

# Supplementary Material

## A Additional Related Work

**Soft Robot Arm Modeling.** While the pose of any point on a traditional, rigid-link robot can be fully defined by its link lengths and joint angles, the kinematics of soft robots are more complex due to their elasticity and continuum nature [1]. While Finite Element Methods (FEM) provide high-accuracy models of soft robots, their computational cost and high dimensionality make them difficult to use in control methods [2]. Instead, researchers opt for more tractable approximations for practical implementation [1]. The most common modeling approximation is the Piecewise Constant Curvature (PCC) model, a kinematic model in which the robot is approximated as a series of constant-curvature arcs. Other works use models that offer a middle ground in computational complexity. Katzschmann et al. [3] introduce an augmented rigid body formulation that models a soft robot as a rigid-bodied robot with parallel elastic actuation, which enables a dynamic model that respects PCC assumptions. Naughton et al. [2] draws from Cosserat rod theory to model soft arms as continuum elements that can bend, twist, shear, and stretch.

Another approach to dynamics modeling is through data-driven modeling methods. Bruder et al. [4] model a soft robot using the Koopman Operator, in which a projected linear state space model is fit with collected data. Jitoshio et al. [5] create a soft robot simulator that models soft growing robots as  $N$ -link pendulums with linear springs and dampers between links, and prismatic joints to model lengthening and retracting, and these parameters are fit with collected data. Bern et al. [6] train a neural network to find a mapping from motor angles to quasi-static tip position. Centurelli et al. [7] model soft robot forward dynamics with a Long Short-Term Memory (LSTM) network to find a mapping from an actuation vector to a tip position.

**Soft Robot Arm Control.** Many previous works leverage traditional, physics-based methods to control soft robot arms. Santina [8] achieves closed-loop stabilization with feedback linearization on a simulated soft inverted pendulum. Weerakoon and Chopra [9] use an energy-based controller for a soft robot swing-up task and then use a linear-quadratic regulator controller for stabilizing the soft robot in the upright position. Bruder et al. [4] use model predictive control with a Koopman model to perform tip trajectory tracking for a soft robot. Grube et al. [10] perform soft robot arm trajectory following with a kinematic controller and a dynamic controller, and they find that the dynamic controller achieves higher accuracy and robustness than the kinematic controller but also requires more computational resources.

Another method for soft robot arm control is training a neural network with large amounts of data. Bern et al. [6] train a neural network to approximate the forward dynamics of a soft robot arm, and then perform quasi-static trajectory following by using gradient-based optimization with this learned model. Thuruthel et al. [11] train a recurrent neural network to model the forward dynamics of a soft robot and then use the learned model and trajectory optimization to create open-loop trajectories. Next, they test these open-loop trajectories on the real robot and use this data to train a neural network in a supervised fashion to be used as a closed-loop predictive controller. Qiuxuan et al. [12] fit a soft robot dynamics model with a multi-layer perceptron and then train a control policy with deep Q-learning.

**Reinforcement Learning for Controlling Physical Robots.** An alternative paradigm for training robot control policies using reinforcement learning involves learning from both simulation and real-world data. Bousmalis et al. [13] first train a control policy in simulation and then fine-tune the policy on a real robot. Rusu et al. [14] use real-world data to train a generator network that transforms simulated images into real images so that the policy can learn from more realistic observations. While this paradigm can reduce the sim-to-real gap, we do not use this approach for our problem because collecting large amounts of real-world data would be time-intensive and result in significant degradation to the hardware.

There is a growing interest in applying reinforcement learning to soft robot arm control, with existing works primarily focusing on trajectory following at relatively slow speeds [2, 7, 11, 15, 16]. Naughton et al. [2] focus on additional tasks that require maneuvering between structured obstacles. Similarly, we achieve tasks that require reasoning about objects in the robot’s environment. Our work differs from previous implementations of RL for soft robot arms because we focus on achieving tasks that require high-speed motion and do not need a predefined motion plan.

## B System Identification Method, Results, and Verification

We use measurements from the physical hardware to compute mass and inertia values for our robot model. Our physical soft robot arm contains internal hardware inside the tip, but otherwise is a hollow, inflated beam. Based on this, we approximate the most distal link as a solid cylinder since it contains internal hardware, and we approximate the proximal links as cylindrical shells. We measure the mass of the full robot arm and of the internal hardware, and use this to compute the weight of the distal cylinder and each of the four cylindrical shells (0.1 kg and 0.05 kg respectively). With the measured masses and the measured cylinder radius (0.038 m), we can compute the inertia for each geometry accordingly.

