Planar Robot Casting with Real2Sim2Real Self-Supervised Learning

Vincent Lim\textsuperscript{1,*}, Huang Huang\textsuperscript{1,*}, Yunliang Chen\textsuperscript{1}, Jonathan Wang\textsuperscript{1}, Jeffrey Ichnowski\textsuperscript{1}, Daniel Seita\textsuperscript{1}, Michael Laskey\textsuperscript{2}, Ken Goldberg\textsuperscript{1}

Abstract—Manipulation of deformable objects using a single parameterized dynamic action can be useful for tasks such as fly fishing, lofting a blanket, and playing shuffleboard. Such tasks take as input a desired final state and output one parameterized open-loop dynamic robot action which produces a trajectory toward the final state. This is especially challenging for long-horizon trajectories with complex dynamics involving friction. This paper explores the task of Planar Robot Casting (PRC): where one planar motion of a robot wrist holding one end of a cable causes the other end to slide across the plane toward a desired target. PRC allows the cable to reach points beyond the robot’s workspace and has applications for cable management in homes, warehouses, and factories. To efficiently learn a PRC policy for a given cable, we propose Real2Sim2Real, a self-supervised framework that automatically collects physical trajectory examples to tune parameters of a dynamics simulator using Differential Evolution, generates many simulated examples, and then learns a policy using a weighted combination of simulated and physical data. We evaluate Real2Sim2Real with three simulators, Isaac Gym-segmented, Isaac Gym-hybrid, and PyBullet, two function approximators, Gaussian Processes and Neural Networks (NNs), and three cables with differing stiffness, torsion, and friction. Results on 16 held-out test targets for each cable suggest that the NN PRC policies using Isaac Gym-segmented attain median error distance (as % of cable length) ranging from 8% to 14%, outperforming baselines and policies trained on only real or only simulated examples. Code, data, and videos are available at https://tinyurl.com/robotcast.

I. INTRODUCTION

Although there is substantial research on learning ballistic throwing or hitting motions for rigid objects [29, 66], there is less research into learning dynamic motions to manipulate deformable objects such as cables and fabrics. Dynamics modeling in these contexts is challenging due to uncertainty in deformability, elasticity, and friction during the object’s motion. When computing a single action in this setting (without feedback control), the complexities of state estimation and dynamics modeling are compounded by the long duration for which the system evolves after the robot action. Simulation is often used to avoid collecting physical examples, which can be time-consuming and hazardous. However, overcoming the simulation to reality (Sim2Real) gap is a long-standing problem in robotics [20, 31, 40, 53], and is particularly difficult for deformable objects.

In this paper, we consider Planar Robot Casting (PRC), a procedure in which a robot dynamically manipulates a cable through 2D space, as illustrated in Fig. 1. We use the generic term cable to refer to any 1D deformable object with low stiffness, such as cables, ropes, and threads. We propose Real2Sim2Real, a self-supervised robot learning framework that starts by collecting physical examples, uses them to tune a simulator, and then uses a combination of physical and simulated examples to train policies for PRC. As motion is restricted to a 2D plane, we do not address the 3D analog of PRC, Spatial Robot Casting (SRC), used in tasks such as fly fishing [15]. PRC is easier to visualize than SRC but can be harder to model due to inherent uncertainty about static and dynamic friction. In both PRC and SRC, the cable is infinite-dimensional and there is a long time horizon for the motion of the cable free end resulting from a motion at the controlled end. This paper contributes:

1) A formulation of Planar Robot Casting, with an automated reset and parameterized one-step action space.
2) Real2Sim2Real, a robot learning framework for efficiently learning policies for PRC.
3) A hardware testbed for PRC with a UR5 robot and an overhead high-speed video camera.

\textsuperscript{*}Equal contribution.

\textsuperscript{1}AUTOLAB at the University of California, Berkeley, USA.

