Reinforcement Learning Control of Shape Memory Alloy Based Soft Robotic Platform

Guo Ning (Andrew) Sue

CMU-RI-TR-25-16

May 7, 2025

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Professor Carmel Majidi, *chair*
Professor Guanya Shi
Cornelia Bauer

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my parents, who sowed the seed of curiosity in me to explore and enjoy this wonderful world, and to all who have been kind to me

Abstract

Soft robots enable safe, adaptive interaction in environments where rigid systems are inadequate, but their continuous, deformable nature makes modeling and control challenging. This work presents a data-driven framework that uses supervised learning and reinforcement learning to model the kinematics of a soft robot, develop task-driven control policies, and form safety filters. Using experimental data from a custom silicone-based, shape memory alloy (SMA)-actuated limb, we train an expressive kinematics model that captures the nonlinear, temperature-dependent behavior of SMA and silicone materials. This model is integrated into a reinforcement learning setup, allowing the robot to acquire control strategies to achieve tasks without relying on computationally expensive simulations. Furthermore, by leveraging the model as part of the learned policy, we preserve performance while reducing training time. To ensure safety, the framework uses a novel reward formulation to enable the training of a safety filter, which rejects actions that would lead the system into states that could be damaging to itself or others, thereby protecting both the robot and its environment from unsafe actions. Experimental results demonstrate robust task performance, good motion prediction, and adherence to safety constraints.

Acknowledgments

First and foremost, I would like to thank everyone on my thesis committee—Prof. Majidi, Prof. Shi, and Cornelia Bauer—for their time, advice, support, and help. I deeply appreciate it. I would also like to thank everyone in the Soft Machines Lab for their support and assistance whenever I needed it, especially Richard Desatnik. I am especially grateful for his generosity in allowing me to drive his robot to the point of breaking and for his help in fixing the soft robotic platform. As the designer and primary creator of the platform, he has always been patient with whatever trouble I may have caused. My deepest gratitude goes to Yogita Choudhury, who first introduced me to the field of robot safety. She patiently answered my sometimes naïve questions and endured my moments of frustration. I would also like to thank Junzhe Hu for his help in running experiments and for never abandoning me while I tackled particularly difficult bugs, even in the middle of the night. Lastly, I thank all my professors and teachers throughout my academic journey. Without what they have taught me, I would never have come this far. You all have my eternal gratitude.

Funding

This work was supported in part by the National Science Foundation under Grant IIS-1925360.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Soft robots, with their soft and compliant nature, hold great potential for tasks where traditional rigid robots fall short. Their flexibility makes them less likely to damage their surroundings and potentially safer to operate around humans. This advantage has been demonstrated in successful proof-of-concept applications such as medical surgeries, underwater manipulation of fragile coral reefs, and soft exosuits. However, many promising applications for soft robots, such as search and rescue, delicate medical procedures, and space robotics, have yet to be fully realized.

Due to their soft and compliant nature, a major challenge lies in accurately controlling the movements of soft robots. Unlike rigid robots, such as robotic arms, quadrupeds, or humanoids, where joint positions can be easily determined using commercial encoders, soft robots typically have a continuum structure. This makes it difficult for the robot to sense its own configuration in space without external tools like computer vision. Additionally, the actuators that power soft robots often exhibit complex, hard-to-model behaviors, unlike conventional motors. While methods to simulate and predict soft robot behaviors do exist [12, 13], they are typically extremely computationally intensive.

Meanwhile, methods such as model predictive control and, more recently, reinforcement learning have been highly successful in enabling traditional robots, such as legged robots, to accomplish complex tasks like walking on uneven terrain and even performing parkour. This success largely stems from the fact that the dynamics of these traditional robots can be easily predicted analytically, which allows for the

development of high-speed, high-fidelity simulators. In these simulators, various control strategies can be tested or autonomously learned faster than in real time. For soft robots, various methods have been explored to create dynamic or kinematic models suitable for model predictive control, for example [6, 17, 31]. However, most of these methods have been tested only on traditional pneumatic-based actuators or for simple planar motions. Unfortunately, for soft robots with more complex dynamics, such as those using electrothermal actuation or those that exhibit high degrees of hysteresis, achieving computationally fast and high-fidelity simulation for control remains extremely difficult.

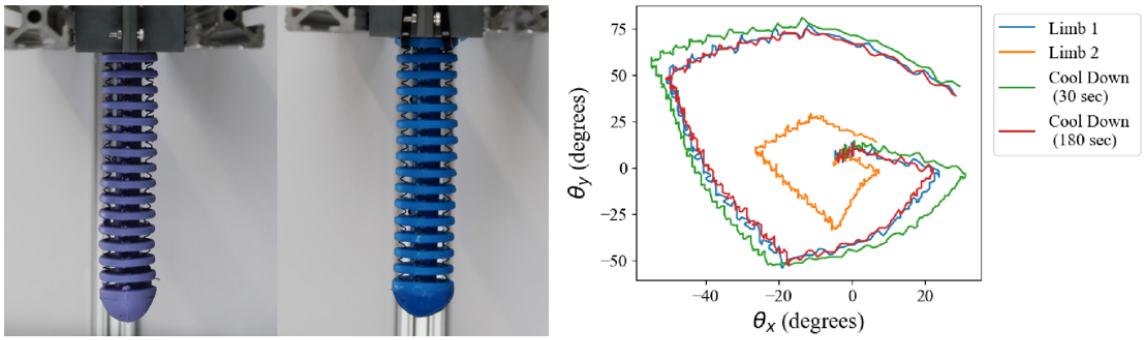
To enable soft robots to benefit from simulation and predictive control similar to traditional robots, this work aims to use machine learning methods to learn the kinematics of the robotic system, and then leveraging the learned kinematics to learn actions that achieve arbitrary tasks.

To that end, this work proposes a three-part data-driven framework that aims to answer three crucial questions regarding soft robotic control:

1. How can we learn an expressive and easily generalizable kinematics model of a soft robotic system?
2. How can we leverage the learned kinematic model to create a controller so that the soft robot can achieve a task—in other words, how can we learn a task-achieving policy?
3. How can we ensure that the actions from the task-achieving policy do not violate any safety constraints?

The framework is validated on a custom shape memory alloy (SMA)-driven, silicone-based soft robotic limb platform. This platform is chosen because it represents the challenging nature of modeling soft robots.

Due to its SMA-based actuation, which relies on heat generated through Joule heating, and the thermal properties of silicone, combined with the fact that the limb system is custom-made using standard hobby-grade tools and materials, the kinematics of the system is highly nonlinear, difficult to predict, and varies from limb to limb even when constructed with the same components and materials. Fig. 1.1 shows the soft robotic limb on which the proposed method is tested, along with a visualization of the hysteresis effect. In Fig. 1.1c), all scenarios are given the same



a) Stiff Limb (Limb 2)

b) Soft Limb (Limb 1)

c) Trajectory of different limb and under different cool down conditions

Figure 1.1: The soft limb platform used for testing consists of two limbs: Limb 1, the softer limb shown in blue in panel (b), and Limb 2, the stiffer limb shown in purple in panel (a). Panel (c) illustrates the heat hysteresis effect and the differences in dynamics between the two limbs. The cool-down scenarios are conducted on Limb 1. All scenarios are given the same open-loop action trajectory. While longer cooling periods help reduce the impact of heat hysteresis, such extended cool-downs are not always feasible during task execution.

open-loop action trajectory, and their actual poses are recorded and shown. It can be seen that the trajectory recorded after cooling for 180 seconds (red) follows the original trajectory (blue) much more closely than the trajectory recorded after 30 seconds (green).

Thus, the success of the proposed framework on the chosen hardware platform serves as the first indication of a valid and generalizable control strategy for similar, and potentially any, custom soft robotic platforms

Chapter 2

Background

For the purpose of controlling soft robots, there are currently two main perspectives on how to approach the problem: physics-based optimal control and learning-based control. The method proposed in this work first creates a model, which is then used to develop a controller for a specific task. Many previous studies have demonstrated tremendous progress in each of these areas.

2.1 Learning-free Methods

2.1.1 Physics based simulator

The most straightforward and currently most accurate method for modeling soft robots is to use physics-based approaches, such as Finite Element Analysis (FEA), through commercial software like COMSOL or ANSYS. A popular FEA-based simulator tailored for soft robotics is SOFA [13, 21], known for its ease of use and real-time multiphysics capabilities. However, despite their unparalleled accuracy, these methods are often unsuitable for robotic applications due to their high computational cost.

To address this, efforts have been made to improve the computational efficiency of soft continuum robot simulations using physics-based theories. One such approach is Cosserat Rod Theory (CRT), which is implemented in the PyElastica simulator [24, 68]. A discrete version of CRT, known as Discrete Elastic Rod Theory, is used in the DisMech simulator [12], which also extends to the simulation of shell structures.

Another physics-based simulator, SoMoGym [25], is specifically built for soft robotics reinforcement learning and is based on the PyBullet physics engine [15]. However, SoMoGym represents a soft rod as a chain of discrete rigid rods connected by spring joints. While this increases computational efficiency, it significantly reduces model accuracy, particularly in capturing complex nonlinear behaviors such as hysteresis, which are common in soft robotic systems.

Furthermore, a common issue among these simulators is that the control inputs for soft limbs are defined as torques or bending moments along rods or shells. Although theoretically sound, these inputs are difficult to map to real-world hardware inputs, which are typically electrical signals sent to the actuators. For instance, in the case of shape memory alloy (SMA) actuators, while several modeling attempts have been made, none accurately describe the relationship between a given voltage input and the resulting force output. This makes such simulators extremely difficult to use for custom robotic platforms employing custom actuators.

2.1.2 Controller

Although having a model is extremely helpful for controlling soft robotic systems, it is not essential. Many optimal control methods can operate using a rudimentary dynamics model or even no model at all. For example, the classic Proportional-Integral-Derivative (PID) controller, and extensions of it, have been successfully used to control shape memory alloy (SMA)-based robotic limbs to an impressive degree [40]. In another example, [19] attempted to control an SMA-based limb using Iterative Learning Control, starting from a DIRCOL-optimized trajectory generated from a linearized model of the soft robotic limb.

Currently, one of the most advanced optimal control-based methods for controlling soft robotic limbs involves using the Koopman operator to build a model of the system, followed by Model Predictive Control (MPC) [8]. However, the authors who pioneered this approach validated their method on a pneumatically powered soft robotic limb, which is considerably easier to control due to the absence of heat hysteresis and the potential inconsistencies introduced by the custom, handmade SMA actuators employed in our robotics platform.

Another common issue with most optimal control-based methods, such as PID, is

the requirement for careful tuning of parameters and coefficients. This tuning process must be performed on the actual hardware, and using incorrect parameters during this process can potentially damage the system. Moreover, tuning is particularly tedious in SMA-based robots. Due to the heat-induced hysteresis effects, it is difficult to identify a single set of parameters that performs well under all operating conditions. Additionally, tuning requires prolonged activation of the hardware and actuators, which risks damaging the robot before optimal parameters are even found. Therefore, with the recent rise of machine learning, data-driven approaches offer a promising alternative to mitigate these challenges.

2.2 Learning-based Methods

2.2.1 Learning Forward Kinematics

Data-driven approaches and neural networks for estimating soft robotic kinematics have gained popularity in recent years. One work particularly relevant to this study is [44], which employed an LSTM network to capture the dynamics of a soft robotic limb. However, their study was limited to strictly planar motion and a single soft limb material. Other works that utilize machine learning techniques—especially those based on recurrent neural network (RNN) structures, which offer considerable advantages in handling time-series data—include [48, 62]. Additionally, architectures such as Seq2Seq, traditionally used in language translation, have been successfully adapted for dynamic prediction in soft robotics [5].

While these studies have demonstrated success in predicting soft robotic kinematics, many are restricted to single-axis planar movements or rely on external motion-capture systems or numerous embedded sensors. These dependencies introduce challenges such as complex wire management and reduced practicality for real-world deployment. In contrast, the present work aims to explore the boundaries of what neural networks can achieve by validating modern time-series models for soft robotic kinematics prediction using only a 2-axis capacitive bend sensor.

2.2.2 Learning task-achieving Policy

Many previous works have applied reinforcement learning (RL) algorithms to the control of soft robots. Some approaches train policies directly on hardware [65], while others, such as [38, 58], perform training in simulation environments like the aforementioned SoMoGym or physics-based simulators originally developed for rigid robots, such as MuJoCo [63]. Other studies, including [11], have also explored learning a forward kinematic model and subsequently training a policy using RL.

Most of these prior works use conventional RL algorithms—such as Soft Actor-Critic (SAC) [27], Deep Q-Learning (DQN) [37], Trust Region Policy Optimization (TRPO) [46], and Proximal Policy Optimization (PPO) [47]—with minimal modification. As a result, methods like [11] that incorporate a learned kinematics model do not fully leverage the model's capabilities, since many of these RL algorithms, including TRPO, do not require an explicit model to learn.

In contrast, the approach proposed in this work integrates the learned kinematic model directly into the RL pipeline by freezing its weights, thereby ensuring that the learned kinematic knowledge is effectively utilized during policy training.

2.2.3 Safety in Soft Robotics

At first glance, due to their soft and compliant nature, soft robots may appear intrinsically safe and not in need of formal safety considerations for path planning and control. However, this is not the case. Take, for example, the use of soft robots in medical surgery: many organs, membranes, and nerves are too delicate to touch, even with compliant robotic systems. These regions can be considered "unsafe" for contact. Similarly, in the SMA-powered soft robot used in this work, there are configurations that the robot should not reach, as excessive bending could potentially damage its actuators. These too represent unsafe states. Therefore, ensuring safety in the motion planning and control of soft robots is far from trivial.

Unfortunately, research focused specifically on safety control in soft robotics has been relatively limited compared to other areas. Some pioneering efforts include [41, 45]. However, valuable insights can be drawn from safety research in other robotic domains, such as autonomous driving.

Since this work primarily explores the use of reinforcement learning (RL) to control

soft robots, existing safety strategies in RL can be broadly categorized into two groups: model-free safe reinforcement learning and model-based safe reinforcement learning.

Model-free Safe Reinforcement Learning

Model-free Safe Reinforcement Learning (MFRL) is often formulated as a Constrained Markov Decision Process (CMDP) [2]. In this framework, the conventional MDP is augmented with an additional cost signal that flags state-action pairs violating predefined constraints. By setting appropriate cost thresholds, one can derive policies that maintain a low probability of failure. Common techniques for solving CMDPs include Lagrange multiplier methods [1, 20, 59], projection-based methods [66], and penalty function strategies [26].

Another research direction within model-free reinforcement learning involves the development of a safety critic to filter out unsafe actions—a concept that also informs the approach presented in this work. For example, Safety Q-functions for Reinforcement Learning (SQRL) [51] train a safety critic to predict the likelihood of future failures and use this prediction to constrain the behavior of the primary policy. This typically involves pre-training the safety critic, followed by fine-tuning the policy on the target tasks while incorporating the learned safety constraints.

In contrast to the recovery-based approach in [60], the method proposed by [29] focuses on the simultaneous optimization of performance and safety. Their strategy refines the reward formulation introduced in [61] by embedding the safety critic directly into the reward function, thereby discouraging exploration in potentially unsafe regions during training. Additionally, [9] introduces a binary safety critic framework based on the assumption that safety can be represented as a binary property.

The formulation proposed in this work aligns with the safety critic paradigm but introduces a novel reward design to further safeguard against unsafe behaviors during both training and execution.

Model-based Safe Reinforcement Learning

Model-based safe reinforcement learning methods typically offer greater sample efficiency compared to their model-free counterparts. Recent research has integrated

CMDPs with model-based RL to reduce training-time safety violations and accelerate learning [4, 61, 67]. For instance, Safe Model-Based Policy Optimization (SMBPO) [61] leverages a learned dynamics model for planning, penalizing unsafe trajectories to avoid potential safety violations. However, this method assumes that safety breaches occur within a fixed horizon after entering irrecoverable states. As a result, uncertainties in the learned model can lead to misclassifications, where unsafe states are mistakenly considered safe.

In contrast, the approach proposed in this work does not rely on assumptions about the time horizon of irrecoverable states. Instead, it adopts a model-free strategy to ensure safety, thereby avoiding issues related to model inaccuracies and offering a more robust safety framework.

Chapter 3

Hardware

SMA robots were chosen to validate the proposed method partly because they are difficult to control due to effects such as heat-induced hysteresis—making them an ideal testbed to demonstrate the method's robustness. Additionally, SMA actuators offer the ability to mimic the muscles of soft animals efficiently and with minimal external hardware, a crucial goal in soft robotics [42]. SMA-driven systems have been used to emulate brittle stars, black bass fish, and even extinct organisms such as the pleurocystitid [14, 18, 39]. SMA materials produce mechanical work when thermally activated through a crystal structure transition from martensite to austenite during contraction [33]. Although there is energy loss due to heat, this form of actuation can be integrated into robotic hardware at relatively low voltages, powered by off-the-shelf lithium batteries, while maintaining high work density.

The SMA-based soft robots used in this work were injection molded using custom molds fabricated with a resin-based Formlabs 3D printer. To test the adaptability of the algorithm, the limbs were constructed using different elastomers. Limb 2, made from the stiffer material Smooth-Sil 945, has a modulus of 260 psi and a Shore hardness of 40A; this limb is shown in Fig. 1.1(a) [50]. Limb 1, which is more flexible and displayed in Fig. 1.1(b), is made from Mold Star 30, with a modulus of 96 psi and a Shore hardness of 30A [49].