We fit  $\mathbf{K}$ ,  $\mathbf{D}$ , and  $\mathbf{b}$  using data from a fixed-base experiment (no use of the cart’s linear actuator). In this experiment, we send a sequence of fPAM pressure commands and measure the resulting sequence of pressure values ( $p^{1:N}$ ) as well as the resulting robot trajectory ( $\theta^{1:N}$ ). From these values, we can compute velocities ( $\dot{\theta}^{1:N}$ ) and accelerations ( $\ddot{\theta}^{1:N}$ ) via 4th-order finite differencing.

Using these values, we compute the joint torques  $\tau^k$  at timestep  $k$  via inverse dynamics [17]:

$$\tau^k = \mathbf{M}(\theta^k)\ddot{\theta}^k + \mathbf{C}(\theta^k, \dot{\theta}^k)\dot{\theta}^k + \mathbf{g}(\theta^k), \quad (2)$$

where  $\mathbf{M}$ ,  $\mathbf{C}$ , and  $\mathbf{g}$ , are functions that compute the mass matrix, Coriolis terms, and torques due to gravity, respectively.

To fit model parameters  $\mathbf{K}$ ,  $\mathbf{D}$ , and  $\mathbf{b}$ , we pre-compute ( $\tau^{1:N}$ ) using Eq. 2, then we fit model parameters using least squares on our definition of joint torques in Eq. 1:

$$\min_{\mathbf{K}, \mathbf{D}, \mathbf{b}} \sum_{k=1}^N \| -\mathbf{K}\theta^k - \mathbf{D}\dot{\theta}^k + \mathbf{b}p^k - \tau^k \|^2. \quad (3)$$

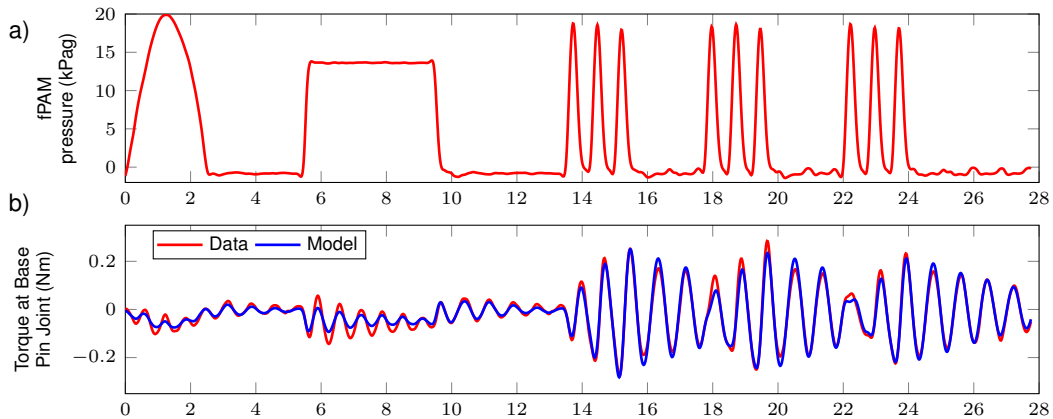


Figure 9: Results of the system identification fitting procedure. a) Input control signal into the system b) Comparison of joint torques computed from experiment data (red) versus our model with fitted parameters (blue). The RMS error is 0.02 Nm, which is small compared to the torque values in the plot.

Fig. 9 shows the results of the system identification procedure. Fig. 9a shows the sequence of fPAM commands sent to the physical robot. It includes a slow sweep of pressures to capture quasi-static behavior, a square wave to capture the step-response, and a series of high-frequency sine waves to

capture swinging dynamics. The fPAM commands were sent at 60 Hz for roughly 28 s, resulting in about 1600 timesteps of measurements. Fig. 9b shows the results of the least squares fitting. We overlay the joint torques derived from measured data (Eq. 2) versus the one computed using estimated parameters (Eq. 1). The RMS error is 0.02 Nm, which is small compared to the actual torque values (-0.28 to 0.25 Nm), showing good agreement between our model and the measurements.

