The initial-guess optimization is non-convex because the hand-tracking objective depends nonlinearly on the command through the Bezier curve, robot forward kinematics, and rotation error. We therefore formulate this trajectory optimization problem using the Drake toolbox [33] and solve using the SNOPT nonlinear solver [14].

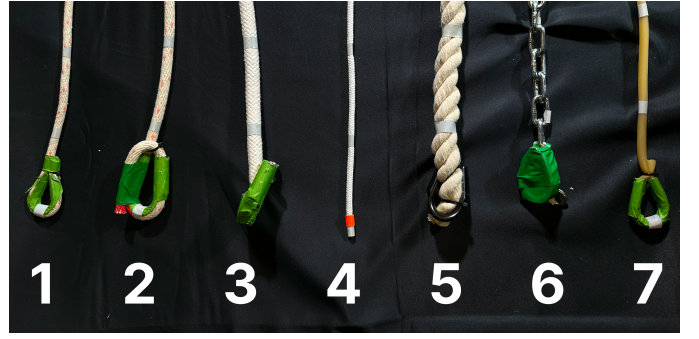


Fig. 6. 7 different rope types used for evaluation, including 4 braided ropes of various thickness and mechanical properties, a large twisted rope, a chain, and latex surgical tubing (very elastic); ranging in thicknesses of 7–25 mm and densities of 0.013–0.5 kg/m. Each rope has a mass affixed to aid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

V. EVALUATION

We evaluate the learning algorithm’s performance across 7 rope types (Fig. 6) and provide 4 variations of the flying knot demonstration (Fig. 2). We define a successful flying knot as achieving a topological overhand 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 our Task-Level ILC approach against two common approaches: direct tracking of the human demonstrator (hand motion) and equally-weighted 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.

A. Experiment Setup

We conduct flying knot experiments using the xArm 7 robot arm [34]. Both command execution and robot motion measurement occur at 250Hz. Commands that exceed the robot’s joint dynamics limits in position, velocity, acceleration and torque result in mid-motion faults and task failure.

Each rope is marked with 11 rings of 12 mm wide retro-reflective tape wrapped around the rope evenly spaced 10 cm apart along its length. These markers are tracked with a Vicon Vantage 16 motion capture system [37] at 200Hz. Tracking of rope markers often fails once the rope-rope collision occurs due to occlusion and marker misidentification across the different cameras, resulting in bad 3D reconstructions. Hand position is also tracked using 4 motion-capture markers. All ropes are 1.1 meters long. The rope is always started in a static state hanging down from the fingertip, though some types of ropes are stiff enough to 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,

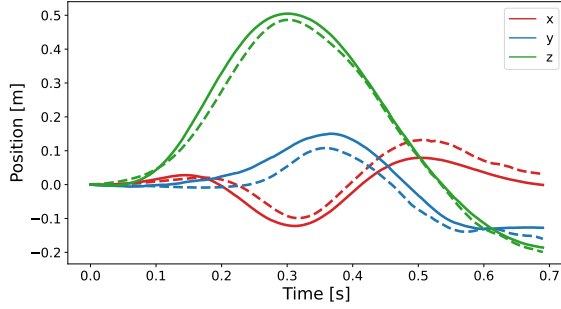


Fig. 7. **Initial command fingertip trajectory:** positions of the demonstration fingertip trajectory are dashed, and the robot's commanded initial attempt to track the demonstration is solid. The robot fails to track the demonstration motion due to kinematic and dynamic constraints.

for certain ropes, such as 4 and 7, following the demonstration nearly achieves a flying knot, even though the demonstration provided was using rope 1.

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 equally-weighted objective. The *critical point* objective, which focuses on errors at one point in the trajectory, is described in Section IV-C. The equally-weighted objective weights rope tracking errors equally along the pre-collision trajectory and is commonly found in trajectory tracking and behavior cloning.