Each limb is actuated by four thermally responsive Nitinol SMA wires from Dynalloy. This nickel-titanium alloy has an actuation temperature range of 60°C–110°C [32]. The SMA wires are embedded in pairs along the x-axis and y-axis. Since

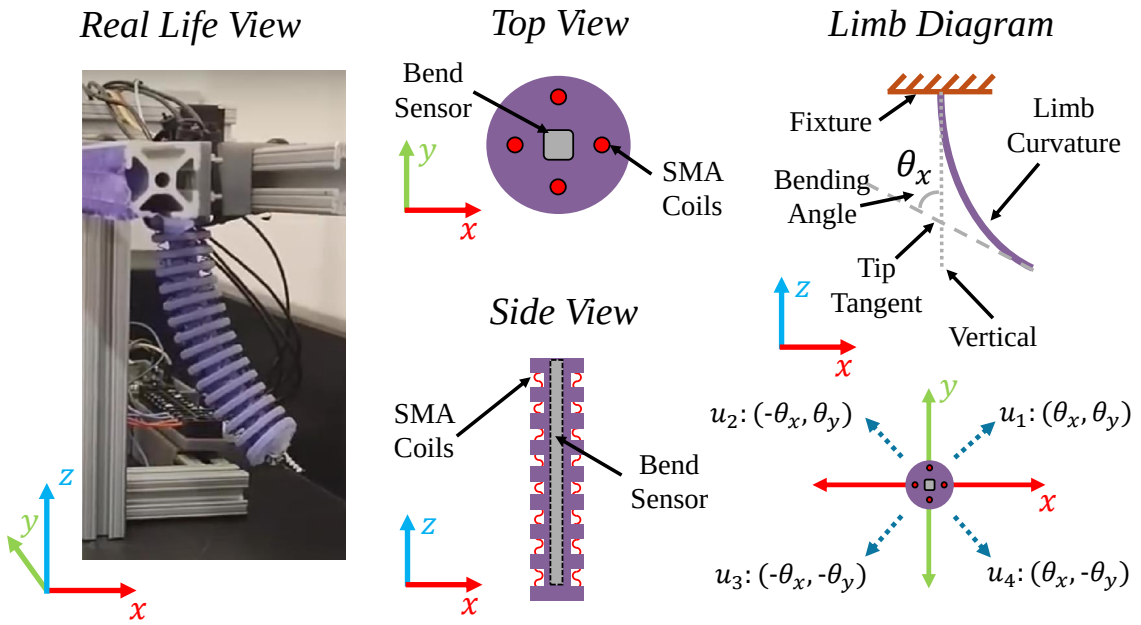


Figure 3.1: Experimental setup of the soft robotic limb. *Left*: Soft limb mounted on an aluminum fixture, actuated by SMA coils controlled via PWM through an Arduino UNO. *Middle*: Cross-sectional view (not to scale) showing the embedded two-axis bending angle sensor (gray) in the silicone limb (purple), with SMA coils at cardinal directions. *Right*: Diagram illustrating the bending angle and the four example actions used.

the actuators can only contract in one direction, pairing them allows bidirectional movement—as one SMA wire contracts, the opposing wire expands. This setup is illustrated in Fig.3.1. During Joule heating, the system is powered at 11.1 V, causing the SMA coils to contract. Since the coils are placed in the 4 cardinal directions, it enables movement of the limb in both the X and Y planes. Each SMA wire is individually controlled using four MOSFETs connected to an Arduino through Pulse Width Modulation (PWM), which collects sensor data and transmits it to a Raspberry Pi.

The end-effector location data is captured via a 2-axis capacitive bend sensor from Bend Labs [57]. This sensor monitors the angular displacement of the end effector by detecting changes in capacitance caused by mechanical strain [57]. The sensor is placed at the center of the limb and securely fixed to prevent slipping along the Z-axis.

Chapter 4

Learning Forward Kinematics

4.1 Preliminaries

The first step in our proposed method for training a controller on the SMA-based soft robotics platform is to create a simulator that reinforcement learning algorithms can leverage. To achieve this, we first learn a kinematics network that predicts the future state of the soft robotic limb, given its current state and the current action, in a discrete-time setting.[1]

4.1.1 State Definition

We define the state x of the soft robotic limb system as a vector of four elements: the bending angles along the X and Y axes, and the bending angle velocities along the X and Y axes, denoted as θ_x, θ_y, $\dot{\theta}_x$, and $\dot{\theta}_y$, respectively:

$$x = [\theta_x, \theta_y, \dot{\theta}_x, \dot{\theta}_y]. \tag{4.1}$$

We are interested in predicting x_{t+1} using the following relation:

$$x_{t+1} = f(x_t, \ldots, x_{t-n}, u_t) \tag{4.2}$$

[1]This section is a collaboration with Richard Desatnik as he designed and built the hardware platform to validate the proposed method on.

where $f(\cdot)$ denotes the underlying dynamics of the limb system, and n represents the number of previous time steps considered.

In our proposed framework, $f(\cdot)$ is approximated by a function $\phi(\cdot)$, which includes a neural network, and is expressed as:

$$\begin{aligned} \hat{f}(x_t, \ldots, x_{t-n}, u_t) &= x_t + \phi(x_t, \ldots, x_{t-n}, u_t) \\ \Delta x_t &= \phi(x_t, \ldots, x_{t-n}, u_t) \end{aligned} \tag{4.3}$$

In other words, unlike related prior works [10, 44], which directly predict the next state, we instead predict the change in state from the current time step to the next, given a history of past states and the current action. We adopt this approach because learning the difference helps eliminate bias arising from variations in initial conditions during data collection and testing, thereby improving the generalizability of the approximated dynamics.

4.1.2 Action Definition

The soft limb platform actuates via Joule heating of attached SMA coil tendons, which are individually controlled using PWM signals—resulting in a total of four PWM input signals. However, because compression of the limb is undesirable, a maximum of only two neighboring SMA coils can be actuated simultaneously. To enforce this constraint and simplify the input action space, we abstract the action input into a range of bounded positive and negative values for the X and Y axes individually:

$$\begin{aligned} u &= [u^x, u^y] \\ u^x, u^y &\in [-10, 10] \end{aligned} \tag{4.4}$$

For example, $u^x = 10$ denotes a 100% duty cycle for the SMA coil that actuates the limb in the $+X$ direction, while $u^x = -10$ denotes a 100% duty cycle for the SMA coil that actuates the limb in the $-X$ direction. The variables u^x and u^y are often referred to as the "X throttle" and "Y throttle," respectively.

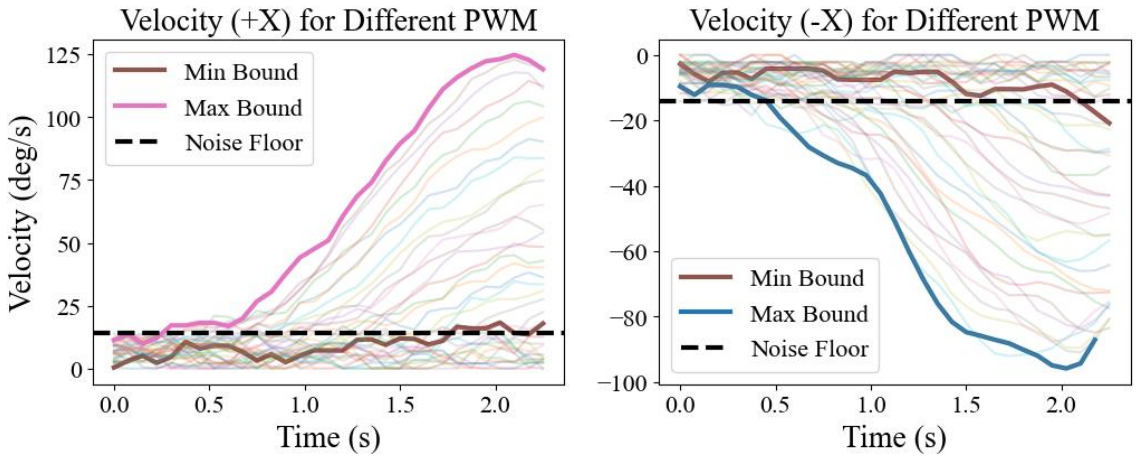


Figure 4.1: One example of how input bounds are determined is shown for the X-axis. The lower bounds correspond to the PWM values that produce the brown curves—the first curves to rise above the noise floor with a noticeable velocity. The upper bounds for the $+X$ and $-X$ directions are defined by the PWM values that generate the magenta and blue curves, respectively. These are the first PWM inputs that cause the limb to bend 100° detected by the bend sensor.

4.2 Software Pipeline

The software pipeline proposed in our framework consists of four stages: calibration, exploration, network training, and fine-tuning. While specific details—such as the definition and format of network inputs and outputs—are tailored to our hardware platform, the overall pipeline is easily adaptable to other custom hardware platforms, as none of the stages depends exclusively on our system.

4.2.1 Autonomous Calibration of Input Bounds

To prevent hardware failure of the physical platform, only a specific range of input values is considered acceptable. For example, in the case of pneumatic actuators, such limits could correspond to the maximum pressure the pneumatic chambers can safely withstand. In our SMA-actuated limb, prolonged exposure to high current can generate excessive heat, potentially damaging the SMA coils. This overheating may cause the material to lose its shape memory, rendering it unable to contract as required in future operations.

Since SMA contraction is proportional to the generated heat, we define any bending angle greater than 100° along any axis as unsafe and to be avoided. Additionally, we observed that there exists a minimum non-zero input required for the SMA to produce any meaningful actuation.

Based on these constraints, we developed a method to automatically determine the input bounds by sweeping through possible PWM actuation values, holding each value for a fixed duration. The first PWM value that causes the limb to reach the 100° threshold is designated as the upper input bound. The first PWM value that causes the limb's velocity—calculated as the change in bending angle over the sampling period—to rise above the sensor's noise floor is considered the lower input bound. An example of this process is illustrated in Fig. 4.1. The upper bound is then redefined as the new 100% duty cycle, and the lower bound is scaled accordingly, with the exception of 0%, which is fixed at a PWM input of zero. This rescaled duty cycle is used in defining the action space described in Eq. 4.4.

For example, if the upper and lower PWM bounds for the $+X$ direction are 35 and 20 respectively, and for the $-X$ direction are 36 and 19, then $u^x = -10$ corresponds to a PWM of 36 and $u^x = -1$ corresponds to 19 for the SMA coil inducing negative X motion. Similarly, $u^x = 10$ corresponds to a PWM of 35 and $u^x = 1$ to a PWM of 20 for the SMA coil inducing positive X motion.

During this calibration phase, hysteresis effects should be disregarded, as the goal is to determine absolute hardware limits independent of previous states. Therefore, after holding each PWM value for a fixed duration, we include a long cooldown period to minimize the impact of thermal hysteresis. In our setup, we used a holding time of approximately 2.5 seconds and a cooling time of 2 minutes, with a sampling period of 50 milliseconds.

This calibration step is not essential for the software pipeline itself, but rather serves as a precautionary measure to identify hardware limits and prevent potential damage to the system.

4.2.2 Count Based Exploration Guided Data Collection

For the neural network representation to perform well, the training data must comprehensively cover the entire available state space. Previous works have attempted

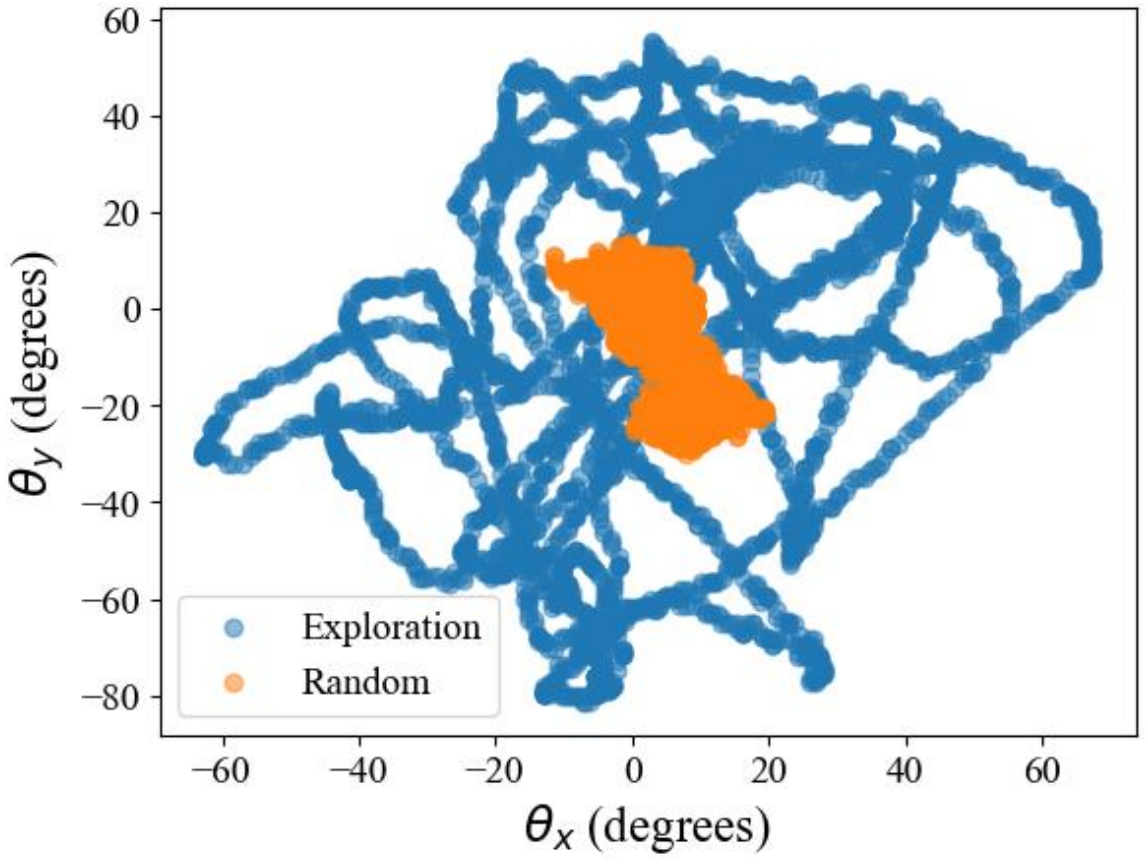


Figure 4.2: The effect of our exploration algorithm. The dots represent the states (in XY angular space) visited during the first 10 minutes of data collection using our exploration scheme (blue) and naive random motor babbling (orange). Our exploration method consistently prompts the limb to visit previously unexplored regions, resulting in more efficient and diverse data collection.

this using naive pseudo-random motor babbling [10] or by actuating the robot with a PI controller [44]. However, tuning the required constants for the PI controller is labor-intensive, especially for our 2-DOF soft limb. Additionally, we found that naive random motor babbling is inefficient for thoroughly exploring and collecting data across the full state space in our setup.

Taking inspiration from [56], we adopted a count-based exploration scheme. The complete algorithm is shown in Alg.1. We discretize the available state space into grids, count the number of times each grid cell has been visited, assign a weight to each grid, and then sample across all grids to select a target state to reach. To move

the limb toward the selected target, we activate the appropriate SMA coils using a randomly sampled u, as defined in Eq.4.4. The weights are computed such that less-visited grid cells are more likely to be selected, thereby encouraging exploration of under-sampled regions of the state space.

This scheme combines the exploratory nature of random actions with a strategic bias toward unexplored areas, improving the efficiency and coverage of data collection. A comparison of the states visited using naive random motor babbling versus our count-based exploration scheme is shown in Fig. 4.2.

4.2.3 Data Preparation

The data we gather from the exploration stage includes only the samples of limb bending angles in the X and Y axes (θ_x, θ_y) and the timestamp of the samples, recorded every 55 milliseconds. We compute the bending angle velocities ($\dot{\theta}_x$, $\dot{\theta}_y$) by finding the change in angle divided by the change in timestamp between two samples, that is

$$\begin{aligned} \theta^t &= [\theta_x^t, \theta_y^t] \\ \dot{\theta}^t &= \frac{\theta^t - \theta^{t-1}}{T^t - T^{t-1}} \end{aligned} \tag{4.5}$$

where T denotes the timestamp and t denotes the time step.

The nature of our problem, predicting the future change in dynamics as described in Eq. 4.3, naturally leads to a time-series formulation of the data. Furthermore, we treat the problem as a supervised learning task, where one data point at time step t includes the bending angles and bending angle velocities from the past n time steps, along with the action executed at t. In our case, n is chosen to be 100. We observed that increasing n improves performance only marginally while significantly increasing training time. We found a length of 100 to be a good balance between performance and efficiency.

The output label for the point at t includes the change in state (Δx_t) for the next time step, which includes the change in bending angles in both the X and Y axes, as

Algorithm 1 Count-Based Exploration

Initialize grid g over state space with counts zero
$m \leftarrow 0$ ▷ Maximum visit count
while True **do**
    Obtain current state x
    Determine grid cell c corresponding to x
    $g[c] \leftarrow g[c] + 1$
    if $g[c] > m$ **then**
        $m \leftarrow g[c]$
    end if
    Sample target cell $tc \leftarrow$ SampleCell
    Compute action $u \leftarrow$ ComputeAction(x, tc)
    Execute action u
end while
function SampleCell
    $S \leftarrow \sum_{c \in g}(m + 1 - g[c])$
    Sample r uniformly from $[0, S]$
    for each cell c in g **do**
        $r \leftarrow r - (m + 1 - g[c])$
        if $r \leq 0$ **then**
            return c
        end if
    end for
end function
function ComputeAction(x, tc)
    for each dimension i **do**
        if $x_i < \theta_i(tc)$ **then** ▷ $\theta_i(\cdot)$ denotes the i'th dimension of the state that denotes the center of tc
            $u_i \leftarrow -$ random number in $(0, 10)$
        else
            $u_i \leftarrow +$ random number in $(0, 10)$
        end if
    end for
    return u
end function

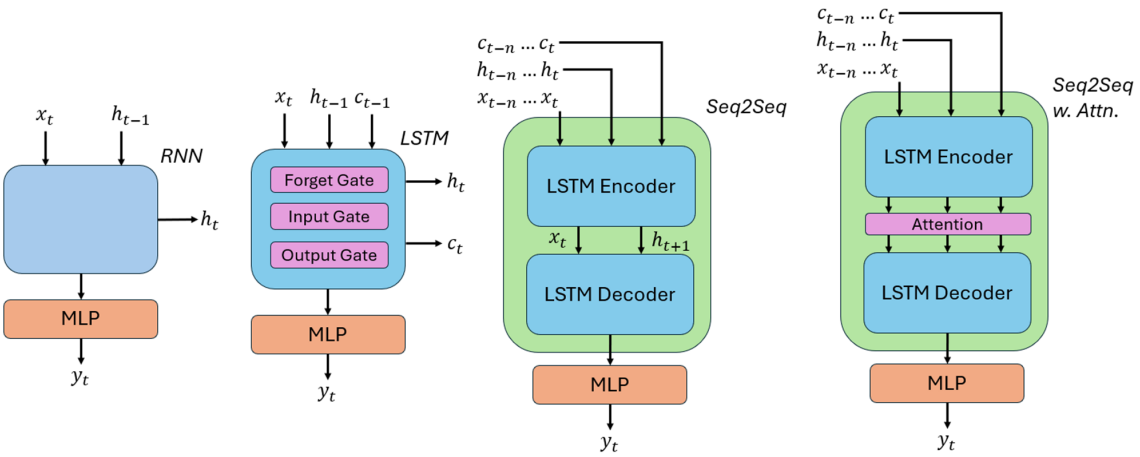


Figure 4.3: The four different architectures used for comparison to determine which performs best in predicting the forward kinematics of the soft limb.

well as the change in velocity.

$$\begin{aligned} \Delta\theta^t &= \theta^{t+1} - \theta^t \\ \Delta\dot{\theta}^t &= \dot{\theta}^{t+1} - \dot{\theta}^t \\ \Delta x_t &= [\Delta\theta^t, \Delta\dot{\theta}^t] \end{aligned} \tag{4.6}$$

We also observed a considerable amount of sensor noise in the state measurements, and we applied an exponential moving average filter in the following form:

$$x_{t+1} = (1 - \alpha)x_{t-1} + \alpha x_t \tag{4.7}$$

across our measurements of state before any further calculations. We used $\alpha = 0.8$.