The numerical values for stiffness, damping, and our control mapping are below:

$$\begin{aligned} \text{diag}(\mathbf{K}) &= (0.8385, 1.5400, 1.5109, 1.2887, 0.4347) \text{ Nm/rad} \\ \text{diag}(\mathbf{C}) &= (0.0178, 0.0304, 0.0528, 0.0367, 0.0223) \text{ Nm*s/rad} \\ \mathbf{b} &= (0.0247, 0.0616, 0.0779, 0.0498, 0.0268) \text{ Nm/psi} \end{aligned}$$

To demonstrate that our dynamic model captures our soft robot arm’s behaviors, we compare the tip trajectories of the real robot with a simulation that uses our fit model. The fPAM command sequence sent to each system is a sinusoid with a frequency not used in the system identification procedure. Fig. 10 shows the comparison of tip position over time. The tip y-coordinate ranges from -10 to 7 cm with an RMS error of 2 cm. The tip z-coordinate ranges from 51 to 52 cm with an RMS error of 0.3 cm. The average distance between the simulated and real robot tip positions across all time steps is 2 cm. We find that this model fidelity is sufficient for the tasks demonstrated in our work.

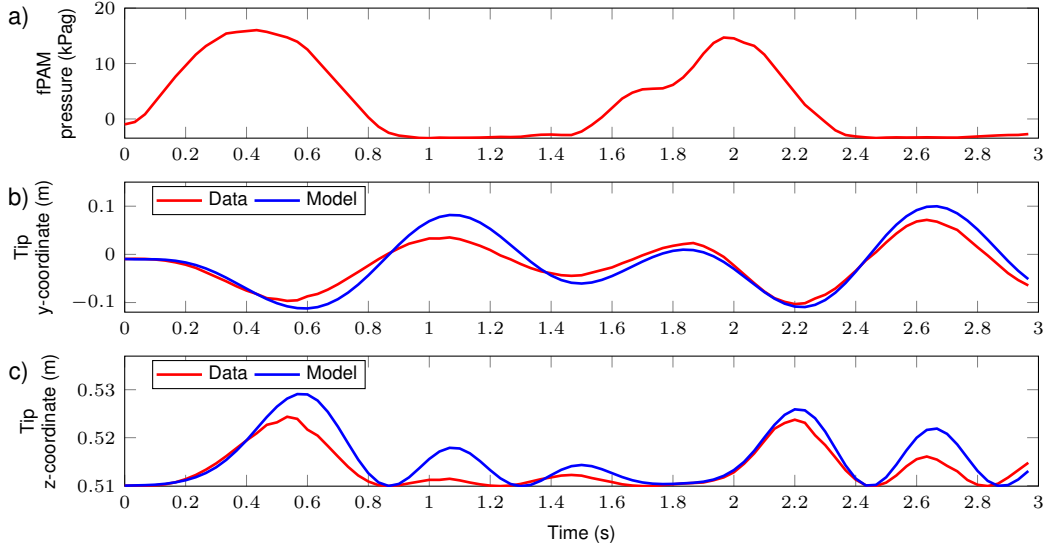


Figure 10: Evaluation of fit model. a) Input control signal into the system that is different than the original system identification experiment. b) Comparison of tip y-coordinate between experiment data versus a simulation with the fit model. The RMS error is 2 cm. c) Comparison of tip z-coordinate between experiment data versus a simulation with the fit model. The RMS error is 0.3 cm.

## C RL Framework Details

**Problem Definition for Reinforcement Learning.** We formulate the soft robot arm control task as a reinforcement learning problem. This is commonly modeled as a Markov Decision Process (MDP) given by  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ , where  $\mathcal{S}$  is the state space,  $\mathcal{A}$  is the action space,  $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  is the transition function, and  $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  is the reward function.  $\mathcal{P}(s_{t+1}|s_t, a_t)$  gives the probability of the agent transitioning from state  $s_t$  to  $s_{t+1}$  when it takes the action  $a_t$ .  $\mathcal{R}(s_t, a_t, s_{t+1})$  gives the reward  $r_t$  the agent receives when it transitions from state  $s_t$  to  $s_{t+1}$  when it takes the action  $a_t$ . The agent’s goal is to maximize the return  $R_t = \sum_{k=t}^{\infty} \gamma^{k-t} r_k$ , which is the total discounted reward from timestep  $t$  onwards, where  $\gamma \in [0, 1]$  is the discount factor that defines how much the agent favors near-term rewards over far-term rewards.