As seen in Fig. 4, the critical-point objective aligns the rope state at a single point, at the collision, allowing for different rope motion beforehand, and results in 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. Over trials, the error at the critical point is reduced as shown in Fig. 8. This error difference 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 learned command. We evaluated this success rate by repeating 40 trials of a learned command for each condition. Execution on the real robot is 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. Continued learning can degrade the command, so learning is stopped after the first

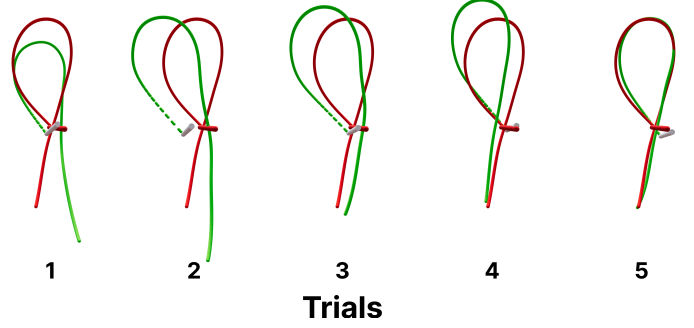


Fig. 8. **Rope configuration at the critical point over learning iterations:** Real rope (green) and demonstration rope (red) states for 5 iterations of Task-Level ILC where trial 5 resulted in a flying knot.

success (see Section VI-C for discussion of high-frequency error amplification over iterations).

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 ???. In general, ropes span a thickness of 7–25 mm, 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, starting from the same single rope 1 demonstration in all cases. The successful flying knot commands are shown in Fig. 9. 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. 9, 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; model errors are well known to cause ILC to slowly amplify high-frequency errors over iterations [6].

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 (Fast, Slow, Swipe, and Inverse). 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.

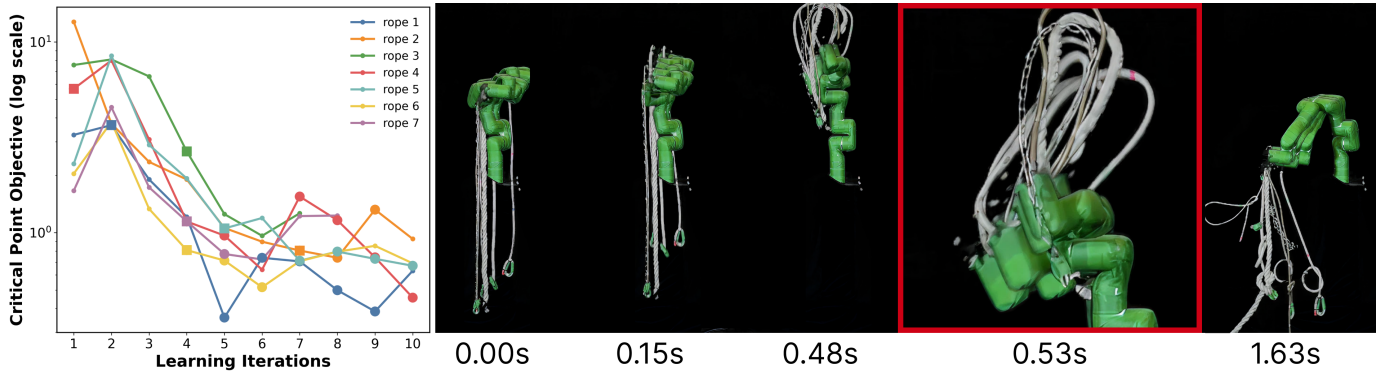


Fig. 9. **Left:** Critical point objective for 7 rope types over 10 iterations (rope 3 and 7 end early due to marker tracking failures). Squares represent the first successful flying knot, and every large solid dot represents a subsequent successful knot. **Right:** Execution on the real robot of successful flying knot execution on 7 rope types overlaid. The red highlighted frame is the critical point.