4.2.4 Training of Recurrent Networks

The time-series nature of the problem lends itself naturally to recurrent networks for solving the supervised learning task of future state prediction. We tested our framework on multiple variants of recurrent networks, both to explore which architecture performs best in our case and to demonstrate that our framework is network-agnostic and adaptable to different network structures. Fig. 4.3 shows the different network architectures that were evaluated.

We tested the vanilla Recurrent Neural Network (RNN) [43], Long Short-Term Memory (LSTM) [28], Sequence-to-Sequence (Seq2Seq) [53], and Sequence-to-Sequence with an attention mechanism [64]. All networks are trained to minimize the Mean Squared Error (MSE) loss between the predicted and ground truth labels. That is,

$$\mathcal{L} = \frac{1}{M} \sum_{m=0}^{M} (\Delta \hat{x} - \Delta x)^2 \tag{4.8}$$

where M denotes the batch size and $\Delta \hat{x}$ denotes the predicted change in state from the neural network.

The vanilla RNN includes a recurrent unit that functions as a memory cell, producing a hidden state that is passed to subsequent time steps, thereby creating a recurrent structure.

However, vanilla RNNs suffer from issues such as vanishing gradients, where the gradients approach zero and gradient descent fails to effectively update the network weights. Intuitively, this is similar to memory saturation, where too much historical information overwhelms the network, and important features are lost. To address this, LSTMs introduce a forget mechanism that allows the network to discard trivial historical information, thereby enabling better handling of long-term dependencies.

The Seq2Seq architecture incorporates an encoder-decoder structure, where one LSTM encodes the input sequence into a latent representation, and another LSTM decodes this latent space into the output sequence. This structure increases expressiveness and enables the network to capture patterns in longer sequences more effectively than standard LSTMs.

The attention mechanism [64], which has been highly successful in natural language processing tasks such as language translation, inspired further experimentation. In some ways, our problem is analogous to translation: we aim to translate a sequence of state and action inputs into another sequence representing predicted changes in state. Therefore, we explored augmenting the decoder of the Seq2Seq architecture with an attention mechanism.

Unfortunately, attention mechanisms alone, such as those used in standard Transformer architectures, struggle to capture long-term dependencies without specific architectural modifications. Techniques such as Transformer-XL [16] or Temporal

Fusion Transformers [35] attempt to address this, but are beyond the scope of this work. Attempts had been made by us to incorporate these advanced attention-only architectures, but unfortunately, to no avail. As such, we focus exclusively on recurrent network architectures in this study and leave the exploration of attention-only models to future work.

4.2.5 Adaptation to Limbs of Different Stiffness

One of the phenomena we hypothesize is that the dynamics approximation trained using our proposed framework can be easily adapted to similar systems through fine-tuning, requiring significantly less data than was used for the original training. To test this hypothesis, we first trained the dynamics approximation on soft limbs built with stiffer materials, having a Shore hardness of 40A. We then ran the exploration algorithm to collect data on softer limbs, made from materials with a Shore hardness of 30A. Starting from the network approximation trained on the stiffer limb, we fine-tuned the network using the smaller dataset collected from the softer limb.

4.3 Results

4.3.1 Metrics

To compare the performance across different network architectures and settings, we chose to use the Root Mean Square Error (RMSE) as the evaluation metric. However, because we aim to validate the predictions of states—which include both bending angles (in degrees) and angular velocities (in degrees per second)—we compute RMSE separately for angle predictions and velocity predictions to maintain unit consistency. N denotes the total number of time steps in a given trajectory of angles and velocities used for performance evaluation.

$$RMSE_{\theta} = \sqrt{\frac{\sum_{t=0}^{N}(\hat{\theta}^t - \theta^t)^2}{N}} \tag{4.9}$$

$$RMSE_{\dot{\theta}} = \sqrt{\frac{\sum_{t=0}^{N}(\hat{\dot{\theta}}^t - \dot{\theta}^t)^2}{N}} \tag{4.10}$$

In our results, we evaluate the one-step-ahead prediction capabilities of our dynamics approximation, which uses the previous n ground truth states as inputs for prediction:

$$\hat{x}_{t+1} = x_t + \phi(x_t ... \; x_{t-n}, u_t) \tag{4.11}$$

Although the one-step-ahead prediction error corresponds to the loss minimized during training, it is not particularly useful for practical model evaluation and is a poor indicator for accuracy, especially for long-term trends. Therefore, we also assess long-term prediction performance by rolling out the model recursively, feeding the predicted states back into the model as inputs for future predictions:

$$\hat{x}_{t+1} = \hat{x}_t + \phi(\hat{x}_t ... \; \hat{x}_{t-n}, u_t) \tag{4.12}$$

The results of both the one-step prediction and the rollout prediction are evaluated using the RMSE metrics defined in Eq. 4.9 and Eq. 4.10.

4.3.2 Model Comparisons

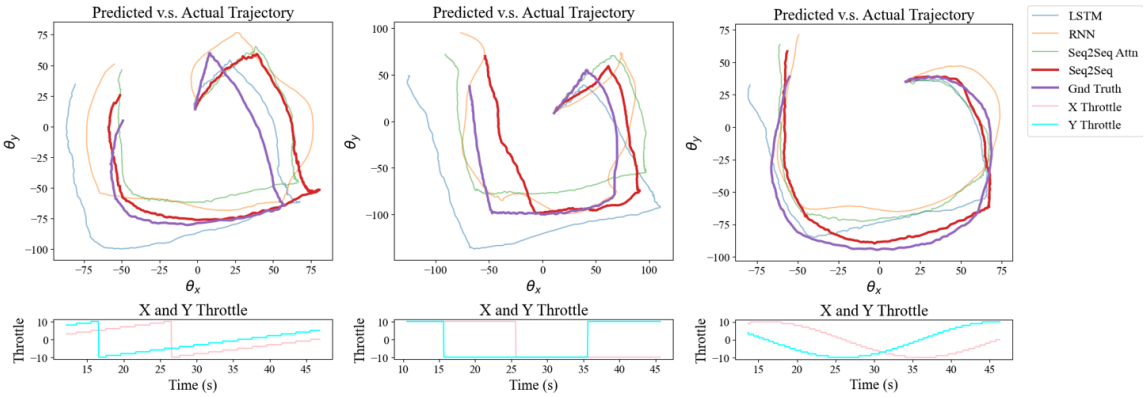
To cross-compare the four different neural network architectures, we collected approximately 130 minutes of data (∼110,000 data points) using our exploration method. Out of the 130 minutes of collected data, around 30 minutes (∼25,000 data points) are used as the testing set, and the remaining 100 minutes (∼85,000 data points) are used for training. Each data point is collected approximately every 50 milliseconds.

The data used in this section comes from the stiffer of the two soft limbs (Shore Hardness 40A).The upper and lower PWM bounds found for actuation are 15% and 7%, respectively, in all directions.

All neural networks are configured with a hidden size of 512 and a total of 3 recurrent layers. The networks are trained with a batch size of 1024, using a context window of 100 ($n = 100$ in Eq. 4.3) for a total of 400 epochs.

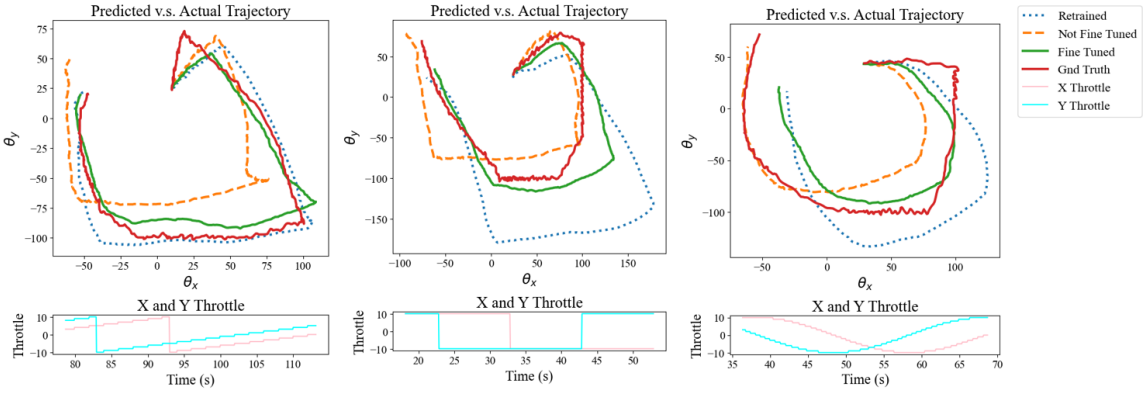
4.3.3 Trajectory Roll Out

A visualization of the trajectory rollouts using three different input throttle signals—sine, square, and sawtooth—over a duration of approximately 40 seconds is



a) Saw tooth throttle input b) Square wave throttle input c) Sine wave throttle input

Figure 4.4: Predicted bending angle trajectory of different neural network architectures and the actual bending angle under different input signals. The ground truth (purple) and the best model (red) is bolded. The shared legend for all sub figures are in c).



a) Saw tooth throttle input b) Square wave throttle input c) Sine wave throttle input

Figure 4.5: Predicted bending angle trajectory of fine tuned, not fine tuned and retrained network. The ground truth (red) and the fine tuned model (green) are solid. The shared legend for all sub figures are in c).

shown in Fig. 4.4. In all three cases, the Seq2Seq architecture without the attention mechanism performed best in qualitatively estimating the forward dynamics. The rollouts are generated using the method described in Eq. 4.12.

One notable observation is that the networks perform better when the input signals are smooth (e.g., sine wave), and struggle more with signals that contain abrupt changes (e.g., square and sawtooth waves).

4.3.4 Metric Comparisons

Table 4.1 shows the RMSE of the bending angles θ and bending angle velocities $\dot{\theta}$ for three different scenarios. The rollouts are performed on the testing dataset, which includes random throttle inputs. RMSE is calculated for rollouts of 1 minute (∼800 data points) and 30 minutes (∼25,000 data points), as well as for one-step-ahead prediction, as described in Eq. 4.11.

For the 1-minute rollout, all models appear to perform equally well. However, for the 30-minute rollout, the Seq2Seq and Seq2Seq with attention architectures demonstrate relatively better performance in tracking bending angles compared to RNN and LSTM.

In the case of one-step-ahead prediction, both Seq2Seq architectures and the LSTM architecture outperform the vanilla RNN. Interestingly, for all models, while the RMSE for bending angle prediction is significantly lower than during rollouts, the velocity estimation tends to be worse.

Model	**RMSE** $\dot{\theta}$ (60s) [*deg./sec.*]	**RMSE** θ (60s) [*deg.*]	**RMSE** $\dot{\theta}$ (30 min) [*deg./sec.*]	**RMSE** θ (30 min) [*deg.*]	**RMSE** $\dot{\theta}$ (1-step) [*deg./sec.*]	**RMSE** θ (1-step) [*deg.*]
SEQ2SEQ w. Attention	9.52	12.28	9.83	9.82	12.45	0.89
SEQ2SEQ	9.78	11.03	9.51	9.84	11.41	0.86
RNN	9.16	12.58	8.48	12.79	28.13	2.73
LSTM	9.94	13.82	9.66	13.43	11.85	0.75

Table 4.1: RMSE values of the prediction of various different neural network architectures

Model	**RMSE** $\dot{\theta}$ (60s) [*deg./sec.*]	**RMSE** θ (60s) [*deg.*]	**RMSE** $\dot{\theta}$ (10 min) [*deg./sec.*]	**RMSE** θ (10 min) [*deg.*]	**RMSE** $\dot{\theta}$ (1-step) [*deg./sec.*]	**RMSE** θ (1-step) [*deg.*]
Finetuned	7.91	23.82	7.86	23.00	8.95	0.65
No Finetuning	9.02	28.58	9.37	29.19	11.88	0.87
Retrained	7.26	29.60	7.77	28.10	9.44	0.73

Table 4.2: RMSE of finetuned or non-finetuned neural network

4.3.5 Fine Tuning Results

To test our hypothesis that pre-trained models can be fine-tuned using data from similar systems, we conducted experiments on a softer soft robotic limb (Shore hardness 30A, Limb 1 in Fig. 1.1). We collected approximately 40 minutes of data

(∼33,000 data points) from the softer limb using our exploration algorithm, using 30 minutes (∼25,000 data points) as the training set for fine-tuning, and the remaining 10 minutes (∼8,000 data points) as the testing set. This dataset is only about one-third the size of the training set used for the stiffer limb models.

Limb 1 (blue in Fig. 1.1), due to its softer nature compared to Limb 2 (purple in Fig. 1.1), has a smaller PWM range—approximately 7% for the lower bound and 11% for the upper bound in all directions. Because Seq2Seq with attention has the most complex architecture among the tested models and is thus more difficult to train, we selected it to study the effect of fine-tuning.

We trained one network from scratch using only the data collected from the softer limb. The fine-tuned network, on the other hand, was initialized using the weights of the Seq2Seq with attention model trained on the stiffer limb data and then further trained on the softer limb data. All networks were trained using the same parameters as previously described.

Fig. 4.5 shows the qualitative results of fine-tuning. Table 4.2 presents the RMSE values for three scenarios: a 1-minute rollout, a 10-minute rollout, and one-step-ahead prediction. These rollouts use the 10-minute testing set, which is shorter than the dataset used in Table 4.1. The results show that the fine-tuned model performs better overall in all scenarios compared to both the model trained from scratch and the model without fine-tuning.

4.4 Discussion

The data pipeline developed in this work demonstrates the effectiveness of neural networks in approximating complex SMA-actuated soft robotic kinematics. The neural network closely approaches the performance limits of the capacitive bend sensor, which has a maximum error of 0.5 degrees [57]. With a sampling rate of 13 Hz, the expected velocity noise is approximately 10 degrees per second. Our model achieves a one-step prediction RMSE of 8 deg/s for velocity and about 0.8 degrees for angular position—values that align well with the sensor's specified error bounds. The long-duration rollout showed that the model is capable of capturing complex dynamics, as the RMSE did not compound significantly over time showing promising results in accurately predicting the state of the soft robotic system.

Chapter 5

Learning Position Control

One primary motivation for learning the forward kinematics is to create a computationally efficient simulator for the soft robotic system. Leveraging this learned kinematics model, we apply reinforcement learning to train a controller that enables the robot to perform goal-reaching tasks.

Rather than treating the learned kinematics solely as a simulator and training a policy network from scratch using standard RL algorithms, we propose using part or all of the kinematics network as a feature extractor. This approach allows the policy network to converge faster and learn more efficiently, framing policy learning as an extended fine-tuning of the pre-trained kinematics model.

We built a custom RL environment based on the learned kinematics and tested various policy architectures, both with and without feature extractors. These networks were deployed on the custom SMA-based soft robotic platform. We also evaluated the robustness of the learned policy under dynamic changes, including external perturbations, added loads, and variations in limb stiffness.