This formulation assumes a fully observable MDP, but in many real-world robotics control problems, the full state of the robot cannot be captured. These problems can be modeled as a Partially-

101 Observable Markov Decision Process (POMDP), where the agent receives observations from an ob-  
 102 servation model  $\mathcal{O}(o_t|s_t, a_t)$ . In this setting, the agent cannot observe the full state at each timestep.  
 103 Common solutions to this problem include stacking a history of observations [18] or compressing  
 104 the history into a hidden state through the use of recurrent neural networks [19, 20]. These solutions  
 105 have the added benefit of access to additional temporal information.

106 **Architecture and Algorithm.** Our control policy consists of a multi-layer perception (MLP) fol-  
 107 lowed by a Long Short-Term Memory (LSTM) layer. The MLP has 3 layers (256, 128, and 64  
 108 hidden units) connected with Exponential Linear Unit (ELU) layers. The output of this MLP is  
 109 concatenated with the original input, which is then fed into an LSTM layer with 256 hidden units  
 110 followed by a Layer Normalization layer. The policy is fed normalized observations (subtract the  
 111 mean and divide by the standard deviation of each dimension). It outputs actions in  $[-1.0, 1.0]$ ,  
 112 which are then scaled to be in the appropriate range for each action dimension. We train the policy  
 113 with the Proximal Policy Optimization (PPO) algorithm [21], using a highly-optimized GPU imple-  
 114 mentation called rl\_games [22], which uses vectorized observations and actions for faster training.

## 115 D Domain Randomization Details

116 We implement domain randomization by applying additive Gaussian noise of  $N(0, \sigma_{\text{obs}})$  for all  
 117 elements of the observation  $\mathbf{o}$  and additive Gaussian noise of  $N(0, \sigma_{\text{act}})$  for all elements of the  
 118 action  $\mathbf{a}$ , where  $\sigma_{\text{obs}} \in \mathbb{R}$  and  $\sigma_{\text{act}} \in \mathbb{R}$  are the observation and action noise parameters. We also  
 119 apply uniform scaling of  $U(1 - \epsilon_{\text{dyn}}, 1 + \epsilon_{\text{dyn}})$  to all elements of the dynamics parameters  $\mathbf{K}, \mathbf{D}, \mathbf{b}$ .  
 120 Note that the observation noise and action noise are applied to the normalized observation and the  
 121 unscaled action (actions in the range  $[-1.0, 1.0]$ ). This ensures that the scale of the noise relative to  
 122 the original value is consistent, so all components are affected similarly.

## 123 E Experimental Setup

124 To demonstrate our RL framework on physical hardware, we design and build an experimental  
 125 setup featuring a soft, inflated-beam robot with a mobile base (Fig. 1). The central component is a  
 126 computer running the control policy. It sends pressure commands to the soft robot, sends velocity  
 127 commands to the linear actuator, and receives state measurements from a motion capture system.  
 128 Despite not having a GPU, policy inference takes under 10 ms running on this computer, which  
 129 enables real-time planning and control.

130 **Inflated-Beam Robot.** The total length and weight of the soft robot arm is 44 cm and 0.12 kg,  
 131 respectively. The main body has a radius of 3.8 cm. As described by Naclerio and Hawkes [23],  
 132 the beams forming the main body and the fPAM are constructed by forming tubes with bias-cut,  
 133 woven fabric. The bias-cut orients the fabric fibers such that the tubes become shorter and wider  
 134 when pressurized. This results in the shortening during pressurization. We use two pressure regu-  
 135 lators (Festo VPPI) to control the pressure in the main body and fPAM, and each regulator also has  
 136 a built-in pressure sensor. The main body is held at a constant pressure (0.4 kPag), and the fPAM  
 137 is commanded to varying pressures (-0.7 to 20 kPag). The lower body pressure reduces opposition  
 138 to fPAM actuation while having enough pressure to maintain its cylindrical shape. The fPAM com-  
 139 mand bounds were determined empirically with the following principles: (1) the minimum pressure  
 140 must show the fPAM visibly deflated (2) the maximum pressure must cause maximum contraction  
 141 of the fPAM (3) the range of pressure commands should be small to increase tracking performance  
 142 of the pressure regulator. Our central computer sends pressure commands and receives pressure  
 143 measurements via serial communication with a microcontroller (Teensy 3.6).