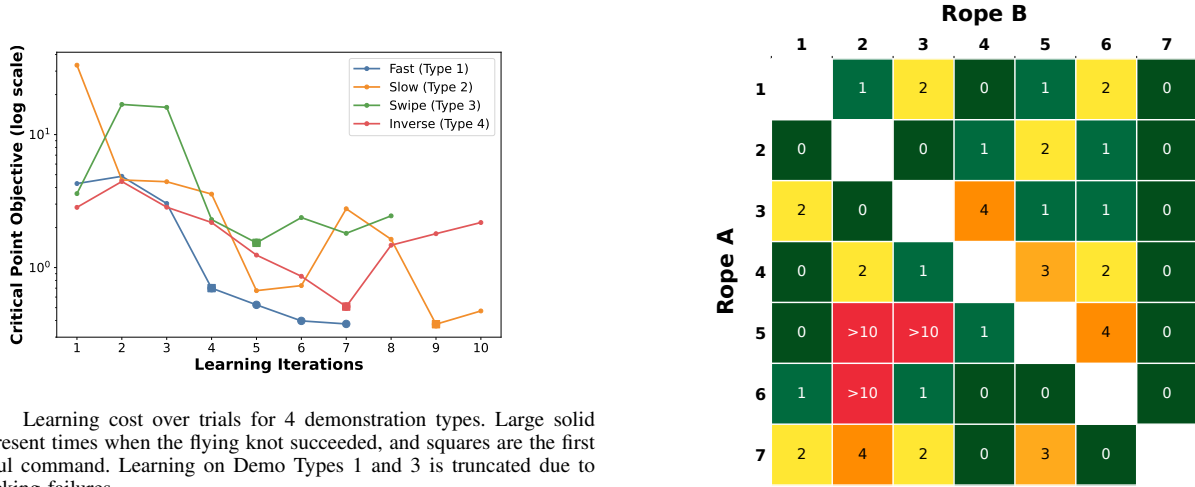


Fig. 10. Learning cost over trials for 4 demonstration types. Large solid dots represent times when the flying knot succeeded, and squares are the first successful command. Learning on Demo Types 1 and 3 is truncated due to rope tracking failures.

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.

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 zero, showing that the dynamics of the ropes are different enough to require different commands. For rope 7 (9mm latex surgical tubing), no additional learning is 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.

TABLE II
EFFECT OF MODEL PARAMETERS ON LEARNING PERFORMANCE.

Stiffness	End Mass	Trials Until Success
1×10^5	5	4
1×10^4	5	4
1×10^3	5	4
1×10^2	5	> 10
1×10^5	0.05	8
1×10^5	0.005	6

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. Model learning attempts to overcome this by adapting the model used for policy optimization with real data, but often suffers from limitations of the model 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.

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. [19] addresses some of these challenges by leveraging an instructor to guide learning toward subgoals. One can view the 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 ($\mathbf{u}(\mathbf{x})$). We further reduce the dimensionality by representing $\mathbf{u}(t)$ with a set of knot points κ . 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 [30].

ILC can suffer from high frequency instabilities. Mechanical systems with energy loss are inherently low-pass filters. Inverse models of these systems become high-pass filters. Learning using an inverse model as the learning operator across trials amplifies high-frequency modeling inaccuracies [6].

We parameterize the command trajectory using knot points of splines (κ) producing motions that are smooth and continuous. This reduces the effect of high frequency instabilities.

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

Our learning system is robust to changes in rope dynamics, different demonstrations, and model parameters. Further robustness evaluations can assess changes in rope length, non-uniform density, and end mass.

Our current method has several main failure modes that 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 minimum. Learning divergence occurs more frequently when task errors are too large at the *critical point*, and the linearized model is not accurate enough.