5.1 Methodology

5.1.1 Environment Setup

Constructing the environment is the first step in training a reinforcement learning policy. The state and action definitions in this section follow from Sec. 6.1. Fig. 5.1

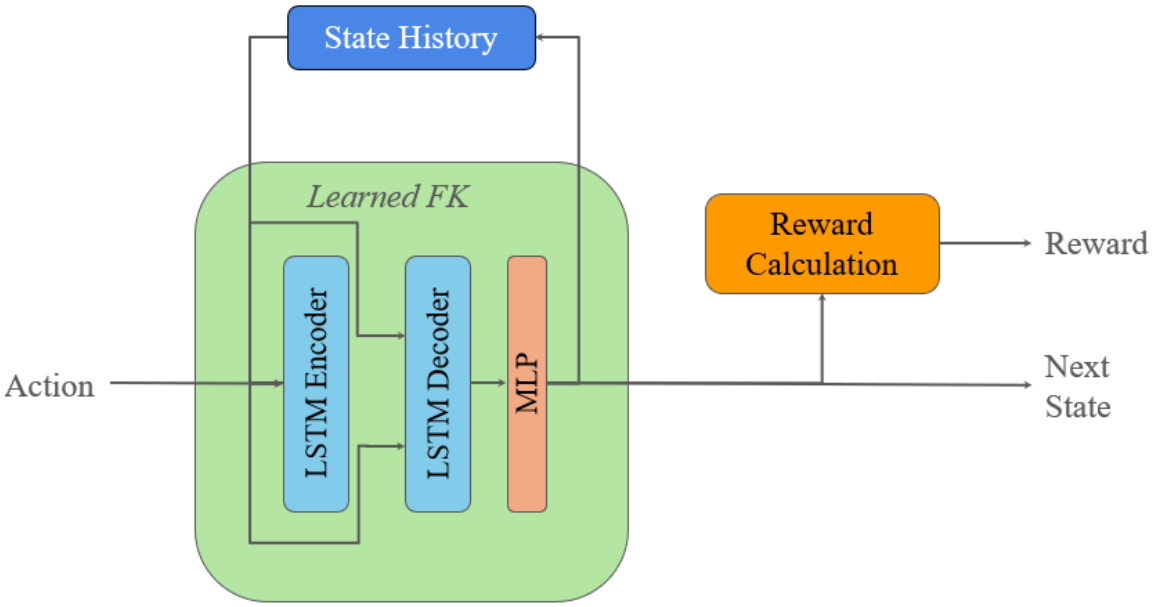


Figure 5.1: Block diagram of our learned kinematic-based training environment. Because the learned kinematics network requires a history of states and actions, a cache of the past 100 states is stored. However, only the next predicted state is used for reward calculation and as the output of the environment (the observation).

shows a block diagram of our training environment.

The backbone of the environment is the full learned forward kinematics network described in Sec. 4. The environment receives an action and outputs an observation, while also computing a reward based on the current state. The observation includes:

1. The current state x_t of the simulated soft robotic limb
2. The previous action u_{t-1}
3. The current goal state g, where $g = [\theta_x^{goal}, \theta_y^{goal}, \dot{\theta}_x^{goal}, \dot{\theta}_y^{goal}]^T$ represents the target bending angles and angular velocities
4. The current accumulated "power" on each SMA, $p = [{}^{+}p_t^x, {}^{-}p_t^x, {}^{+}p_t^y, {}^{-}p_t^y]$

Power

One addition to the original state definition of the soft robotic limb is the notion of "power." The "power" here is not a real physical value, only a proxy to approximate the temperature of each SMA without physically attaching thermocouples, which would

Algorithm 2 Cooling and Heating Update

1: ${}^{+}p^{x} \leftarrow^{+} p^{x} \cdot (1 - k\,\Delta t)$
2: ${}^{+}p^{y} \leftarrow^{+} p^{y} \cdot (1 - k\,\Delta t)$
3: ${}^{-}p^{x} \leftarrow^{-} p^{x} \cdot (1 - k\,\Delta t)$
4: ${}^{-}p^{y} \leftarrow^{-} p^{y} \cdot (1 - k\,\Delta t)$
5: **if** $u^{x} > 0$ **then**
6: $\quad {}^{+}p^{x} \leftarrow^{+} p^{x} + \dfrac{u^{x2}}{3}\,\Delta t$
7: **else**
8: $\quad {}^{-}p^{x} \leftarrow^{-} p^{x} + \dfrac{u^{x2}}{3}\,\Delta t$
9: **end if**
10: **if** $u^{y} > 0$ **then**
11: $\quad {}^{+}p^{y} \leftarrow^{+} p^{y} + \dfrac{u^{y2}}{3}\,\Delta t$
12: **else**
13: $\quad {}^{-}p^{y} \leftarrow^{-} p^{y} + \dfrac{u^{y2}}{3}\,\Delta t$
14: **end if**

complicate the wiring. As stated previously, SMA-based systems suffer significantly from hysteresis due to slow convection cooling, a result of the thermal properties of the materials used. By incorporating this power proxy, we aim to give the neural network policy an implicit understanding of residual heat in the SMAs, thereby improving performance.

Power is calculated at each time step as the net result of heat input to the SMA, modeled using Ohm's law:

$$P = \frac{V^2}{R} \quad \text{(Ohm's law)} \tag{5.1}$$

minus the heat loss due to cooling, modeled using Newton's law of cooling:

$$\frac{dT}{dt} = -k\,(T - T_{\text{env}}) \tag{5.2}$$

Discretizing the equation above:

$$T_{t+1} = T_t - k\,\Delta t\,(T_t - T_{\text{env}}) \tag{5.3}$$

We treat T_t as the "power" proxy at time step t, and set $T_{\text{env}} = 0$, assuming all input power eventually dissipates. Therefore, without loss of generality, consider the SMA actuator in the $+x$ direction. The power at each time step is updated as follows:

$$^{+}p^{x}_{t+1} = \frac{u^{x2}_{t}}{R}\Delta t - {}^{+}p^{x}_{t}(1 - k\Delta t) \tag{5.4}$$

where k is the cooling constant, Δt is the time interval between time steps, and R is the resistance of the SMA actuator.

During training, we set $k = 0.2$, $\Delta t = 0.075$, and $R = 3$. Fig. A.4 shows how different values of k affect the cooling behavior. We observed that SMAs embedded in the soft limb typically return close to ambient temperature after approximately 30 seconds. Since antagonistic SMA pairs are never actuated simultaneously, we update the power values following the procedure outlined in Alg. 2.

Reward Formulation

For the reward function, we drew heavy inspiration from OpenAI Gymnasium's Reacher environment [7]. We define our reward function as follows:

$$r_{\text{reach}}(x, u) = -||\theta - \theta_{\text{goal}}|| + ||u||^2 \tag{5.5}$$

$$r_{\text{vel}}(x) = \begin{cases} -\sqrt{||\dot{\theta}|| \cdot ||\dot{\theta}_{\text{goal}}|| - \dot{\theta}^T\dot{\theta}_{\text{goal}}}, & \text{goal reached} \\ 0, & \text{otherwise} \end{cases} \tag{5.6}$$

$$r_{\text{term}} = \begin{cases} 1000, & \text{goal reached} \\ -5000, & \text{out of bounds} \\ 0, & \text{otherwise} \end{cases} \tag{5.7}$$

$$r(x, u) = r_{\text{reach}} + r_{\text{vel}} + r_{\text{term}} \tag{5.8}$$

Here, x is the state, consisting of $[\theta, \dot{\theta}]$ as described in Sec. 6.1.

The meaning of each reward term is as follows:

1. r_{reach} encourages the agent to learn a policy that moves toward the goal position

as quickly as possible.

2. r_{vel} provides an incentive for the agent to match the goal velocity once the robot has reached the goal position.
3. r_{term} is the terminal reward: a positive reward is given when the goal is reached to encourage task completion, while a large negative penalty is applied when the agent moves the robot out of bounds. We found that without this penalty, the policy often exploits the reward function by ending the episode early through leaving the valid state space.

Terminating Conditions & Resets

There are a total of three conditions under which we consider a training episode to have ended:

1. **Goal Reached:** The bending angles θ of the current simulated robot state are within a tolerance of the goal bending angles, i.e., $||\theta - \theta_{\text{goal}}|| < \delta$, where δ is the goal tolerance.
2. **Out of Bounds:** The robot's state exceeds the valid range, defined as $\theta_x, \theta_y, \dot{\theta}_x, \dot{\theta}_y \in [-80, 80]$.
3. **Timeout:** The robot neither reaches the goal nor goes out of bounds within 250 timesteps.

Dynamic Goal Tolerance: To aid the training process, the tolerance for determining whether the goal is reached follows a linearly decreasing schedule based on the total number of simulated timesteps. Initially, the tolerance is large, making it easier to satisfy the goal condition—as long as the robot enters the general vicinity of the goal, it is considered successful. As training progresses, the tolerance shrinks, making the goal condition stricter. Eventually, the robot must be within 1° of the goal position to be considered successful. After an episode terminates, the environment is reset to prepare for the next episode. We define two types of resets: a *full reset* and a *half reset*.

Half Reset: Since the underlying model that predicts the robot's kinematics relies on the learned kinematic network, whose input includes both the last action and a history of previous states and actions, a half reset reassigns a new goal position

while leaving the current state and history unchanged. This allows the model to potentially capture long-term dependencies that extend beyond a single episode.

Full Reset: A full reset reassigns not only a new goal position but also a new starting state for the robot. It also clears the history of past states and actions, simulating a fresh system startup after a long cooldown period. Full resets occur randomly about 1% of the time when an episode terminates.

5.1.2 Training Setup and Architecture Used

We tested four commonly used reinforcement learning algorithms: Soft Actor-Critic (SAC) [27], Proximal Policy Optimization (PPO) [47], Deep Deterministic Policy Gradient (DDPG) [34], and Twin Delayed DDPG (TD3) [23]. Among these, only SAC and TD3 demonstrated substantial learning performance, with SAC performing marginally better overall. Therefore, all studies presented in this section use SAC as the reinforcement learning algorithm for training the task-achieving policy. Our SAC implementation is adapted from the CleanRL library [30]. For the complete set of training hyperparameters, please refer to Table A.1.

Input Parsing: Because the learned kinematics network takes a time series of past states and actions as input, we store a history of observations from the environment and use a sequence length of 100—consistent with the window size used when training the kinematics network. Since the kinematics network requires only the previous history of states and actions, we parse and pass these components to the head. The remaining components—SMA power information, goal position, and the state information from the most recent timestep—along with the layer-normalized output of the head, are passed directly to the actor and critic networks individually.

Architectures: In this study, we investigate whether using the learned kinematics network as a feature extractor or shared "head" for both the actor and critic networks can improve training speed, sample efficiency, and overall performance. To this end, we trained policies using four different neural network architectures: *MLP No Head*, *MLP Head*, *Learned Kinematics Head (Encoder)*, and *Learned Kinematics Head (Full)*. Fig. 5.2 illustrates the different architectures in graphical form.

Learned Kinematics Head (Full): This architecture uses the entire learned kinematics network, including both the encoder LSTM and decoder LSTM, as the feature

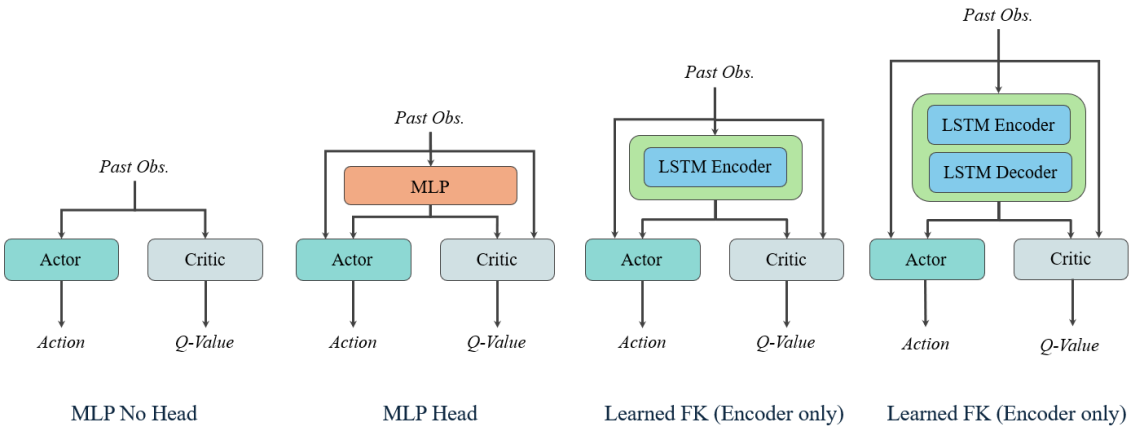


Figure 5.2: The architectures used for comparison in training a task-achieving policy. Since the learned forward kinematics models require a history of states, the observations are cached with a window length of 100 and passed to the networks, following the details in Sec. 5.1.2.

extractor. The output of the head is the predicted delta in state, which is then passed to both the actor and critic networks.

Learned Kinematics Head (Encoder): This architecture uses only the encoder LSTM portion of the learned kinematics network. The output is a latent vector embedding that potentially encodes useful information about the system's kinematics, which the actor and critic networks can leverage.

MLP Head: Instead of a recurrent network, this architecture uses a multilayer perceptron (MLP) as a feature extractor. The input history of states is flattened and passed through the MLP, and the resulting latent embedding is forwarded to the actor and critic networks.

MLP No Head: The simplest architecture omits the use of a feature extractor entirely. In this setup, both the actor and critic networks are MLPs that directly take the flattened history of observations as input.

5.2 Hardware Experiments

To compare the performance of the trained policies, we deployed them on our real-world soft robotics platform. The policies were tasked with tracking three patterns—a spiral, a square, and the letters "SML" (abbreviation for Soft Machines Lab)—as

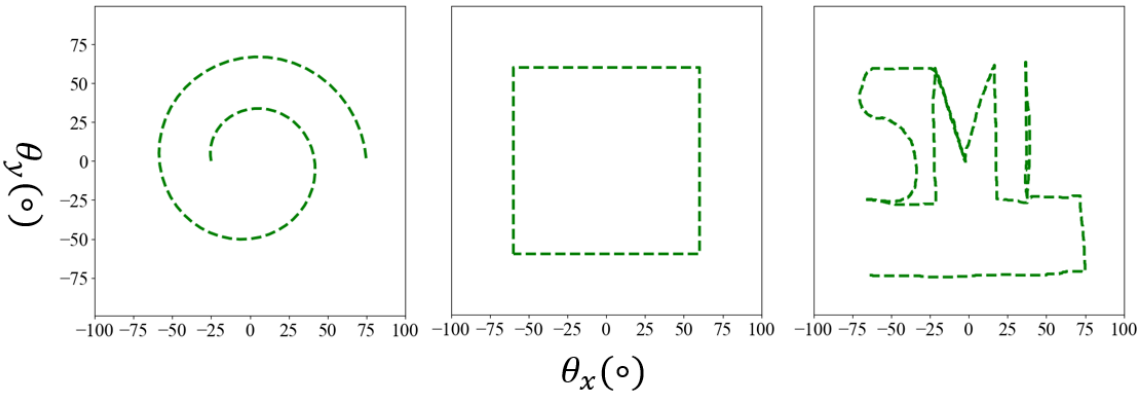


Figure 5.3: The three patterns used for trajectory tracking in hardware experiments: Spiral (left), Square (middle), and SML (right).

shown in Fig. 5.3, in a closed-loop manner within the bending angle space.

We built the software pipeline using ROS2 Humble [36], which relays real-time robot state information to the learned policy and sends the resulting actions back to the robot. Fig. 5.4 illustrates the hardware control loop used in these experiments.

To track the patterns, we implemented a simple controller: if the robot reaches within a specified tolerance of a point along the trajectory, the controller updates the goal to the next point further along the path. In all experiments, the tolerance was set to 10°.

5.2.1 Architecture Comparisons

We deployed the learned policies from the *Learned Kinematics Head (Full)*, *Learned Kinematics Head (Encoder)*, and *MLP No Head* architectures, along with a classic non-learning-based Proportional-Integral-Derivative (PID) controller, on the real-world soft robotic platform.

We chose not to include the *MLP Head* architecture, as it failed to demonstrate meaningful learning during training. The PID controller serves as a baseline to evaluate whether our proposed learning-based methods perform on par with, below, or better than traditional, model-free, non-learning-based control approaches.

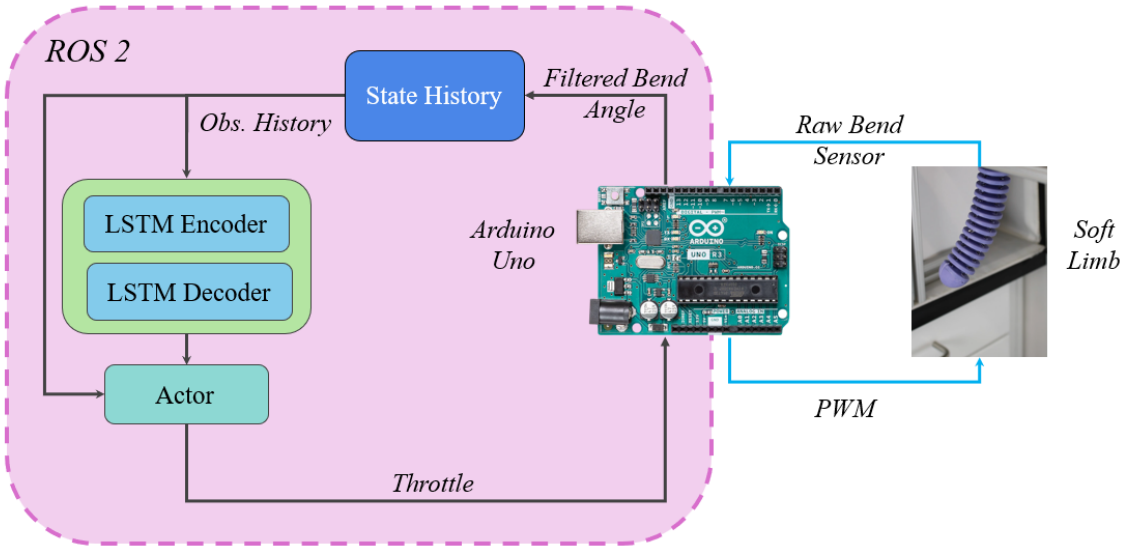
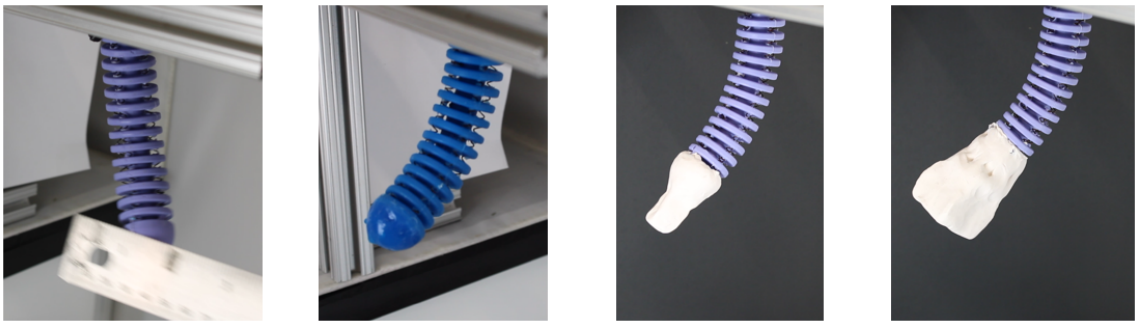


Figure 5.4: Block diagram of the control loop used for deploying policies on hardware. Data from the bend sensor installed inside the soft limb is sent to an Arduino Uno [3], which relays the data to a ROS node via serial communication. The ROS node broadcasts the state information, which is cached and preprocessed to match the observation format used during training. A history of these observations is then sent to another ROS node running the learned policy. The policy generates control actions, which are sent back to the Arduino via ROS, and the Arduino actuates the robot accordingly.



a) External Perturbation
b) Softer Limb (Blue)
c) External Load (10g)
d) External Load (50g)

Figure 5.5: The four robustness tests conducted to evaluate the learned policies' ability to adapt to previously unseen situations.