144 **Mobile Base.** We utilize a cart on a belt-driven linear actuator (Igus ZLW-1040B) that acts as a  
 145 mobile base for our soft robot arm. The base of the soft robot arm is directly attached to the cart,  
 146 which slides along the actuator rails within a 0.6 m range. We use the linear actuator’s “Velocity  
 147 Mode”, which requires setting an acceleration value followed by sending velocity targets over time.

Our central computer communicates with the linear actuator over Transmission Control Protocol (TCP).

**Motion Capture Sensing.** To simplify the hardware and sensing scheme for our robot, we use a motion capture system (OptiTrack with Flex 13 cameras) to measure the robot’s current configuration. We place five sets of markers equally spaced along the soft robot arm, and a sixth set on the sliding cart. For each set of markers, the motion capture system provides the position and orientation in the global frame. We also use motion capture to measure task-specific observations  $\mathbf{o}_{\text{task}}$ .

**Simulation and Policy Learning Details.** We simulate the soft robot arm using Isaac Gym [24], a high-performance simulator that leverages GPU parallelization to simulate thousands of robots simultaneously. Using one NVIDIA RTX 3090 GPU, our simulation runs at 18,000 FPS (each frame is one action step with a control timestep of 33 ms) by running 4,096 environments in parallel. The simulation timestep and control frequency are two important parameters to determine. Our real, physical system runs at a 30 Hz control frequency (most of this time is spent communicating with the sensors to measure the current state), so we run the control policy at 30 Hz in simulation accordingly. However, simulation often requires smaller timesteps to ensure numerical stability. We found that simulating the vine robot at 1200 Hz (0.833 ms timestep) is sufficiently stable.

For modeling the fPAM pressure, we found that the filtering parameter  $\alpha$  was different for inflation ( $a_p > p$ ) and deflation ( $a_p < p$ ). Thus, we use  $\alpha_{\text{inflate}} = 0.86$  and  $\alpha_{\text{deflate}} = 0.81$  for modeling inflation and deflation, respectively. For modeling the cart dynamics, we use  $k_v = 0.3$ ,  $k_v = 30$ , and an action delay of 1 control timestep (33 ms). For domain randomization, we use  $\sigma_{\text{obs}} = 0.001$ ,  $\sigma_{\text{act}} = 0.001$ , and  $\epsilon_{\text{dyn}} = 0.001$ .

We train all learned policies with a learning rate of 3e-4, a discount factor  $\gamma$  of 0.99, and a PPO clipping interval  $\epsilon_{\text{clip}}$  of 0.2. We also normalize the observations, values, and advantages, and we train the policy with 4 epochs per policy update. Using a horizon length of 16 (number of timesteps between updates for each robot, with all robots running in parallel), 4096 simulated robots, and a maximum of 500 update iterations, the approximate number of training timesteps is 32M steps ( $16 \times 4096 \times 500$ ). Training takes about 40-80 minutes on an NVIDIA RTX 3090 GPU, which is substantially less time than Elastica’s  $\sim 11$  hour RL training [2].

## F Trajectory Optimization Planning and Control Method

We use trajectory optimization to determine a reference trajectory (control trajectory  $\mathbf{u}^{1:N-1}$  and resulting state trajectory  $\mathbf{x}^{1:N}$ ) that brings the tip of the soft robot arm to the goal position within a fixed time horizon  $N$ . We define the state vector  $\mathbf{x} := (y_{\text{cart}}, \dot{y}_{\text{cart}}, \theta_1, \dot{\theta}_1, \dots, \theta_5, \dot{\theta}_5)$  and control input  $\mathbf{u} := (F_{\text{cart}}, a_p)$  where  $F_{\text{cart}}$  is the force applied to the cart and  $a_p$  is the fPAM pressure. We solve the following optimization problem to compute the reference trajectories.