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 dynamic rope manipulation by learning to tie overhand 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. URL <https://doi.org/10.1145/1143844.1143845>.
- [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. URL <https://doi.org/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. URL <https://mitpress.mit.edu/9780262510578/model-based-control-of-a-robot-manipulator/>.
- [5] Suguru Arimoto, Sadao Kawamura, and Fumio Miyazaki. Bettering operation of robots by learning. *Journal of Robotic Systems*, 1(2):123–140, 1984. URL <https://doi.org/10.1002/rob.4620010203>.
- [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. URL <https://doi.org/10.1109/MCS.2006.1636313>.
- [8] Vince Bruce. Ropes & whips: Over hand knot. YouTube video, BookofCoolOfficial, June 2018. Accessed: May 11, 2026. [Online]. Available: <https://www.youtube.com/watch?v=7gya0XP53tU>.
- [9] 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. URL http://link.springer.com/10.1007/978-3-030-66723-8_12. Series Title: Springer Proceedings in Advanced Robotics.
- [10] 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>.
- [11] 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. URL <https://doi.org/10.1109/TCST.2017.2735358>.
- [12] 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>.
- [13] 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. URL <https://doi.org/10.1002/asjc.943>.
- [14] 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. URL <https://doi.org/10.1137/S1052623499350013>.
- [15] 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>.
- [16] 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>.
- [17] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding, 2021. URL <https://arxiv.org/abs/2105.03655>.
- [18] 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. URL <https://ieeexplore.ieee.org/document/10802478>.
- [19] 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>.
- [20] Zixuan Huang, Xingyu Lin, and David Held. Self-supervised cloth reconstruction via action-conditioned cloth tracking. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7111–7118, 2023. URL <https://doi.org/10.1109/ICRA48891.2023.10160653>.
- [21] 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. URL <https://doi.org/10.1109/70.105392>.
- [22] 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. URL <https://ieeexplore.ieee.org/document/9811651>.
- [23] 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. URL <https://ieeexplore.ieee.org/document/9222356>.
- [24] Dan Mink. Trick roping: The flying overhand knot. YouTube video, The Rhinestone Roper, January 2021. Accessed: May 11, 2026. [Online]. Available: <https://www.youtube.com/watch?v=lwsydbq3SWU>.
- [25] Kevin L Moore, Mohammed Dahleh, and SP Bhat-tacharyya. Iterative learning control: A survey and new results. *Journal of Robotic Systems*, 9(5):563–594, 1992. URL <https://doi.org/10.1002/rob.4620090502>.
- [26] 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. URL <https://doi.org/10.1109/BioRob49111.2020.9224399>.
- [27] 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. URL <https://doi.org/10.1109/ICRA.2017.7989247>.
- [28] 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. URL <https://doi.org/10.1109/TRO.2006.882927>.
- [29] 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. URL <https://doi.org/10.1109/LRA.2022.3187843>.
- [30] Angela P. Schoellig, Fabian L. Mueller, and Raffaello D’Andrea. Optimization-based iterative learning for precise quadcopter trajectory tracking. 33(1):103–127. URL <https://doi.org/10.1007/s10514-012-9283-2>.
- [31] Angela Schöllig and Raffaello D’Andrea. Optimization-based iterative learning control for trajectory tracking. In *2009 European Control Conference (ECC)*, pages 1505–1510, 2009. URL <https://doi.org/10.23919/ECC.2009.7074619>.
- [32] 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. URL <https://ieeexplore.ieee.org/document/8403315>.
- [33] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL <https://drake.mit.edu>.
- [34] UFACTORY. xArm 7 Robotic Arm. <https://www.ufactory.us/product/ufactory-xarm-7>, 2026. Accessed: 2026-05-11.
- [35] 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. URL <https://doi.org/10.23919/ACC.2004.1384669>.
- [36] 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>.
- [37] Vicon Motion Systems Ltd. V16 Camera Specifications. <https://help.vicon.com/space/Vantage/15041618/V16%2Bcamera%2Bspecifications>, 2024. Accessed: 2026-05-11.
- [38] 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>.
- [39] 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. URL <https://ieeexplore.ieee.org/document/10719935/>. ISSN: 2155-1782.
- [40] 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. URL <https://ieeexplore.ieee.org/document/9636663>.
- [41] 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. URL <https://doi.org/10.1109/IROS.2010.5651168>.
- [42] 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. URL <https://doi.org/10.1109/ICRA.2011.5979606>.
- [43] Brent Yi, Chung Min Kim, Justin Kerr, Gina Wu, Rebecca Feng, Anthony Zhang, Jonas Kulhanek, Hong Suk Choi, Yi Ma, Matthew Tancik, and Angjoo Kanazawa. Viser: Imperative, web-based 3d visualization in Python, 2025. URL <https://arxiv.org/abs/2507.22885>.
- [44] Hang Yin, Anastasia Varava, and Danica Kragic. Modeling, learning, perception, and control methods for deformable object manipulation. *Science Robotics*, 6(54): eabd8803, 2021. URL <https://www.science.org/doi/abs/10.1126/scirobotics.abd8803>.
- [45] Uksang Yoo, Adam Hung, Jonathan Francis, Jean Oh, and Jeffrey Ichnowski. Ropotter: Toward robotic pottery and deformable object manipulation with structural priors, 2024. URL <https://arxiv.org/abs/2408.02184>.

- [46] 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>.
- [47] Chengyang Zhao, Uksang Yoo, Arkadeep Narayan Chaudhury, Giljoo Nam, Jonathan Francis, Jeffrey Ichnowski, and Jean Oh. Dymo-hair: Generalizable volumetric dynamics modeling for robot hair manipulation, 2026. URL <https://arxiv.org/abs/2510.06199>.