5.2.2 Robustness Testing

In addition to evaluating the policies' ability to track various patterns, we also tested their robustness by introducing external disturbances. We designed four scenarios to assess how well the policies adapt to previously unseen conditions. Fig. 5.5 illustrates these four robustness test scenarios.

Sudden External Perturbation: In this scenario, the robotic limb is manually disturbed (i.e., tapped or pushed) mid-trajectory while tracking a circular path, to evaluate the policy's ability to recover and return to the intended trajectory.

Softer Limb: Here, we replace the stiffer purple limb (Shore Hardness 40A)—on which the learned forward kinematics model was trained—with a softer blue limb (Shore Hardness 30A). The policy is then deployed in a zero-shot fashion to track a circle, testing its adaptability to altered limb dynamics.

External Load (10g and 50g): In these scenarios, we attach external clay loads of 10g and 50g to the tip of the limb. The policies are again deployed in a zero-shot fashion to track a circle, evaluating their ability to adapt to additional external forces and, effectively, a "stiffer" dynamic response.

5.3 Results

5.3.1 Training Results

Metrics

For training, we use two metrics to compare performance. The first is the *episodic return*, which is the sum of rewards accumulated at each step within a training episode. The second metric is the *success rate*. Every 1000 training steps, the policy is frozen, and 100 evaluation episodes are rolled out to assess how many episodes terminate by reaching the goal. The fraction of successful episodes (goal reached) out of 100 is defined as the *success rate*.

Architecture Comparisons (Training)

Fig. 5.6 shows the success rate of policies trained with different architectures. All learned kinematic feature extractors (LK Head (Encoder) and LK (Full)) are frozen during training. Both the power representation and the dynamic goal tolerance mechanism are enabled during training. Policies using learned kinematic heads outperform those using an MLP head (which failed to learn) or policies without any feature extractor.

Architecture	Final Success Rate	Final Episodic Return
LK Head (Encoder)	**1.00**	**-1507.87**
LK (Full)	0.90	-1794.63
MLP Head	0.00	-7046.79
MLP No Head	0.76	-2009.50

Table 5.1: Final training performance metrics for different architectures after 100k training steps in the learned kinematic simulator. Maximum success rate and episodic return are bolded.

Table 5.1 presents the final success rates and episodic returns for each architecture after 100k training steps. Because episodic returns fluctuate significantly during training (as shown in Fig. A.1), the reported values are smoothed to better reflect underlying trends.

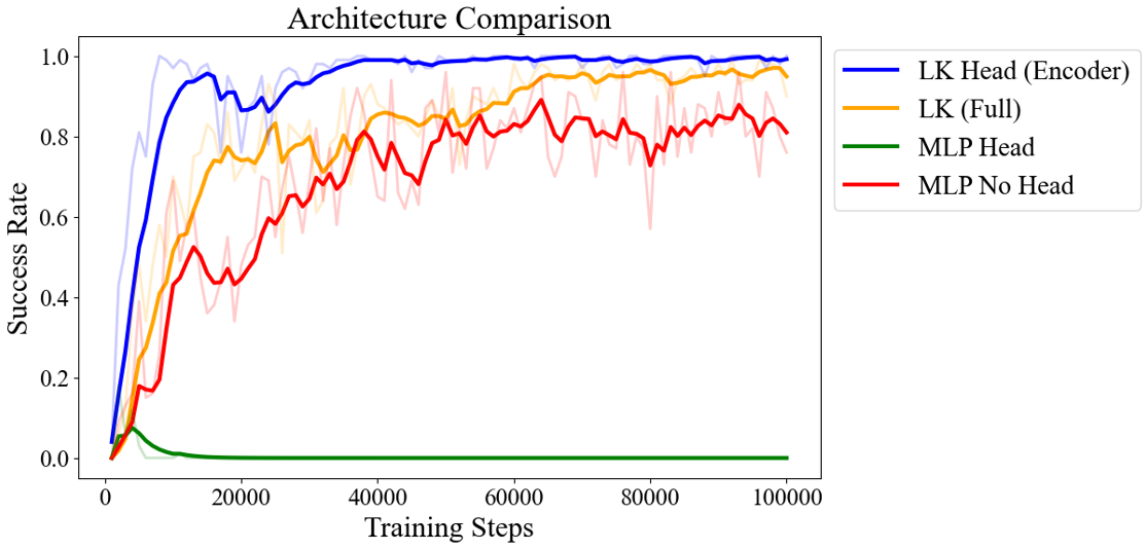


Figure 5.6: Success rate of different architectures over the training process. LK stands for "Learned Kinematics." Faint lines represent raw data, while solid lines show smoothed trends using an exponential moving average with $\alpha = 0.3$.

Among the four architectures, those utilizing learned kinematics—LK Head (Encoder) and LK (Full)—outperformed the MLP-based architectures in both success rate and episodic return. Notably, MLP Head appeared to converge to a poor local minimum and failed to learn a viable task-achieving policy.

Between the two learned kinematic architectures, LK Head (Encoder) achieved the best performance. This may be due to the actor and critic networks operating directly on the latent space representation, which could retain useful information otherwise abstracted or lost in the full kinematics model.

Ablation Study

Freeze vs. Unfreeze & Pretrained vs. Not Pretrained: In the previous section, we observed that the LK Head (Encoder) architecture performed the best—specifically when the encoder from the learned kinematics was used and its weights were frozen during training. However, it is possible that the strong performance was due to the use of a time-series architecture rather than pretraining itself. To isolate the effects of pretraining and freezing, we conducted an ablation study across four configurations, covering all combinations of these two variables: frozen pretrained head, frozen

non-pretrained head, non-frozen pretrained head, and non-frozen non-pretrained head.

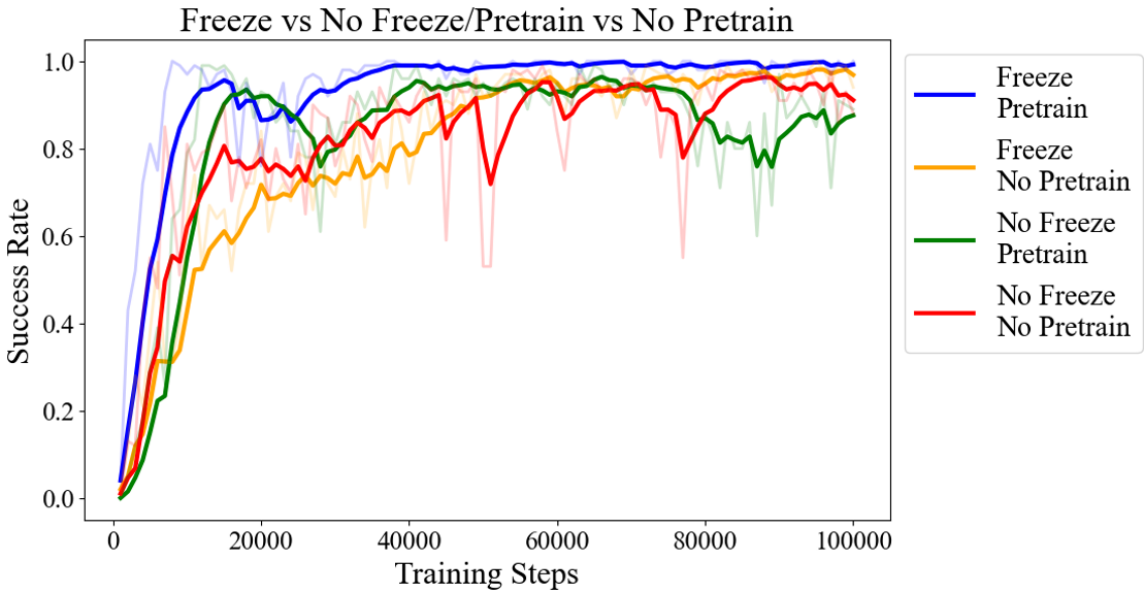


Figure 5.7: Ablation study on the impact of using a pretrained feature extractor and freezing its weights. The solid lines are smoothed using an exponential moving average filter with $\alpha = 0.3$; faint lines show raw values.

	Success Rate	Episodic Returns	Training Time (Hrs)
Freeze Pretrain	**1.00**	**-1632.21**	4.08
Freeze No Pretrain	0.94	-2116.66	4.06
No Freeze Pretrain	0.89	-1821.54	4.89
No Freeze No Pretrain	0.88	-2115.56	4.88

Table 5.2: Final success rate, episodic returns, and training time for different combinations of pretrained feature extractor usage and weight freezing. Best performance values are bolded.

Fig. 5.7 shows the success rates over training steps for all configurations, while Table 5.2 summarizes the final success rates, episodic returns, and training durations. The frozen-weight configurations appear to perform better in terms of success rate, regardless of whether the feature extractor was pretrained. However, the episodic returns show that models using pretrained weights achieve better performance overall.

Additionally, models with frozen weights trained significantly faster—up to 20% faster—than those with trainable heads. Taken together, these results suggest that the **frozen pretrained** model not only performs best but also offers improved training efficiency compared to non-frozen alternatives.

Effect of Power and Dynamic Goal Tolerance: Another set of ablations investigates the effect of including the power proxy (Sec. 5.1.1) and dynamic goal tolerance mechanism (Sec. 5.1.1) in the LK Head (Encoder) training setup. Fig. 5.8 shows the success rate over time for four configurations: with and without power proxy, and with and without dynamic goal tolerance. Table 5.3 presents the final success rates and episodic returns after 100k training steps.

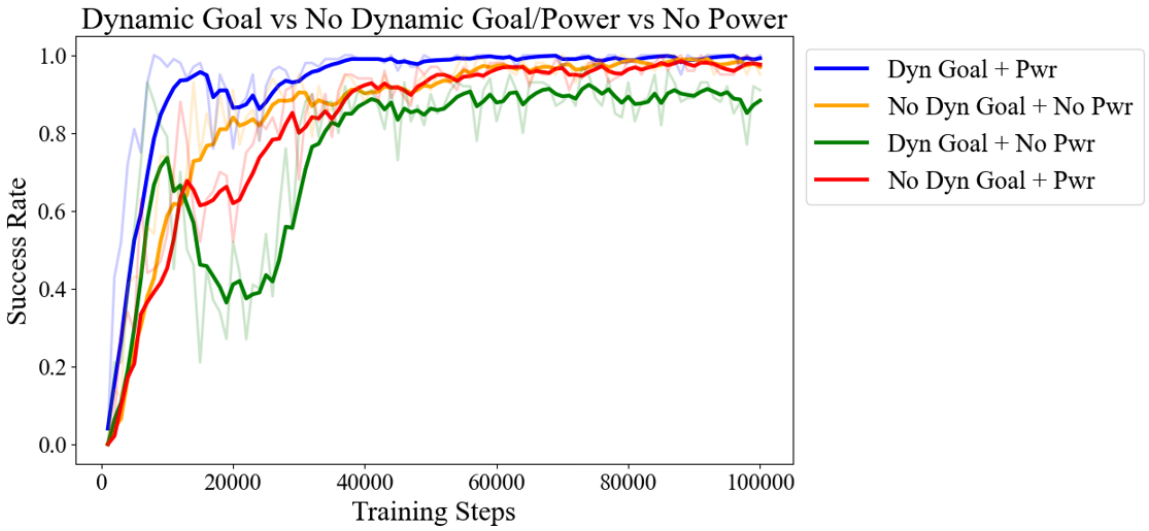


Figure 5.8: Ablation study on the impact of using power proxy and dynamic goal tolerance on training performance. Solid lines show smoothed data using an exponential moving average ($\alpha = 0.3$); faint lines show raw results.

The results show that incorporating the power proxy improves policy performance: both configurations using power achieved higher success rates than their counterparts without it. Moreover, the combination of both dynamic goal tolerance and power proxy achieved the best overall performance, indicating that these features are complementary and beneficial for improving training outcomes.

	Final Success Rate	Final Episodic Returns
Dyn Goal + Pwr	**1.00**	**-1351.15**
No Dyn Goal + No Pwr	0.94	-2312.02
Dyn Goal + No Pwr	0.91	-2027.02
No Dyn Goal + Pwr	0.97	-2248.68

Table 5.3: Final success rates and episodic returns after training for 100k steps, comparing configurations with and without dynamic goal tolerance (Dyn Goal) and power proxy (Pwr). Best performance values are bolded.

5.3.2 Hardware Results

Based on the training results and ablation studies, the configuration using frozen pretrained heads along with both dynamic goal tolerance and the power proxy demonstrated the best performance. Therefore, for all hardware evaluations, we test the following four models and methods: *LK Head (Encoder)* and *LK (Full)* as the top-performing training-based models, *MLP No Head* as the baseline for training-based methods, and *PID* as the baseline for non-training-based (classical) control.

Architecture Comparison (Hardware)

We deployed the four models and methods mentioned above to the task of tracking three different patterns: Spiral, Square, and SML. Fig. 5.9 shows the recorded trajectories from the *LK Head (Encoder)* model, while Fig. 5.10 presents results from the *LK (Full)* model. Fig. 5.11 displays the trajectories from the *MLP No Head* baseline, and Fig. 5.12 illustrates the trajectories from the *PID* controller.

Method	Spiral	Square	SML	Avg.
LK (Encoder)	**4.73**	3.22	**3.62**	**3.86**
LK (Full)	4.75	**3.00**	4.00	3.92
MLP No Head	4.73	4.75	4.10	4.53
PID	5.06	3.48	3.91	4.15

Table 5.4: RMSE of different architectures for tracking various patterns. Best RMSE values per pattern are bolded.

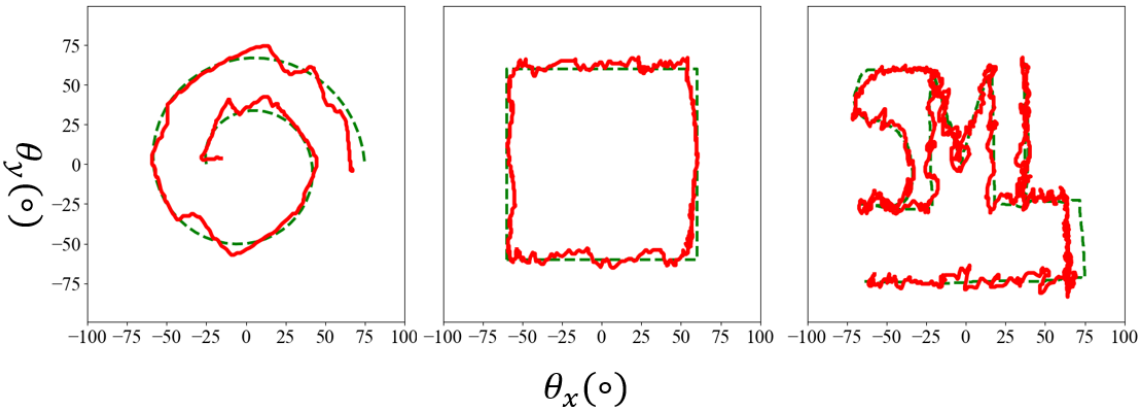


Figure 5.9: Recorded tracking trajectory of the policy trained with the encoder portion of the pretrained learned kinematics (LK Encoder) as the head. All models were trained with the power proxy and dynamic goal tolerance enabled.

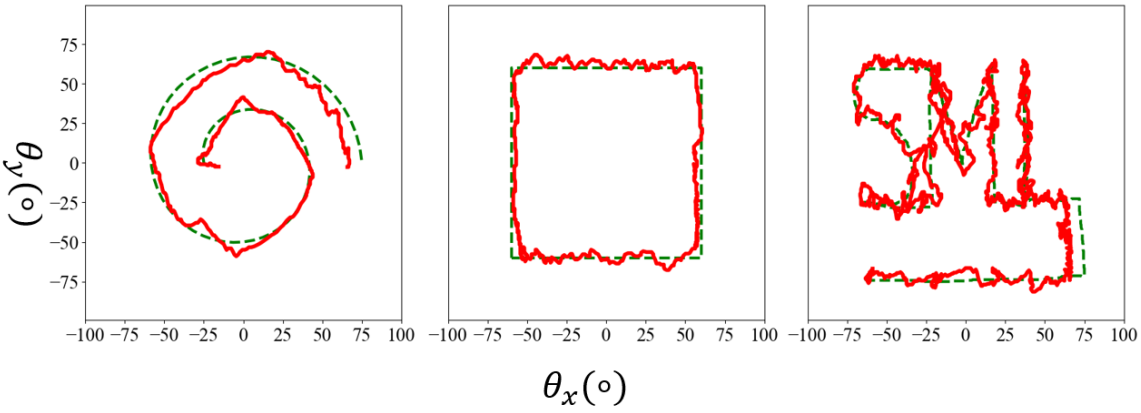


Figure 5.10: Recorded tracking trajectory of the policy trained with the full pretrained learned kinematics (LK Full) as the head.