$$\begin{aligned}
& \min_{\mathbf{x}^{1:N}, \mathbf{u}^{1:N-1}} \sum_{k=1}^N \|\mathbf{x}^k - \bar{\mathbf{x}}^k\|_{Q^k}^2 + \sum_{k=1}^{N-1} \|\mathbf{u}^k\|_{R^k}^2 \\
& \text{s.t.} \quad \mathbf{x}^{k+1} = f(\mathbf{x}^k, \mathbf{u}^k), \quad k = 1, \dots, N-1, \\
& \quad \mathbf{u}_{\min} \leq \mathbf{u}^k \leq \mathbf{u}_{\max}, \quad k = 1, \dots, N-1, \\
& \quad y_{\text{cart-min}} \leq y_{\text{cart}}^k \leq y_{\text{cart-max}}, \quad k = 1, \dots, N-1, \\
& \quad \dot{y}_{\text{cart-min}} \leq \dot{y}_{\text{cart}}^k \leq \dot{y}_{\text{cart-max}}, \quad k = 1, \dots, N-1, \\
& \quad g(x^N) = 0
\end{aligned} \tag{4}$$

where  $y_{\text{cart}}^k$  and  $\dot{y}_{\text{cart}}^k$  are the first and second elements of  $\mathbf{x}^k$  respectively. The objective function is a quadratic cost on deviation from a nominal state trajectory  $\bar{\mathbf{x}}^{1:N}$  with weight matrices  $Q^{1:N}$  and a quadratic cost on control effort with weight matrices  $R^{1:N-1}$ . The first constraint is for dynamic feasibility, the second is for control limits, the third is for cart position and velocity limits, and the

fourth is for the tip position to reach the goal position at the final timestep. We set each  $\bar{\mathbf{x}}^k$  of the nominal trajectory to be a pose that bends to the left (with zero velocity) since all goal positions require bending to the left. Because we use a multi-link rigid body approximation as our dynamic model, we have a broader array of options for our simulator and optimizer choice. For this work, we chose Dojo [25] for its numerical stability (since we have stiff equations of motion and would like to take larger time steps), and we used its associated trajectory optimization package which implements iterative Linear Quadratic Regulator with Augmented Lagrangian methods.

The numerical values for  $Q^k$  and  $R^k$  for  $k = 1, \dots, N - 1$  are:

$$\begin{aligned} \text{diag}(Q^k) &= (10, 1, 10, 1, 10, 1, 10, 1, 10, 1, 10, 1) \\ \text{diag}(R^k) &= (1, 1) \end{aligned}$$

The numerical values for  $Q^N$  are:

$$\text{diag}(Q^N) = (100, 10, 100, 10, 100, 10, 100, 10, 100, 10, 100, 10)$$

Solving the trajectory optimization problem took between 1-13 minutes, depending on the target position. Re-planning in real-time, (e.g. with model predictive control), was not possible due to the optimization solve time, so instead we used a simple but fast tracking controller that adds a feedback term to the reference action trajectory based on deviation from the reference tip trajectory:

$$a_{\text{cart-vel}}^k = a_{\text{cart-vel ref}}^k + k_y(y_{\text{tip ref}}^k - y_{\text{tip}}^k) \quad (5a)$$

$$a_p^k = a_{p \text{ ref}}^k + k_z(z_{\text{tip ref}}^k - z_{\text{tip}}^k). \quad (5b)$$

Reference action trajectories  $a_{\text{cart-vel ref}}^{1:N-1}$  as well as  $a_{p \text{ ref}}^{1:N-1}$  and reference tip trajectories  $y_{\text{tip ref}}^k$  and  $z_{\text{tip ref}}^k$  are extracted from the optimal solution for  $\mathbf{x}^{1:N-1}$  and  $\mathbf{u}^{1:N-1}$ . The current tip position  $(y_{\text{tip}}^k, z_{\text{tip}}^k)$  is measured with motion capture, and  $k_y$  and  $k_z$  are controller gains. The adjusted actions are clamped to be within the action limits for the cart and fPAM and then sent to the physical hardware. During trajectory optimization, we use conservative constraints on cart velocity and fPAM pressure so that the tracking controller has margin to exceed the reference control before needing to be clamped within actuator limits. We empirically chose gains that improved performance for one of the more difficult target positions, and then used this for all other experiment runs. The gains used were  $k_y = 0.1$  and  $k_z = 5.0$ .