\textsuperscript{2}Toyota Research Institute, CA, USA.
Furthermore, unlike Chang and Padir, we do not require potentially brittle RL algorithms [1, 18] and we train policies using supervised learning and do iterations of collecting real-world data with simulator tuning. Similarly, we tune a simulator, but do not require repeated step uses sensors to update a cable model in simulation. The Sim2Real step determines grasp points, and the Real2Sim step uses sensors to update a cable model in simulation. In contrast to these works, we use dynamic motions, but focus specifically on cable manipulation. Furthermore, we use a tuned simulator to accelerate learning and do not require tactile sensing.

C. Dynamic Manipulation

In dynamic manipulation, a robot executes rapid actions to quickly move objects to desired configurations [33]. In early work, Lynch and Mason [29] study planar dynamic manipulation primitives such as snatching, throwing, and rolling, and Ruiz-Ugalde et al. [45] build a physics-based model of pushing actions so that a robot can slide objects into a desired pose. More recent work has introduced robots that can catch items by predicting item trajectories [21], toss arbitrary objects via self-supervised learning [66], and swing items upwards using tactile feedback [55]. As with these works, we use dynamic motions, but focus specifically on cable manipulation. Furthermore, we use a tuned simulator to accelerate learning and do not require tactile sensing.

D. Dynamic Manipulation of Deformable Objects

Researchers have developed several analytic physics models to describe the dynamics of moving cables. For example, Gatti-Bonoa et al. [15] present a 2D dynamic model of fly fishing. They model the fly line as a long elastica and the fly rod as a flexible Euler-Bernoulli beam, and propose a system of differential equations to predict the movement of the fly line in space and time. In contrast to continuum models, Wang et al. [56] propose using a finite-element model to represent the fly line by a series of rigid cylinders that are connected by massless hinges. In contrast to these works, we evaluate on physical robots.

For robotic dynamic-cable manipulation, Yamakawa et al. [60–63] show that a high-speed manipulator can tie knots using snapping motions. They simplify the modeling of cable deformation by assuming each cable component follows the robot end-effector motion with constant time delay. Kim et al. [22] study a mobile robot system with a cable attached as a tail that can strike objects. They use a Rapidly-exploring Random Tree (RRT) [23] and a particle-based representation to address the uncertainty in state transition. In contrast, we aim to control cables for tasks in which we may be unable to rely on assumptions in [22, 61] for cable motion.

In closely related prior work, Zhang et al. [67] propose a self-supervised learning technique for dynamic manipulation of fixed-end cables for vaulting, knocking, and weaving. They parameterize actions by computing a motion using a quadratic program and learning the apex point of the trajectory. In contrast, we use free-end cables. Zimmermann et al. [68] study the dynamic manipulation of deformable beams and cloth in the free-end setting. They model elastic objects using the finite-element method [4] and use optimal control techniques for trajectory optimization. They study whipping a beam so that its free end hits a predefined target with maximum speed. The simulated motions perform well in real for simple dynamical systems such as a pendulum, but performance deteriorates for complex soft bodies due to the reality gap. We develop a learned, data-driven approach for robotic manipulation of free-end cables and focus on PRC.
III. Problem Statement

In Planar Robot Casting, a robot gripper holds a cable at one endpoint and swings it along a planar surface with a single continuous motion so that the other endpoint comes to rest at a target \( s_d \). The held cable endpoint is at polar coordinate \( s = (r, \theta) \), with the robot base at the origin. The objective is to find a per-cable policy \( \pi \) that minimizes the expected error \( |s_{f,c} - s_{d,c}|^2 \), where \( s_{f,c} \) is the final state and \( s_{d,c} \) is \( s_d \) in Cartesian coordinates. We assume that the \( s_d \) is reachable by the cable endpoint (see Fig. 1).