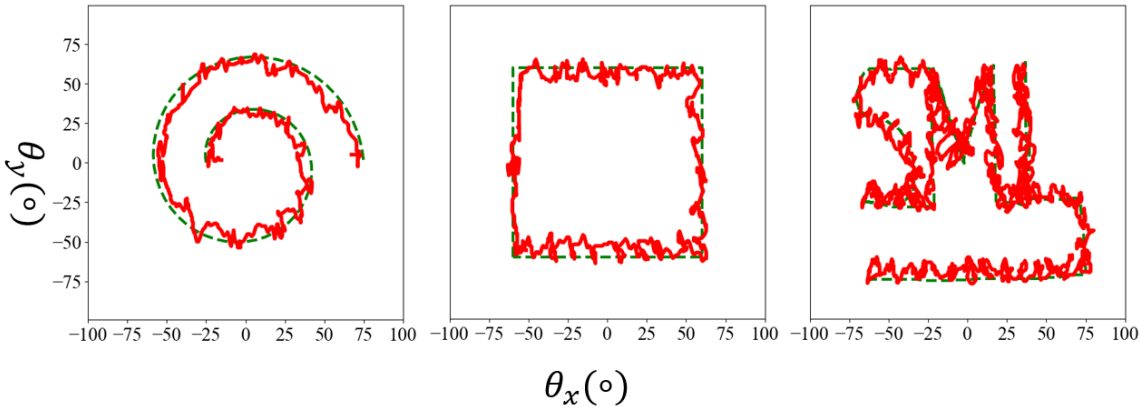


Figure 5.11: Recorded tracking trajectory of the policy trained without any feature extraction head (MLP No Head).

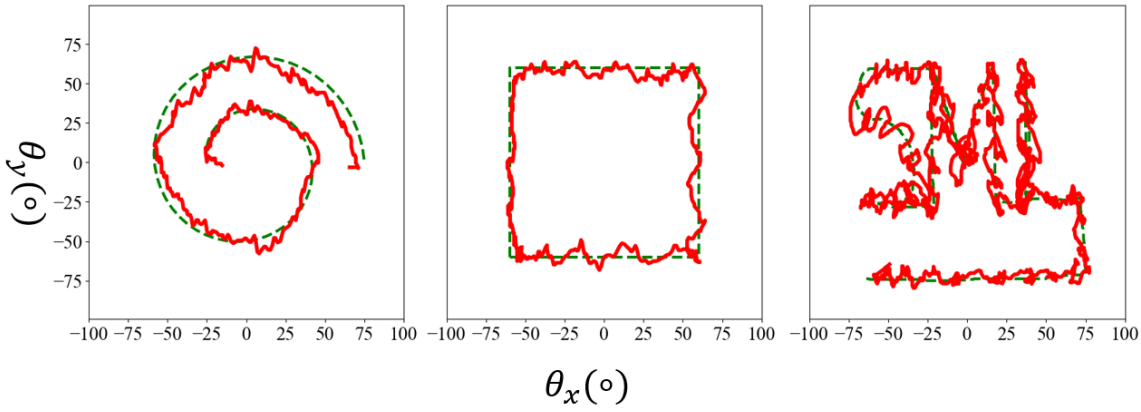


Figure 5.12: Recorded tracking trajectory using PID control. Gains were tuned to the best of our ability with $K_p = 2$, $K_d = 0.5$, and $K_i = 0.001$.

Table 5.4 shows the Root Mean Square Error (RMSE) between the recorded and reference trajectories for each model across the three patterns. The models trained with pretrained learned kinematics (LK Encoder and LK Full) performed better overall than both the MLP and PID baselines, achieving the lowest RMSE values across all patterns.

Furthermore, qualitative inspection of the recorded trajectories revealed that the motions of the robotic limbs were smoother and less jittery when using the pretrained head models, as compared to those driven by the MLP or PID controllers.

Robustness Testing

Since the policies are deployed in a closed-loop fashion on hardware, we also evaluated their ability to adapt to changes in intrinsic dynamics to assess generalizability and robustness. As *LK Head (Encoder)* performed the best in the trajectory tracking tasks, we selected the following three models and methods for testing: *LK Head (Encoder)* (referred to as Pretrained Encoder), *MLP No Head*, and *PID* as the baseline.

These models and methods were subjected to four different robustness tests: External Perturbation, Limb Swap (Blue), and External Loads (10g and 50g). Details of each test are described in Sec. 5.2.2. Fig. 5.13 shows the recorded trajectories for the external perturbation test. Fig. 5.14 presents results for the softer blue limb (Shore 30A), tested zero-shot despite the kinematic model being trained solely on the purple limb (Shore 40A). Figs. 5.15 and 5.16 show trajectories when 10g and 50g loads were added to the limb tip, respectively. Table 5.5 summarizes the RMSE between the recorded and reference (circular) trajectories for each test and model.

Quantitatively, from Table 5.5, the Pretrained Encoder was the most robust, achieving the lowest average RMSE of 5.78 across all tests. This supports the hypothesis that leveraging a pretrained time-series model enhances policy performance. However, the MLP No Head and PID baselines also performed competitively, with only about 1° higher RMSE than the Pretrained Encoder on average. This trend is also evident in the qualitative results.

Interestingly, in the softer limb and external load tests, PID's trajectories closely resemble those generated by the Pretrained Encoder, suggesting the encoder may be implicitly learning a near-optimal set of PID-like control gains. This observation

opens the door for future research into the relationship between classical PID control and learned policies based on pretrained encoders.

Test	LK (Encoder)	MLP No Head	PID
External Perturb.	**4.62**	5.18	5.53
Blue Limb	**3.83**	5.20	4.72
External Load (10g)	5.92	**5.22**	7.45
External Load (50g)	**8.75**	9.20	9.28
Average	**5.78**	6.20	6.75

Table 5.5: Comparison of RMSE values for each model under different robustness tests. Best RMSE values per test are bolded. For the external perturbation test, only trajectory segments before and after the perturbation are used to compute RMSE to evaluate recovery behavior.

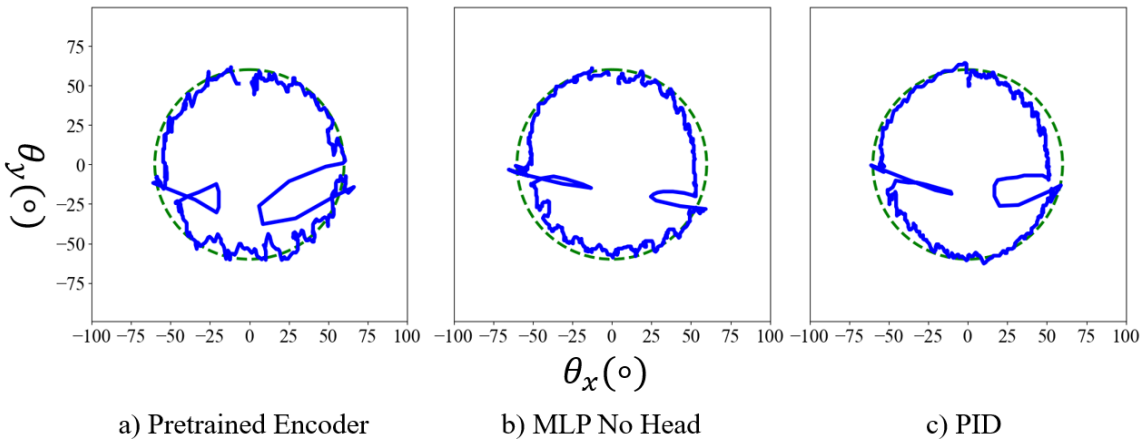


Figure 5.13: Tracking trajectories under external perturbation. Left: LK (Encoder), Center: MLP No Head, Right: PID. The shock occurs at 1/4 and 3/4 of the circular path.

5.4 Discussion

In this work, we proposed using a learned kinematic network as a simulator for training a goal-reaching policy via reinforcement learning. To further leverage the learned model, we incorporated it as a feature extractor for the policy network. Our

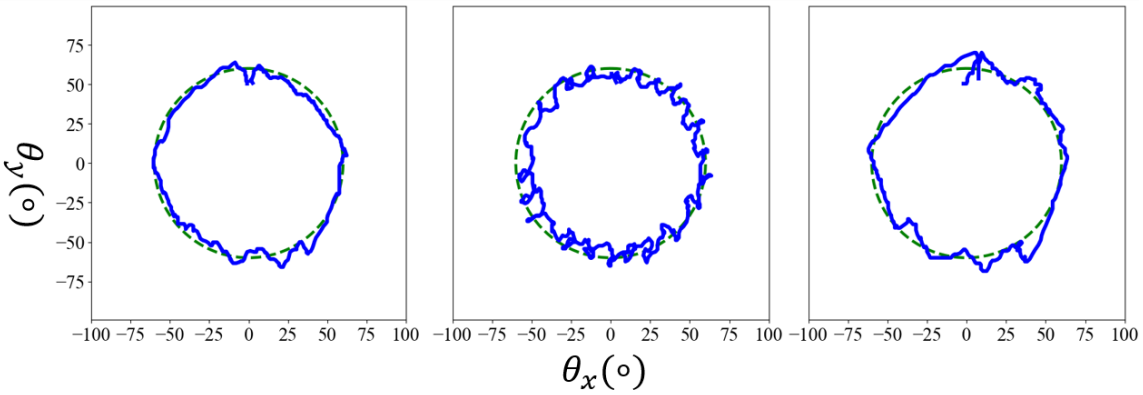


Figure 5.14: Tracking trajectories when deployed zero-shot on the softer blue limb. Left: LK (Encoder), Center: MLP No Head, Right: PID.

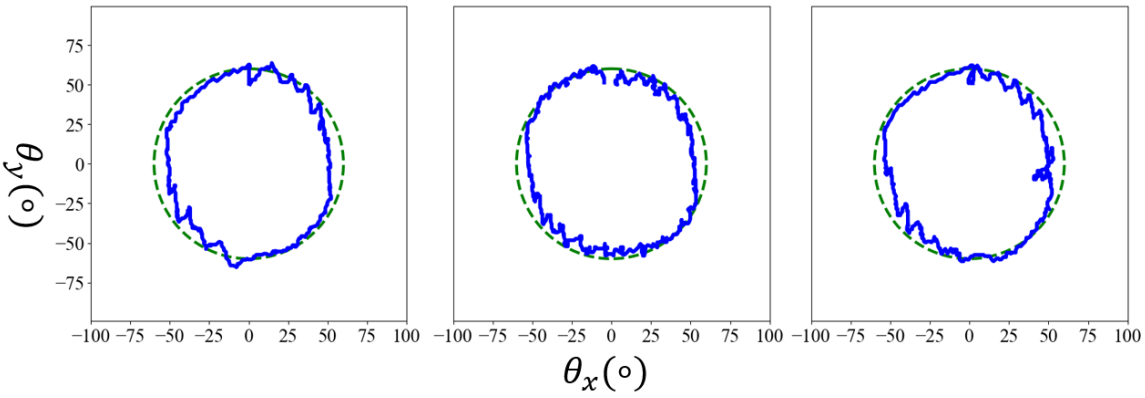


Figure 5.15: Tracking trajectories with a 10g external load added to the limb tip. Left: LK (Encoder), Center: MLP No Head, Right: PID.

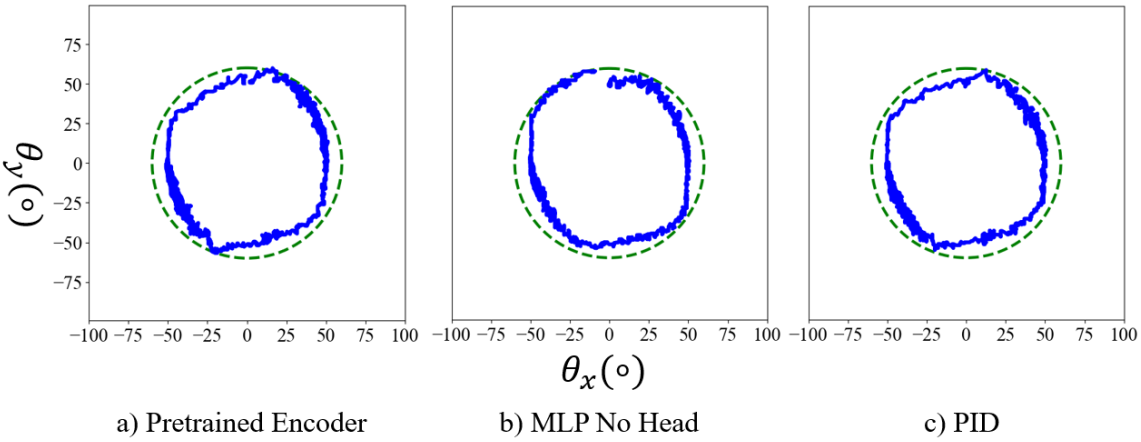


Figure 5.16: Tracking trajectories with a 50g external load added to the limb tip. Left: LK (Encoder), Center: MLP No Head, Right: PID.

results show that freezing the pretrained kinematic weights and treating the policy training as a fine-tuning task offers performance comparable to training from scratch, but at a significantly reduced training cost.

The learned policies demonstrated robust performance when deployed on hardware, showing strong generalization across various real-world conditions, including unseen dynamics. These results validate our framework as an efficient and effective method for training reinforcement learning controllers for soft robots with custom hardware platforms.

Chapter 6

Learning Safety Filter

In the previous sections, we proposed and validated a method for learning a goal-reaching policy for a soft robotic limb. While effective, the learned policies sometimes cause the limb to deviate from the desired path. Although harmless in our experiments, such deviations could be critical in real-world applications. For example, a soft robotic limb assisting in surgery must avoid sensitive areas of the body to prevent harm. Even in our own setup, excessive bending can overheat the SMA actuators, leading to permanent damage. To mitigate this, one can define regions of the state space—such as large bending angles—as *unsafe* and avoid them. In this section, we introduce simple modifications to standard SAC and Q-learning algorithms that allow the agent to learn safe and unsafe regions during training. A safety filter is then constructed from this knowledge to reject actions that might lead the system into unsafe states, improving both hardware safety and task reliability.[1]

[1]This section is a collaboration with Yogita Choudhury as she helped formulated and implemented half of the software required to realize the proposed method. This section is also adapted from our paper [52]

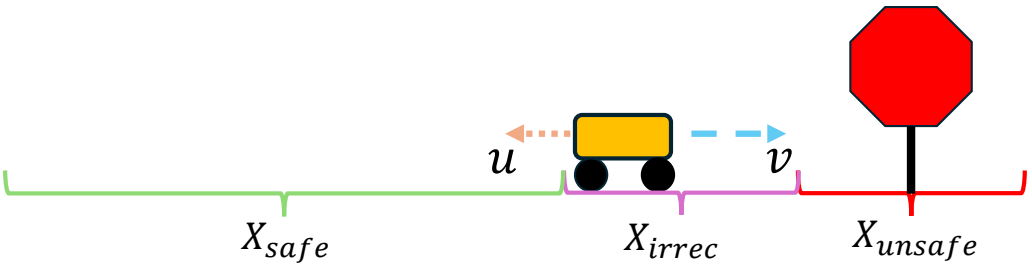


Figure 6.1: The full state space is broken down into three regions, $\mathcal{X}_{safe}$, $\mathcal{X}_{irrec}$, $\mathcal{X}_{unsafe}$. $\mathcal{X}_{safe}$ is the region the control input u can always be applied to prevent the system from entering $\mathcal{X}_{unsafe}$. $\mathcal{X}_{irrec}$ is the region where no control input can prevent entry into $\mathcal{X}_{unsafe}$. If a car is moving too fast and too close to the unsafe region, it is an example of the system being in the irrecoverable region.

6.1 Preliminaries

6.1.1 State Space Partitioning

Consider a car traveling along a road (Fig. 6.1) where areas like the curb are designated as unsafe states ($\mathcal{X}_{\text{unsafe}}$) due to hazards. If the car is traveling at 80 km/h and is only 1 m from the curb, a collision is inevitable even with maximum deceleration, defining this state as the irrecoverable state ($\mathcal{X}_{\text{irrec}}$). In contrast, if the car is 1 km away from the curb, there are various control actions that can ensure safety, categorizing this state as an absolutely safe state ($\mathcal{X}_{\text{safe}}$). Therefore, we can divide the state space $\mathcal{X}$ into three subspaces: $\mathcal{X}_{\text{safe}}$, $\mathcal{X}_{\text{irrec}}$ and $\mathcal{X}_{\text{unsafe}}$.

Identifying irrecoverable states is challenging due to system dynamics, but crucial as entering $\mathcal{X}_{\text{irrec}}$ leads to $\mathcal{X}_{\text{unsafe}}$.

6.1.2 Q Learning

We propose to use Q-learning for constructing a safety filter that filters hazardous actions and keeps the system in $\mathcal{X}_{safe}$. In Q-learning, the goal is to learn the expected discounted cumulative reward for a state-action pair in a Markov Decision Process (MDP) [54]. The Q-value is updated during training using the Bellman update rule,

under deterministic dynamics:

$$Q(x,u) = r(x,u,x') + \gamma \max_{u'} Q(x',u') \tag{6.1}$$

where r is the reward function and γ is the discount factor. The value function is defined as:

$$V(x) = \max_{u} Q(x,u) \tag{6.2}$$

We propose learning optimal Q-function and value function Q^*_{safe} and V^*_{safe}, respectively using a specialized reward function r_{safe}, where states in $\mathcal{X}_{safe}$ exceed a threshold ϵ_2. This threshold helps in determining whether actions from task specific nominal policies π_{task} are unsafe based on the learned Q and V functions, $\hat{Q}_{safe}$ and $\hat{V}_{safe}$ and replace with safer actions which provide the highest expected discounted cumulative safe reward.

6.2 Methodology

In this work, we propose a Q-learning-based, model-free reinforcement learning approach to construct a safety filter. A block diagram of the framework is shown in Fig. 6.2. Our method leverages the fact that $\hat{Q}_{safe}$ and $\hat{V}_{safe}$ can be learned off-policy, enabling the simultaneous training of a task-specific policy π_{task} and the safety policy π_{safe}.