As with our policy trained with RL, we require careful but simple strategies to achieve sim-to-real transfer. First, the trajectory constraints allow us to respect state and control limits (i.e. cart position, velocity, and acceleration as well as fPAM pressure). We note that we indirectly enforce constraints on cart acceleration by constraining  $F_{\text{cart}}$ . Second, the use of a tracking controller allows us to overcome minor model errors, similar to how domain randomization during policy training produces robustness to model errors. Finally, we did not model actuator dynamics when solving the reference trajectory for simplicity, but this caused a delay between the reference and actual tip trajectory that could not be overcome by the tracking controller alone. We address this by extending the state and control reference trajectories for a few extra timesteps and filling the new elements with  $x^N$  and  $u^{N-1}$ , respectively. We hypothesize that this adjustment handles the sim-to-real gap introduced by actuator latency and response times.

## G Comparison to PID Control for Free Space Target Reaching Task

We compare our learned policy to a proportional-integral-derivative (PID) controller on the free space target reaching task. This serves as a baseline controller that does not reason about leveraging swinging, and we show that it is largely unsuccessful in this task.

We use two separate PID controllers to achieve tip-position control. We use cart actuation to drive the y-coordinate error towards zero, and we use fPAM actuation to drive the z-coordinate error towards zero. We acknowledge that the fPAM actuation also affects the tip y-coordinate, but find



that the y-coordinate PID controller is able to account for this disturbance. Below are the equations used to compute the PID control commands (actions) for cart velocity and fPAM pressure:

$$a_{\text{cart-vel}} = -K_{p,y}e_y - K_{d,y}\dot{e}_y - K_{i,y}\int e_y dt$$

$$a_p = -K_{p,z}e_z - K_{d,z}\dot{e}_z - K_{i,z}\int e_z dt$$

where  $e_y = y_{\text{tip}} - y_{\text{target}}$  and  $e_z = z_{\text{tip}} - z_{\text{target}}$  are the y-error and z-error of the tip position, respectively.  $K_*$  are the PID controller gains. We took a manual approach to tuning our PID gains that is similar to the Ziegler-Nichols method. Our final gains were  $K_{p,y} = 1$  and  $K_{p,z} = 20$ . We found that derivative and integral gains had little effect on overall performance, and that increasing these gains led to instability, so we ultimately set these to zero.

We run this PID controller for the same 54 target positions discussed in Sec. 4, and the robot achieves a success rate of 17% across the 54 target positions. With this control method, the robot is not able to reach any of the target positions with a z-coordinate of 0.6 m or greater; the maximum tip z-coordinate reached across all PID control experiments was 0.56 m. The PID controller limits the robot's workspace because it aims to greedily reduce tip position error and does not incorporate any reasoning about swinging, significantly reducing its ability to reach higher target positions. We illustrate this in Fig. 11. Using our learned policy, the robot is able to reach the target tip position by building up momentum over multiple swings to increase its tip height. In contrast, using the PID controller, the robot is unable to reach the target tip position, as it simply moves directly towards the target and fails to bring the tip high enough. This demonstrates that the free space target reaching task is not possible without well-timed actuation that leverages swinging motion.

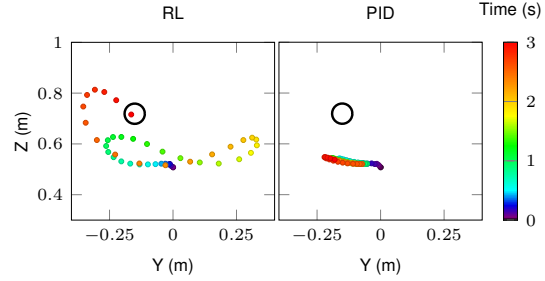


Figure 11: Comparison of learned policy (RL) vs. PID control (PID) for reaching a target tip position in free space. The black circle is centered at the target position with a radius of 4 cm. The RL control policy is able to perform high-speed swinging behavior to reach the target position. The PID control is unable to track the target position because it greedily approaches the target position directly.