IV. Method

To learn a policy that accounts for variations in cable properties such as mass, stiffness, and friction, we propose a Real2Sim2Real (R2S2R) framework (Fig. 2) and apply it to PRC. To support this framework, we define a reset procedure to bring the system into a consistent starting state and a parameterized trajectory function that generates a dynamic arm action. The first step of R2S2R autonomously collects physical trajectories, and the second step tunes a simulator to match the physical environment. R2S2R then uses the tuned simulator to generate simulated trajectories, and combines the simulated and physical datasets to train a policy.

A. Reset Procedure

To automate data collection and bring the cable into a consistent starting state before each action, we define a 5-step reset procedure, in which the robot (1) lifts the cable up with the free-end touching the surface to prevent the cable from dangling; (2) continues to lift the cable such that the free-end is just above the surface; (3) hangs still for 3 seconds to stabilize; (4) swings the cable out of the plane, so that the cable straightens along the center axis of the surface and lands with its endpoint far from the robot base; and (5) slowly pulls the cable towards the robot to the reset position \((r_0, 0)\). The project website illustrates the reset procedure.

B. Parameterized Actions

To generate quick, smooth, dynamic actions that are low dimensional, thus facilitating data generation and training, we define a parameterized action composed of two sweeping arcs (Fig. 3):

\[
a = (\theta_1, r_1, \theta_2, r_2, \alpha, v_{\text{max}}).
\]

The motion starts at \((r_0, 0)\), arcs to \((r_1, \theta_1)\), and arcs back to \((r_2, \theta_2)\). In the motion, \(\alpha\) is the wrist joint rotation about the \(z\)-axis during the second arc, and \(v_{\text{max}}\) is the maximum velocity. This parameterization is motivated by observing human attempts at the PRC task.

We convert action parameters to a trajectory in polar coordinates using a cubic spline to smoothly interpolate the radial coefficient from \(r_0\) to \(r_2\), and use a maximum-velocity spline to interpolate the angular coefficient from 0 to \(\theta_1\) and from \(\theta_1\) to \(\theta_2\), with maximum velocity \(v_{\text{max}}\). The maximum-velocity spline uses a jerk-limited bang-bang control, having observed that the UR5 has difficulty following trajectories with high jerk [19]. We assume a direction change between the two arc motions, so the angular velocity at \(\theta_1\) is 0.

This parameterization enables state symmetry. For all datasets, we sample actions such that \(\theta_1 > 0, \theta_2 < 0, \) and \(\alpha > 0\), to obtain targets on the left of the workspace axis of symmetry. If \((\theta_1, \theta_2, r_2, \alpha, v_{\text{max}})\) produces target \((r_d, \theta_d)\), \((-\theta_1, -\theta_2, r_2, -\alpha, v_{\text{max}})\) will produce \((r_d, -\theta_d)\). Thus, we do not evaluate targets on the right of the axis of symmetry.

C. Self-Supervised Physical Data Collection

For the first step of R2S2R, we autonomously collect a physical dataset \(D_{\text{phys}}\) and take a subset of \(D_{\text{phys}}\) to form a simulator tuning dataset \(D_{\text{tune}}\) to perform system
identification in simulation (Sec. IV-E). To create $D_{phys}$, we grid sample each action parameter to generate $5 \times 5 \times 5 \times 4 \times 2$ trajectories, then filter it such that each trajectory abides by joint limits and is collision-free, to generate $|D_{phys}| = 522$ trajectories.

For automatic data labeling, we record each trajectory using a camera and use contour detection in OpenCV [6] to track a brightly colored endpoint. We extract the 2D waypoint location $p_t = (x_t, y_t)$, where $t$ is the timestep, every 100 ms from the start of the robot’s trajectory. For each trajectory $m_j$ in $D_{phys}$, the number of waypoints collected is $K_j = \lfloor T_j / 100 \rfloor$, where $T_j$ is the duration of $m_j$ in milliseconds.

### D. Three Simulation Models for Robot Casting

R2S2R then tunes a simulator so that it can generate data to augment policy training. We first consider which simulator and model best match real from 3 options: PyBullet [10] and two versions of NVIDIA Isaac Gym [32]. We set simulated cable geometry (e.g., length and radius), to match the real cable. Fig. 4 shows a visual comparison, and the project website provides details.