The two policies are trained in parallel but remain decoupled by a gating mechanism that sorts observations into separate replay buffers based on episodic conditions. This decoupling allows π_{task} to be swapped with any π'_{task} during testing. Additionally, a gradient-guided search is used to optimize ϵ_2 for π_{filter}, ensuring good performance based on the learned $\hat{V}_{safe}$.

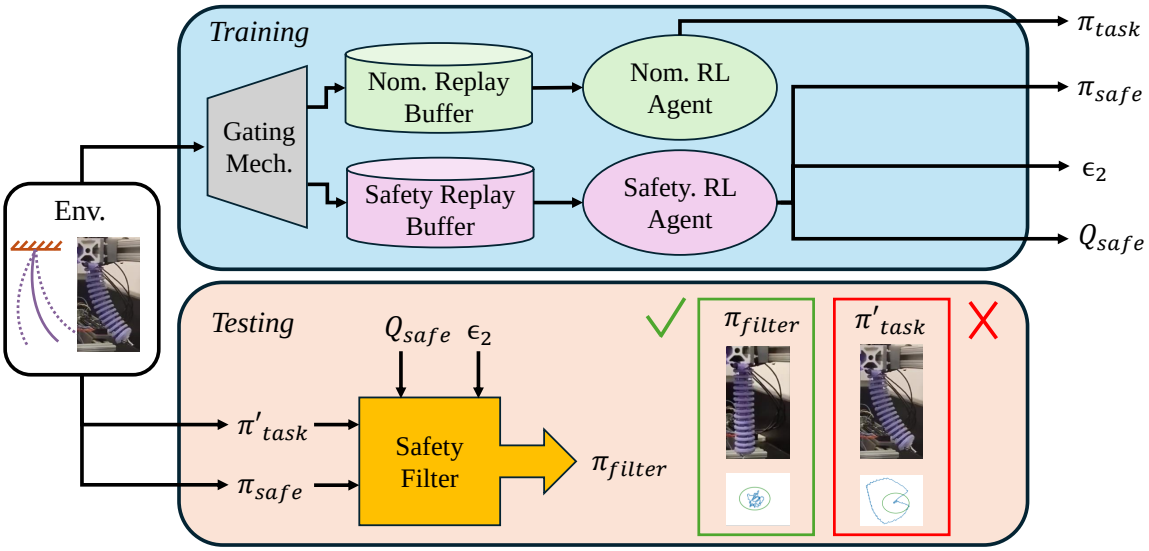


Figure 6.2: The block diagram shows our model-free RL-based safety filter framework. During training, environment observations are stored in the replay buffers of both task specific nominal and safety agents, enabling their simultaneous training. In testing, observations are processed by both policies, and the nominal action is filtered based on the safety agent's Q-function and threshold ϵ_2.

6.2.1 Reward Formulation

We define the safety reward function $r_{\text{safe}}(x, u, x')$ as:

$$r_{\text{safe}}(x, u, x') = \begin{cases} l(x), & x, x' \notin \mathcal{X}_{\text{unsafe}} \\ -\dfrac{1}{\gamma^t(1-\gamma)}, & x \notin \mathcal{X}_{\text{unsafe}},\ x' \in \mathcal{X}_{\text{unsafe}} \\ -1, & x, x' \in \mathcal{X}_{\text{unsafe}} \end{cases} \tag{6.3}$$

where x is the state comprising of time t, position and velocity. x' is the state the system transitions to after applying control u, and $\gamma \in (0, 1)$ is the discount factor. The function $l(x) \in (0, 1]$ increases as the system moves deeper into the safe region; for example:

$$l(x) = \frac{d(x)}{\max_{x \notin \mathcal{X}_{\text{unsafe}}} d(x)}, \tag{6.4}$$

where $d(x)$ is the positive signed distance to $\mathcal{X}_{\text{unsafe}}$. For x where $d(x) = 0$, i.e. on the boundary of $\mathcal{X}_{unsafe}$, we consider $x \in \mathcal{X}_{unsafe}$. For incorporating N safety constraints into the reward design, the distance is defined as $d(x) = \min\{d_1, d_2, ...d_N\}$ where d_i represents the positive signed distance to constraint i. Furthermore, we assume that the safe set is bounded by either safety or dynamic constraints.

During training, we optimize the $\hat{Q}_{\text{safe}}$ function using standard Bellman updates using r_{safe}.

Furthermore, we assume an episode terminates at the time step when the agent reaches the unsafe region or at maximum episode length, T. We consider a finite episode length scenario as from Eq. 6.3, when $x \in \mathcal{X}_{\text{safe}}$ and $x' \in \mathcal{X}_{\text{unsafe}}$, the reward can become unbounded if t is very large. Therefore, we define T as the episode length during training to bound the reward and prevent divergence.

Our design of r_{safe} aims to ensure a clear separation on $\hat{V}_{safe}$ between states in $\mathcal{X}_{\text{safe}}$, and those in $\mathcal{X}_{\text{irrec}}$. While an intuitive approach is to add a large negative penalty upon entering the unsafe region [61], fixed penalties diminish over time due to discounting, blurring the value separation between $\mathcal{X}_{\text{safe}}$ and $\mathcal{X}_{\text{irrec}}$. This reward formulation ensures a clear separation between $\mathcal{X}_{\text{safe}}$ and $\mathcal{X}_{\text{irrec}}$ in discounted cumulative rewards shown in the properties described below.

6.2.2 Reward Properties

The true value function V^*_{safe} or the maximum discounted cumulative reward has the following properties.

1. If $x \in \mathcal{X}_{safe}$, then the optimal value of the value function satisfies the following: $0 < V^*_{safe}(x) \leq \frac{1-\gamma^{T+1}}{1-\gamma}$.
2. If $x \in \mathcal{X}_{irrec}$, then the optimal value of the value function satisfy the following $-\frac{1+\gamma}{1-\gamma} \leq V^*_{safe}(x) < 0$.
3. If $x \in \mathcal{X}_{unsafe}$, then the optimal value of the value function satisfy the following $V^*_{safe}(x) = -\frac{1-\gamma^{T+1}}{1-\gamma}$.

These properties show that $V^*_{\text{safe}}(x)$ generally increases from $\mathcal{X}_{\text{unsafe}}$ to $\mathcal{X}_{\text{safe}}$, especially the transition from $\mathcal{X}_{\text{irrec}}$ to $\mathcal{X}_{\text{safe}}$. This indicates that the boundary between $\mathcal{X}_{\text{irrec}}$ and $\mathcal{X}_{\text{safe}}$ corresponds to a threshold value distinguishing these two regions in $V^*_{\text{safe}}(x)$. Fig. 6.3 depicts a conceptual visualization of the $V^*_{\text{safe}}(x)$.

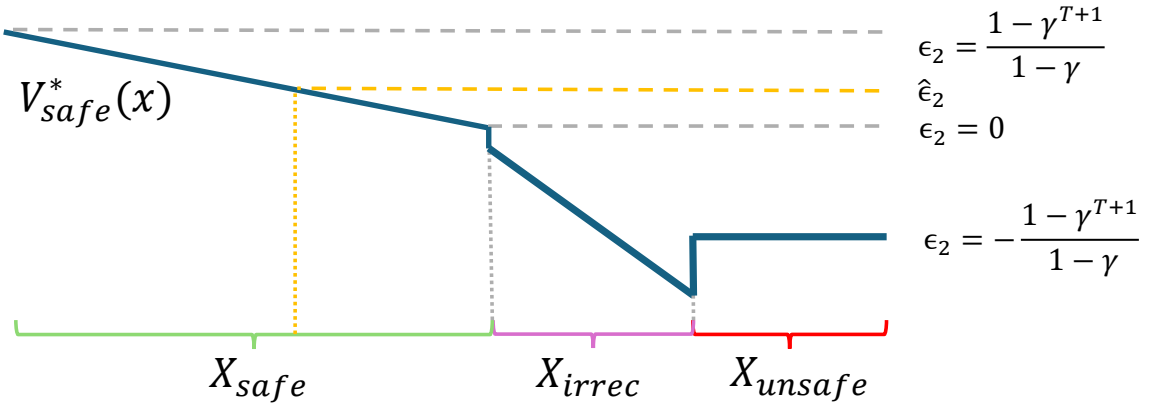


Figure 6.3: A 1 Dimensional concept visualization of how the V^*_{safe} could be like. $\epsilon_2 = 0$ is the threshold value that separates $\mathcal{X}_{\text{safe}}$ and $\mathcal{X}_{\text{irrec}}$. Because V^*_{safe} is increasing deeper inside $\mathcal{X}_{\text{safe}}$, picking a threshold value $\hat{\epsilon}_2$ that is higher than the optimal ϵ, will gives a more conservative estimate of the safe set.

From these properties, and the fact that $V^*_{safe}(x) = \max_{\mathbf{u}\in\mathcal{U}} Q^*_{safe}(x,u)$, where $\mathcal{U}$ denotes the set of permissible control inputs, leads to the following action filtering scheme to ensure safety:

$$\pi_{filter}(x) = \begin{cases} \pi_{task}(x), & Q^*_s(x, \pi_{task}(x)) > \epsilon_2 \\ \underset{u\in\mathcal{U}}{\text{argmax}}\, Q^*_s(x,u), & Q^*_s(x, \pi_{task}(x)) \leq \epsilon_2 \end{cases} \tag{6.5}$$

where ϵ_2 denotes the value threshold that separates $\mathcal{X}_{safe}$ and $\mathcal{X}_{unsafe}$. Q^*_s represents Q^*_{safe}, π_{task} is any policy trained to achieve a specific task under arbitrary task rewards r_{task}, and $\text{argmax}_{\mathbf{u}\in\mathcal{U}} Q^*_s(x,\mathbf{u})$ is the safety policy π_{safe}. π_{filter} refers to the filtered policy that ensures task completion while maintaining safety. For the optimal V^*_{safe}, we have $\epsilon_2 = 0$. Details regarding the theoretical proof and setup can be found in [52]

6.3 Implementation

To implement π_{filter} and keep the system within $\mathcal{S}_{\text{safe}}$, the task policy action is replaced by a greedy policy that maximizes the learned Q_{safe} under r_{safe}. We use Soft Actor-Critic (SAC) [27] for simulations with continuous action spaces and Deep Q-Learning

(DQN) [37] for real-world validation.

6.3.1 Simultaneous Training of Task Policy and Safety Policy

Leveraging the off-policy nature of SAC and DQN agents, we simultaneously train a task agent that maximizes task reward and a safety agent that maximizes the proposed safety reward. Both agents share the environment and rolled-out data but maintain independent replay buffers. A gating mechanism prevents the safety agent from collecting observations once the system enters the unsafe region. Initially, observations are added to both agents' replay buffers with their respective rewards. When the safety agent reaches the unsafe region, it stops receiving new observations, while the nominal agent continues until the episode ends. This approach minimizes interference with the task agent, preserving task performance, and ensures the safety policy remains valid for any task policy despite simultaneous training and shared data.

Due to the early termination assumption, the safety agent does not observe the unsafe region, as the episode terminates once the unsafe region is reached. Therefore a supervised loss, similar to the loss used in [55], is introduced to the safety agent in addition to the standard RL losses to classify whether a state belongs to $\mathcal{X}_{unsafe}$. The loss is defined as follows:

$$\mathcal{L}_{unsafe} = \|V_{safe}(x) + \frac{1}{1-\gamma}\|, \ \forall x \in \mathcal{X}_{unsafe} \tag{6.6}$$

where V_{safe} denotes the learned value function.

6.3.2 Searching for ϵ_2

Precisely converging to V^*_{safe} is challenging, and while the theoretical optimal value $\epsilon_2 = 0$ holds under ideal conditions, it may not represent the actual boundary for V_{safe}. Assuming the agent is sufficiently trained so that $V_{\text{safe}} \approx V^*_{\text{safe}}$ and has a similar shape, a threshold value separating $\mathcal{S}_{\text{safe}}$ and $\mathcal{X}_{\text{irrec}}$ should exist due to the increasing nature of V^*_{safe} from the irrecoverable to the safe region.

Algorithm 3 Gradient-Based Search for ϵ_2

Initialize x randomly such that $V_{\text{safe}}(x) > 0$
Set step size α, tolerance δ, and max episode length $T_{\max}$
$V_{\text{prev}}, l_{\text{prev}} \leftarrow -\infty$
while $|V_{\text{safe}}(x) - V_{\text{prev}}| > \delta$ **do**
    $V_{\text{prev}} \leftarrow V_{\text{safe}}(x)$
    Roll out the safety policy from state x for one episode
    $l \leftarrow$ length of the episode
    if $l = T_{\max}$ **then**
        if $l_{prev} \neq T_{max}$ **then**
            $\alpha \leftarrow 0.5 * \alpha$
        end if
        $x \leftarrow x - \alpha \nabla V_{\text{safe}}(x)$
    else
        if $l_{prev} = T_{max}$ **then**
            $\alpha \leftarrow 0.5 * \alpha$
        end if
        $x \leftarrow x + \alpha \nabla V_{\text{safe}}(x)$
    end if
    $l_{prev} \leftarrow l$
end while
$\epsilon_2 \leftarrow V_{\text{safe}}(x)$

To find this boundary ϵ_2 on V_{safe}, we propose a gradient-guided linear search method (Alg. 3). The algorithm determines whether a state x belongs to $\mathcal{S}_{\text{safe}}$ or $\mathcal{X}_{\text{irrec}}$ by rolling out the safety policy $\pi_{\text{safe}} = \arg\max_{\mathbf{u}\in\mathcal{U}} Q^*_{\text{safe}}(x, \mathbf{u})$ and checking if the system survives the maximum episode length T. If the system survives, x is likely in $\mathcal{S}_{\text{safe}}$, suggesting the boundary is less than $V_{\text{safe}}(x)$, prompting the algorithm to descend along the gradient at x. Conversely, if the system enters the unsafe region before T, x is likely in $\mathcal{X}_{\text{irrec}}$, and the algorithm ascends along the gradient to find ϵ_2.

Since V_{safe} is a neural network approximation and may not be smooth, the algorithm might output local optima. Therefore, we repeat the algorithm multiple times with random starting states and select the maximum ϵ_2 from these trials. The resulting ϵ_2 is used as the threshold in π_{filter}.

Due to stochasticity and imperfections in neural network approximations, particularly overestimation common in DQN, the threshold from Alg. 3 may not correspond to the actual boundary between safe and irrecoverable regions. However, it provides a useful guideline. Moreover, because π_{safe} is not guaranteed to be safe in suboptimal cases, raising ϵ_2 results in more conservative estimates of the safe region. This means π_{filter} will more frequently select actions from the safety policy, reducing the likelihood of reaching irrecoverable states. As shown in the Results section (Sec.6.4), by using the search algorithm's output as a starting point and manually increasing the threshold, our method can keep the system safe even when significant discrepancies exist between the learned and actual value functions.

6.4 Experiments and Results

We experimentally validate our method in simulations and on real hardware. Simulations involve a double integrator system for basic validation and a Dubin's car environment for static obstacle avoidance to compare with existing methods. On real hardware we demonstrated the filter's capability to constrain a soft robotic limb within a specified region.

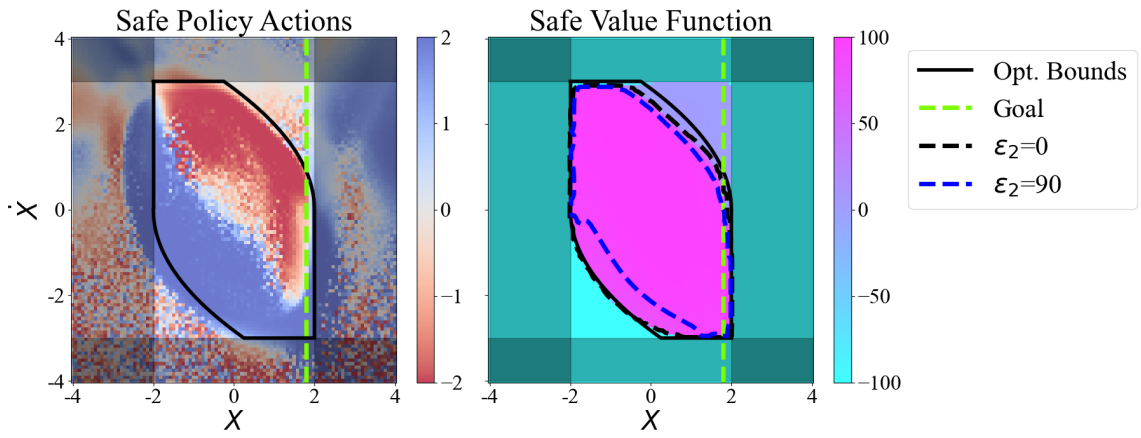


Figure 6.4: *Left*: The actions of the safe policy as a function of the state space. *Right*: The value function of the safety agent for double integrator. The dark shaded area is the unsafe region.

6.4.1 Simulation Environments

Double Integrator

The double integrator state is two-dimensional, with position x and velocity $\dot{x}$. Safe bounds are ± 2 m and ± 3 m/s, with absolute maximum bounds of ± 4 m and ± 5 m/s. The input acceleration is bounded between ± 2 m/s^2, and the task is to reach a goal at 1.8 m (neon green dashed line in Fig. 6.4).

The two-dimensional state allows easy visualization of the learned value function. In Fig. 6.4, safety policy actions (acceleration) are shown on the left, and the value function on the right. The solid black contour represents the analytical safe set [22]. The black dashed lines denote the learned safe set where $\epsilon_2 = 0$, which lies entirely within the analytical safe set. The blue dashed line represents the contour for $\epsilon_2 = 90$; as per our formulation, this contour is smaller than the $\epsilon_2 = 0$ contour, leading to more conservative estimates of $\mathcal{S}_{\text{safe}}$, which is observed empirically.

Moreover, on the left of Fig. 6.4, at the boundary of the analytical safe set, the actions align with intuition: when approaching the right boundary at high speed, the action is a leftward force (red), and vice versa. These empirical results are consistent with our theoretical expectations.