## References

- [1] I. Robert J. Webster and B. A. Jones. Design and kinematic modeling of constant curvature continuum robots: A review. *The International Journal of Robotics Research*, 29(13):1661–1683, 2010.
- [2] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola. Elastica: A compliant mechanics environment for soft robotic control. *IEEE Robotics and Automation Letters*, 6(2):3389–3396, 2021.
- [3] R. K. Katzschmann, C. D. Santina, Y. Tshimitsu, A. Bicchi, and D. Rus. Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model. In *IEEE International Conference on Soft Robotics*, pages 454–461, 2019.
- [4] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan. Modeling and control of soft robots using the koopman operator and model predictive control. *Robotics: Science and Systems*, 2019.
- [5] R. Jitoshio, N. Agharese, A. M. Okamura, and Z. Manchester. A dynamics simulator for soft growing robots. In *IEEE International Conference on Robotics and Automation*, pages 11775–11781, 2021.
- [6] J. M. Bern, Y. Schnider, P. Banzet, N. Kumar, and S. Coros. Soft robot control with a learned differentiable model. In *IEEE International Conference on Soft Robotics*, pages 417–423, 2020.
- [7] A. Centurelli, L. Arleo, A. Rizzo, S. Tolu, C. Laschi, and E. Falotico. Closed-loop dynamic control of a soft manipulator using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):4741–4748, 2022.
- [8] C. D. Santina. The soft inverted pendulum with affine curvature. In *IEEE Conference on Decision and Control*, pages 4135–4142, 2020.
- [9] L. Weerakoon and N. Chopra. Swing up control of a soft inverted pendulum with revolute base. In *IEEE Conference on Decision and Control*, pages 685–690, 2021.
- [10] M. Grube, J. C. Wieck, and R. Seifried. Comparison of modern control methods for soft robots. *Sensors*, 22(23):9464, 2022.
- [11] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi. Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators. *IEEE Transactions on Robotics*, 35(1):124–134, 2019.
- [12] W. Qiuxuan, Y. Gu, Y. Li, B. Zhang, S. Chepinskiy, J. Wang, A. Zhilenkov, A. Krasnov, and S. Chernyi. Position control of cable-driven robotic soft arm based on deep reinforcement learning. *Information*, 11(6):310, 2020.
- [13] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *IEEE International Conference on Robotics and Automation*, pages 4243–4250, 2018.
- [14] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 262–270, 2017.
- [15] S. Satheeshbabu, N. K. Uppalapati, T. Fu, and G. Krishnan. Continuous control of a soft continuum arm using deep reinforcement learning. In *IEEE International Conference on Soft Robotics*, pages 497–503, 2020.



- 297 [16] N. K. Uppalapati, B. Walt, A. J. Havens, A. Mahdian, G. Chowdhary, and G. Krishnan. A  
298 berry picking robot with a hybrid soft-rigid arm: Design and task space control. In *Robotics:  
299 Science and Systems*, page 95, 2020.
- 300 [17] K. M. Lynch and F. C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge  
301 University Press, USA, 1st edition, 2017. ISBN 1107156300.
- 302 [18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller.  
303 Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- 304 [19] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi,  
305 R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang,  
306 L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen,  
307 V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff,  
308 Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap,  
309 K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using  
310 multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.
- 311 [20] P. Zhu, X. Li, P. Poupart, and G. Miao. On improving deep reinforcement learning for pomdps.  
312 *arXiv preprint arXiv:1704.07978*, 2017.
- 313 [21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization  
314 algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- 315 [22] D. Makoviichuk and V. Makoviychuk. rl-games: A high-performance framework for rein-  
316 forcement learning. [https://github.com/Denys88/rl\\_games](https://github.com/Denys88/rl_games), 2022.
- 317 [23] N. D. Naclerio and E. W. Hawkes. Simple, Low-Hysteresis, Foldable, Fabric Pneumatic Arti-  
318 ficial Muscle. *IEEE Robotics and Automation Letters*, 5(2):3406–3413, 2020.
- 319 [24] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin,  
320 A. Allshire, A. Handa, and G. State. Isaac gym: High performance gpu-based physics simula-  
321 tion for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.
- 322 [25] T. Howell, S. Cleac’h, J. Kolter, M. Schwager, and Z. Manchester. Dojo: A differentiable  
323 physics engine for robotics. *arXiv preprint arXiv:2203.00806*, 2022.