**PyBullet** is a CPU-based deterministic rigid-body physics simulator used in prior work on deformable object manipulation [34, 48]. We model the cable as a string of capsule-shaped rigid bodies with 6-DOF spring constraints between each consecutive pair. We tune ten parameters: twist stiffness, bend stiffness, mass, lateral friction, spinning friction, rolling friction, endpoint mass, linear damping, angular damping, and dynamic friction.

**NVIDIA Isaac Gym** is a GPU-based robotics simulation platform that supports the FleX [30] particle-based physics simulator designed for deformable objects and rigid bodies. We test two simulation models for the cable: a segmented model and a hybrid model, both using FleX. The segmented model is a string of 18 capsule-shaped rigid bodies with consecutive pairs linked together by a ball joint. To model cable stiffness, we tune the joint friction, cable mass, endpoint mass, and planar friction to capture variation in each respective parameter in real. The hybrid model has the same rigid endpoint from the segmented model, but the rest of the cable is a soft-body rod. We tune the Young’s modulus of the soft-body rod to model both the cable stiffness and elasticity, dynamic friction of the ground plane, and rigid endpoint mass to capture variations in friction and endpoint properties.

### E. Simulator Parameter Tuning for $D_{sim}$ collection

To tune the parameters of each simulator so that simulated and real trajectories closely match, we employ a consistent protocol. Following system identification [26], the protocol uses physical trajectories from $D_{tune}$ (Sec. IV-C) to minimize the discrepancy between simulated and real trajectories. We define the simulation tuning objective as finding simulation parameters that minimize the average $L^2$ distance between the cable endpoint in simulation $p_t$ to the target endpoint $p_t$ over all trajectory timesteps, averaged over a batch of trajectories. We call this metric the average waypoint error.

As the simulation models may have parameters that have no physical analog, such as the joint friction in the segmented model, or have parameters that do not act similarly to their physical counterparts [27], we choose tuning algorithms that make no assumptions about the underlying dynamics model. We evaluate two tuning algorithms, Bayesian Optimization (BO) and Differential Evolution (DE), both of which are derivative-free black box optimization methods.

Bayesian Optimization builds a probabilistic surrogate model and uses an acquisition function that leverages the uncertainty in the posterior to decide where to query next [14]. BO has been used in policy search for RL hyperparameters [25, 50], and for tuning simulation fluid dynamics [27]. Differential Evolution is an evolutionary population-based stochastic optimization algorithm widely used for global optimization of nondifferentiable, multi-modal, and nonlinear objectives [52]. DE maintains a population of candidate solutions at each iteration and generates new candidates by combing each population member with another mutated population candidate member. It then evaluates the fitness of the trial candidates to update the population of candidate solutions until convergence. Prior work has shown DE to be effective at simulation parameter tuning [9].

We evaluate each algorithm via Sim2Sim tests. Using a fixed set of simulation parameters, we generate a small dataset of trajectories to tune randomly initialized simulators to test the effectiveness of DE and BO. We consider the discrepancy between ground truth and predicted simulation parameters and average waypoint error (see results in Sec. V-A). We then tune the simulator parameters for each simulation model to the physical data using the best performing tuning algorithm. We tune the simulator using the same process but with physical trajectories from $D_{tune}$.

### F. Size of Simulator Tuning Set $D_{tune}$

We subsample 20 trajectories from $D_{phys}$ to generate dataset $D_{tune}$. We evaluated whether increasing the size of $D_{tune}$ would reduce the discrepancy between simulation and real using a test set of 30 random physical trajectories not in $D_{phys}$. We observed negligible difference in test errors between using 20 and 60 trajectories, but the latter would require $2 \times$ as long to tune on the hybrid and PyBullet models. Tuning with $|D_{tune}| = 20$ trajectories did not lead to a significant speedup in the segmented model, therefore we held this value fixed.
After tuning, R2S2R generates training data $D_{\text{sim}}$ by grid sampling $\theta_1, \theta_2, r_2, \psi$, and $v_{\text{max}}$ to generate $15 \times 15 \times 15 \times 10 \times 2$ values for each respective parameter. We then filter out any trajectories that violate joint limits or with collisions to generate $|D_{\text{sim}}| = 21,450$ $(a, s)$ simulated trajectories.

G. Policies

1) Forward Dynamics Model: There is multi-modality between a target endpoint $s$ and valid actions $a$, so we do not learn a policy that directly predicts $a$ given $s_d$. Instead, we learn a forward dynamics model $f_{\text{fowr}}$ that predicts $s$ given $a$, and interpolates to select an action. We parameterize $f_{\text{fowr}}$ using a fully connected neural network. Given a dataset $D$ of $(a, s_f)$ trajectories, the neural network learns to predict $s_f$ given $a$ via supervised regression using MSE. During evaluation, the policy $\pi$ grid samples 67,500 input actions $a$ to form a set $A$ of candidate actions. It then passes each action through the forward dynamics model $f_{\text{fowr}}$, which outputs the predicted endpoint location $(\hat{x}, \hat{y})$ in Cartesian space. Given a target endpoint $s_d$ (in polar coordinates), the action $a = \pi(s_d)$ selected is the one minimizing the Euclidean distance between the predicted endpoint $s_f$ and the target endpoint $s_d$:

$$\pi(s) = \arg\min_{a \in A} \|s_{f,c} - s_{d,c}\|_2,$$  

where $s_{f,c}$ and $s_{d,c}$ are $s_f$ and $s_d$ in Cartesian coordinates respectively.

2) Baseline Policies: We consider two alternative policies: 

- **Cast and Pull**: Given a target cable endpoint location $s_d = (r_d, \theta_d)$, first perform a casting motion that causes the free end to land at $(r, \theta_d)$. To perform this motion, the robot rotates $\theta_d$ radians, and executes the rotated reset motion as in Sec. IV-A. After the casting motion, the robot slowly pulls the cable toward the base for a distance of $r - r_d$. The robot arm is limited to a minimum $r$ coordinate of $r_{\text{min}} = 0.55$ to prevent the end effector from hitting the robot’s supporting table, where $r_{\text{min}}$ is the distance from the center of the robot base to the edge of the table, limiting the reachable workspace.

- **Forward Dynamics Model learned with Gaussian Process**. As a learning baseline, we use Gaussian Process (GP) regression [37, 44]. Using $D_{\text{phys}}$, we train a GP regressor to predict the cable endpoint location $s_f$ given input action $a$. As with the neural network forward dynamics model, we grid sample 67,500 input actions and select the trajectory that minimizes the predicted Euclidean distance to the target.

V. EXPERIMENTS

We evaluate the R2S2R pipeline on PRC with 3 cables (Fig. 5) using a physical UR5 robot (Fig. 1). The working surface is a 2.45 m wide and 1.55 m tall masonite board, painted blue and sanded to create a consistent friction coefficient across the surface. We record observations from an overhead Logitech Brio 4K webcam recording 1920×1080 images at 60 frames per second.

We define $\pi_3$ as a forward dynamics model trained on a small real dataset $D_{\text{phys}}$ and $\pi_2$ as a forward dynamics model trained on a large simulated dataset $D_{\text{sim}}$. We define $\pi_1$ as a forward dynamics model trained on the combined dataset $D_{\text{phys}} \cup D_{\text{sim}}$, but since $|D_{\text{sim}}| \gg |D_{\text{phys}}|$, we 1) upsample $D_{\text{phys}}$ by randomly duplicating samples such that the real data make up 30-40% of the combined dataset to force the model to learn more from the real data, and 2) weight the loss function so that samples from $D_{\text{phys}}$ are weighted higher.

A. Comparing Tuning Methods

To test the effectiveness of BO and DE for tuning the Isaac Gym simulators, we generate simulated data for several hybrid models and segmented models with arbitrarily chosen parameters. We then apply BO and DE to tune the respective simulator model parameters. For BO, we use the GPyOpt [3] library, and test three acquisition functions: Expected Improvement (EI), Lower Confidence Bound (LCB), and Maximum Probability of Improvement (MPI). We use DE as implemented in SciPy [54]. With 5 tuning trajectories, EI performs better than LCB and MPI. The tuning errors using BO range from 3.39% to 17.98% for the hybrid model and 1.09% to 9.25% for the segmented model. DE consistently tunes the parameters to within 1% of the ground truth parameters for both the hybrid and segmented models, thus we use DE for all further simulator tuning results.

B. Real2Sim

Table I reports the Real2Sim simulator tuning results using DE. We observe that the segmented model consistently outperforms the PyBullet and hybrid models in minimizing the discrepancy between simulation and real. We attribute the larger errors of the hybrid model to the Young’s modulus parameter’s effect on the cable stiffness and elasticity. We find that the tuning algorithms reduce the value of Young’s modulus to decrease cable stiffness, which causes the cable to stretch along the length of the cable, an effect we did not observe in real. We attribute the PyBullet model performance to spring-system simulation instability associated with high

![Fig. 5: Three cables used in physical experiments. Cable 1 is a thin blue paracord, Cable 2 is a nylon cable, and Cable 3 is a thick jump rope. Each endpoint has an attached mass. The respective cable lengths are 0.63 m, 0.65 m, and 0.65 m, the respective masses are 8 g, 50 g, and 45 g, and the respective radii are 4.5 mm, 10 mm, and 14 mm.](image-url)
As summarized in Table II, we evaluate PRC policies \( \pi_3, \pi_2, \) and \( \pi_1 \) on cable 1, as well as two baseline policies.

1) Baseline Policies: The analytic “Cast and Pull” baseline performs the worst out of any of the policies, as the robot would collide with the base for targets near the base. The GP trained on \( D_{\text{phys}} \) has substantially worse median, 3rd quartile, and maximum errors, but \( \pi_2 \) has a substantially higher maximum error, corresponding to trajectories where the simulator has high tuning error. As the simulator is tuned with only 20 trajectories, compared to the over 500 trajectories used to train \( \pi_3 \), we achieve similar evaluation performance while using 96% fewer physical trajectories. When we combine the two datasets to train \( \pi_1 \), the median error drops by nearly 50%. However, the maximum error rose to above 100% for a single target, suggesting that while the combined dataset substantially improves overall results, it may introduce outliers in regions of the workspace where the simulator is unable to match reality.

Given that \( \pi_1 \) performs the best out of the policies, we proceed to evaluate the performance for 3 cables on 16 target positions using policy \( \pi_1 \) trained for each cable separately, and repeat each action 5 times per target. Results are summarized in Table III. We quantify the aleatoric uncertainty with the 95% confidence ellipses in Fig. 6. We attribute the epistemic uncertainty to two sources: the Sim2Real gap, where trajectories executed in simulation do not accurately reflect real, and the learning error in the forward dynamics model. The results suggest that the R2S2R pipeline can apply to other cables.

VI. Conclusion and Future Work

In this paper, we present Real2Sim2Real, a self-supervised learning framework, and apply it to Planar Robot Casting. Experiments suggest that the framework, which collects physical data, tunes a simulator to match the physics of the real data, and trains policies from a weighted combination of real and simulated data, can achieve an error between 12% and 15% on the PRC task. The self-supervised fine-tuning process relies on executing hundreds of trajectories in the physical setup, which may be infeasible in other dynamic manipulation domains. Hence, in future work, we will explore more advanced simulator tuning methods for more sample-efficient simulator tuning and robust Sim2Real policy transfer [2, 12, 42, 43]. The presented framework also must be rerun for each cable, motivating meta-learning [13] approaches to rapidly adapt to other cables.

References