Dubin's car

Because the double integrator's goal lies within the safe region, the nominal agent remains safe by simply achieving the task, which doesn't demonstrate the full effect of our safety filter. Therefore, we tested our method in the Dubin's car environment, featuring complex nonlinear dynamics and conflicts between the nominal task and safety, e.g. the shortest path to goal goes through $\mathcal{X}_{unsafe}$. In this environment, safety bounds are ± 2 m with a central keep-out region of radius 1 m. The absolute maximum bounds are defined with an input velocity of 1.2 m/s, aiming for a goal at (1.8 m, 1.8 m) with a radius of 0.5 m. As shown in Tab. 6.1, our method's performance is comparable to, and sometimes better than, other common safe RL algorithms.

In Tab. 6.1, we compare our filtered policy (safety filter with co-trained nominal policy) against Lagrangian Relaxation (LR), Risk Sensitive Policy Optimization (RSPO) [60], the unconstrained task policy, and a task policy trained with a large penalty for entering the unsafe region (Reward Penalty). Results are the mean and standard deviation over 10 runs with different seeds. Using $\epsilon_2 = 67.38$ found by Alg. 3 for our safety threshold, our method attains the highest average episodic return among methods achieving a 100% safety rate (fraction of episodes reaching the maximum length without safety violations out of 100 episodes).

We also tested our filter with nominal policies not co-trained with our safety agent (Tab. 6.2): PPO [47], DDPG [34], TD3 [23], and a uniform random policy. Over 10 runs with different seeds, only PPO had safety violations, with an average safety rate of 98.8%. We attribute this slightly less than perfect safety rate to the stochastic nature of our safety filter trained with SAC as it samples from a learned distribution to output an action.

As shown in Fig. 6.5, there is a clear trend between ϵ_2 and performance. Increasing ϵ_2 results in greater safety but reduced nominal rewards, aligning with our theoretical prediction that higher ϵ_2 leads to a more conservative method. The blue cross indicates $\hat{\epsilon}_2$, the value found by Alg. 3.

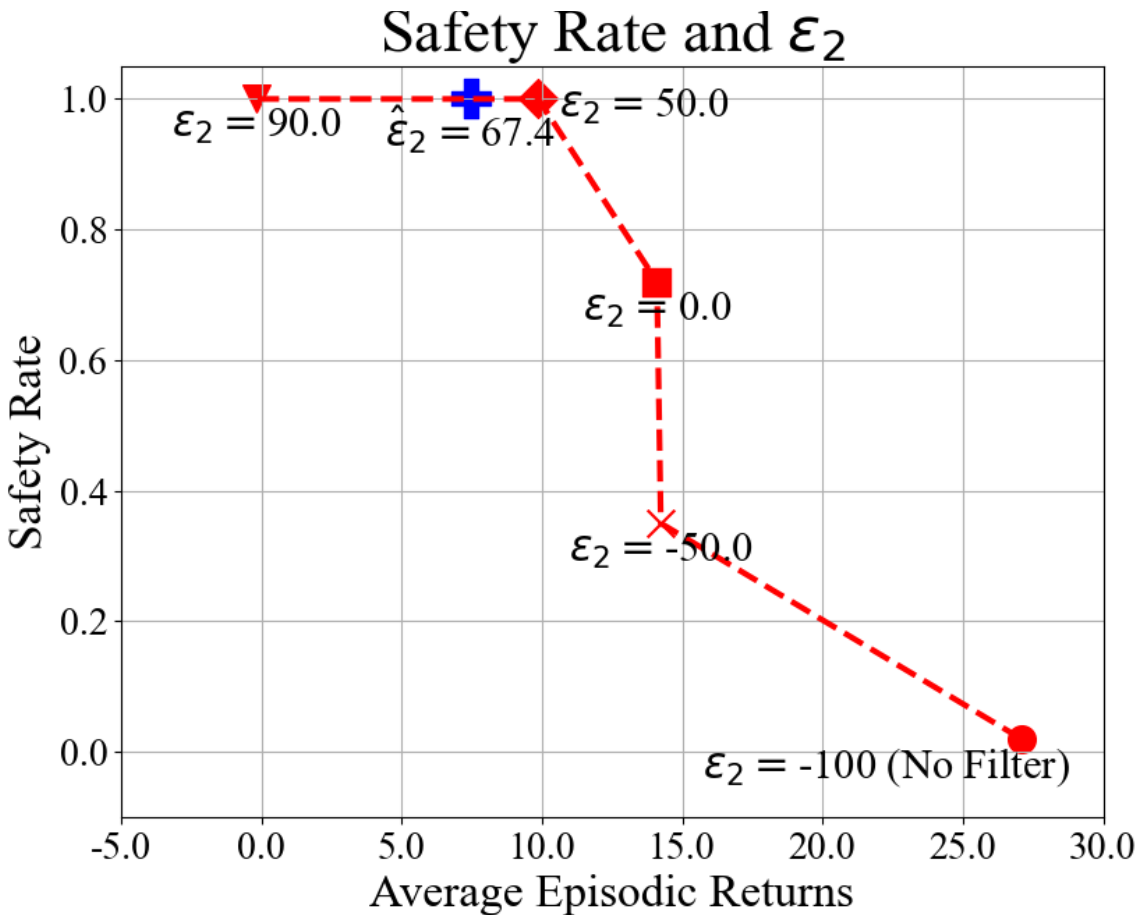


Figure 6.5: Performance of our filter evaluated by average episodic return and safety rate at different ϵ_2 levels in the Dubin's car environment.

Table 6.1: Performance comparison of our safety filter to other safe model-free reinforcement learning methods.

	Average Episodic Return	**Safety Rate**
Unconstrained Task Policy	46.38 ± 20.37	0.017 ± 0.051
RSPO	13.37 ± 4.61	0.855 ± 0.138
Reward Penalty	-2.10 ± 8.49	1.000 ± 0.000
LR	2.84 ± 2.34	1.000 ± 0.000
Filtered Policy	***7.08 ± 3.37***	***1.000 ± 0.000***

Table 6.2: Safety Rate of filtering different policies

	Safety Rate	**Std. Safety Rate**
Co-trained Policy	1.0	0.0
PPO	0.988	0.023
DDPG	1.0	0.0
TD3	1.0	0.0
Random	1.0	0.0

6.5 Hardware Experiments

6.5.1 Data Setup

To enable efficient training, instead of naively deploying RL training on the hardware, we first create a data driven simulator to approximate the system dynamics. The training process for creating the simulator is detailed in Section 4. However, due to the complexity of the dynamics, the best dynamic model we were able to train had a root mean square error (RMSE) of approximately $8° - 15°$. All RL procedures are trained in this simulated environment and then zero-shot deployed to the real hardware device without fine-tuning.

6.5.2 RL Setup

To validate our method, we trained the safety policy using DQN with the observation space $x = [\theta_x, \theta_y, \dot{\theta}_x, \dot{\theta}_y]$ and a discrete action space -move upper right (u_1), upper left (u_2), lower left (u_3), and lower right (u_4)—as depicted in Fig. 3.1. The safety policy was trained using our approximated dynamics simulator to constrain movements within $\pm 30°$ on both x and y axes. The constraint boundary is shown as a green circle in the right figure of Fig. 3.1, i.e., $\mathcal{X}_{\text{safe}}$ represents states where $\|\theta\| < 30°$.

For the nominal policy, we used a predefined action sequence: u_1 for 3 seconds, u_4 for 3 seconds, u_3 for 6 seconds, u_2 for 12 seconds, u_1 for 6 seconds, and u_3 for 3 seconds, aiming to trace a diamond-shaped trajectory. The real-life rollout of this sequence is shown in orange in Fig. 6.6. The task of the safety filter is to constraint the limb movement within the $\|\theta\| < 30°$ boundary.

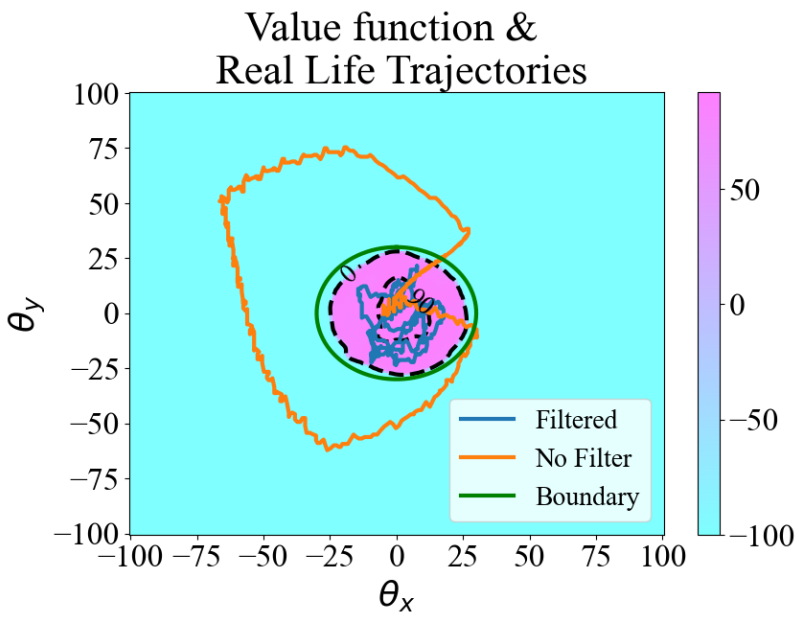


Figure 6.6: The real-life recorded limb trajectory overlaid on the learned value function. Note that the value function is learned through simulation and is a 2D projection of a 4D function, whereas the trajectory values are recorded in real life. The specific value function visualization is generated by setting the velocity terms in the states to 0. The 0 and 90 threshold contour is denoted by dark dashed lines.

6.5.3 Hardware Results

As shown in Fig. 6.6, the safety policy successfully keeps the limb within the safe region, as the blue trajectories, which represent the recorded limb trajectory with our safety filter, are entirely contained within the safe region denoted by the green circle. However, due to the sim-to-real gap between the learned dynamics and the real limb dynamics, $\epsilon_2 = 90$ was required for our scheme to work in real life, resulting in a very conservative filter. Despite this conservatism, the empirical results validate the robustness of our method in maintaining system safety, even in the presence of a significant sim-to-real gap.

Additionally, in Fig. 6.7, we observe that as ϵ_2 increases, the system becomes "safer", meaning it takes longer before reaching the unsafe region, aligning with the theoretical predictions and demonstrates the robustness of our approach.

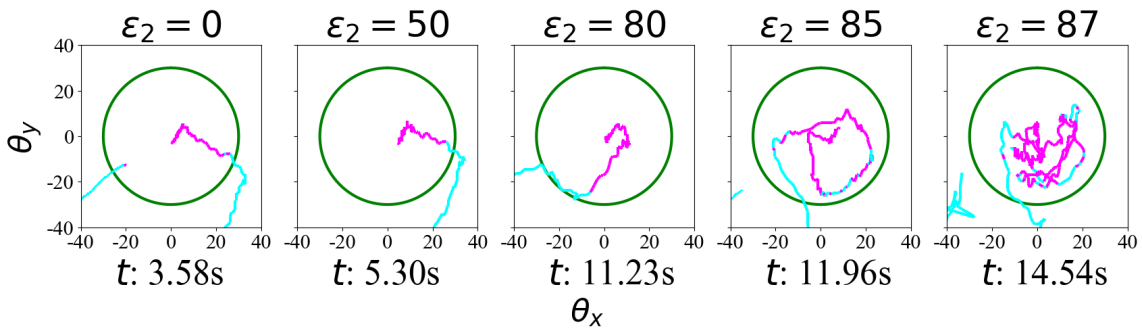


Figure 6.7: Visualization of recorded trajectories for different values of ϵ_2. t denotes the average amount of time before the limb reaches the unsafe region for a total of 5 hardware trials for the specific value of ϵ_2 it is under. The magenta segments denotes segments of the trajectory where the value function is above the specific ϵ_2, the cyan denotes segments of trajectory below the specific ϵ_2. Note that the value function computed here uses real velocity information as well unlike Fig.6.6

6.6 Discussion

In this section, we propose a reinforcement learning approach that uses reward shaping to distinguish between safe, unsafe, and irrecoverable states. Central to our method is a policy filter that provides theoretical safety guarantees under ideal conditions and can be adapted—through threshold tuning—to perform effectively even in less-than-optimal settings, such as environments with significant sim-to-real discrepancies.

Our framework enables the concurrent training of both a task-specific policy and a safety policy, the latter of which is designed to generalize across various task policies. In simulation benchmarks, our method performs on par with existing approaches and is further validated on a real-world, highly nonlinear soft silicone limb. This demonstrates the method's practicality, improved training efficiency, and broader applicability compared to prior work.

Chapter 7

Conclusion

7.1 Limitations and Future Work

Although the proposed framework achieved strong results, it is not without limitations. For instance, in the learned forward kinematics model, performance degrades when the robotic limb approaches the edge of the state boundary. Additionally, the model struggles when subjected to abrupt input changes. These issues stem from limitations in the sampling strategy. While our proposed count-based exploration outperforms naive random motor wobbling, it still has room for significant improvement, making it a promising direction for future research.

Furthermore, the current forward model only considers kinematics (i.e., position and velocity), yet in robotics, force understanding is equally critical. Extending the model to include force information could improve generalizability and utility, particularly in real-world applications involving contact and compliance.

In terms of trajectory tracking, the current approach only focuses on reaching discrete goal positions. As seen in some recorded trajectories, this can cause the limb to overshoot and form unnecessary loops. Future work could explore training directly on trajectory-following tasks, possibly by incorporating temporal information into the reward function or the waypoint controller. Again, integrating force data into this task could further enhance the framework's effectiveness and real-world applicability in soft robotic control.

Regarding the safety filter, while decoupling task and safety policies improves

generalization, it can lead to issues in scenarios where safety and task objectives conflict—potentially resulting in deadlock. Future work could investigate safety filter designs that retain generalizability while minimizing task constraints. Moreover, although the proposed approach performs well under suboptimal conditions, it currently lacks formal safety guarantees. Extending the framework to provide such guarantees—even when operating under imperfect policies—presents an important and impactful avenue for future research.

7.2 Summary

This work presented a complete data-driven framework for learning to control soft robotic systems, with real-world validation on a custom soft robotic limb platform actuated by shape memory alloys (SMAs). We demonstrated a process for learning an expressive forward kinematics model using a count-based exploration strategy, which enabled efficient sampling of the state space. This allowed us to train a Seq2Seq model capable of producing accurate predictions over long rollouts, even in the presence of challenges such as hysteresis and material variability. We also showed that this model could be adapted to limbs of different stiffness through a simple, data-efficient fine-tuning procedure.

Building on this model, we trained reinforcement learning-based controllers that could perform goal-reaching tasks with high accuracy and robustness in real-world conditions. By using the learned model as a feature extractor and treating policy learning as a fine-tuning problem, we outperformed policies trained with standard MLP architectures. Further improvements—including a power proxy and dynamic goal tolerance—enhanced the controller's robustness to environmental variability.

Finally, we introduced a model-free safety filter that integrates seamlessly into existing reinforcement learning pipelines through minor algorithmic modifications and a novel reward formulation. Our approach enables practical safety enforcement—even across large sim-to-real gaps—by adjusting a simple threshold parameter. We also proposed a method for determining a good initial value for this threshold.

Together, these contributions form a generalizable strategy for controlling soft robots with complex, nonlinear dynamics—especially in scenarios where analytical modeling or high-fidelity simulation is impractical.

Appendix A

Appendix

SAC Hyperparameter	Value
total timesteps	100,000
buffer size	1,000,000
gamma	0.99
tau	0.005
batch size	256
policy learning rate	5e-4
Q network learning rate	5e-4
policy network update frequency	2
target network update frequency	2
alpha	0.2
number of parallelized environments	10
max gradient size (clip norm)	1.0
sequence window length	100

Table A.1: Hyperparameters used to train SAC for goal reaching policy. Both the Actor and Q Network are MLPs with a hidden layer of size 256.

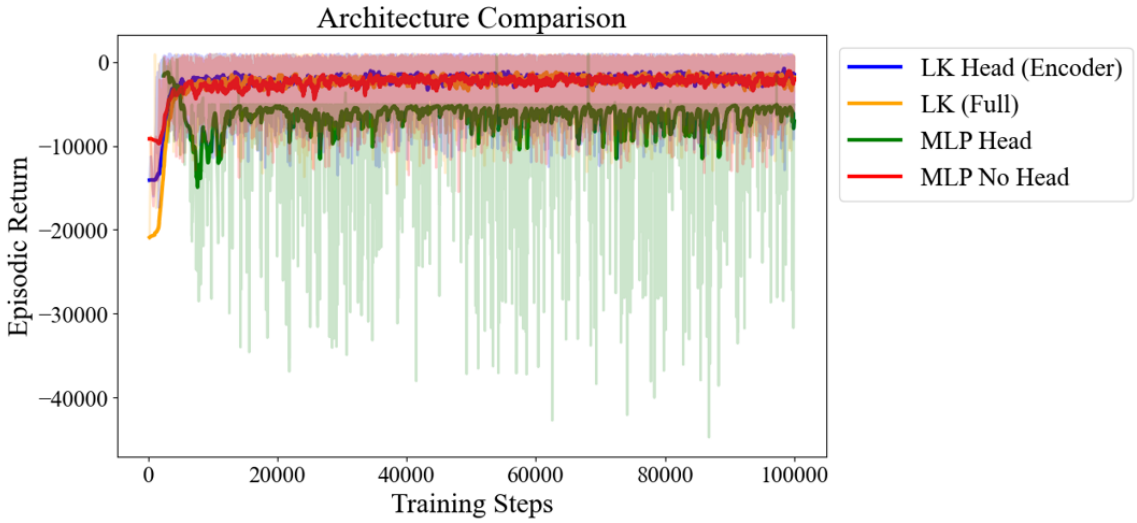


Figure A.1: The episodic returns over 100k training steps for policies using different neural network architectures. The bolded lines are the smoothed curves with a smoothing factor of $\alpha = 0.1$, and the faint lines are the raw values. Because the return fluctuates a lot, it is extremely difficult to spot any differences without smoothing.

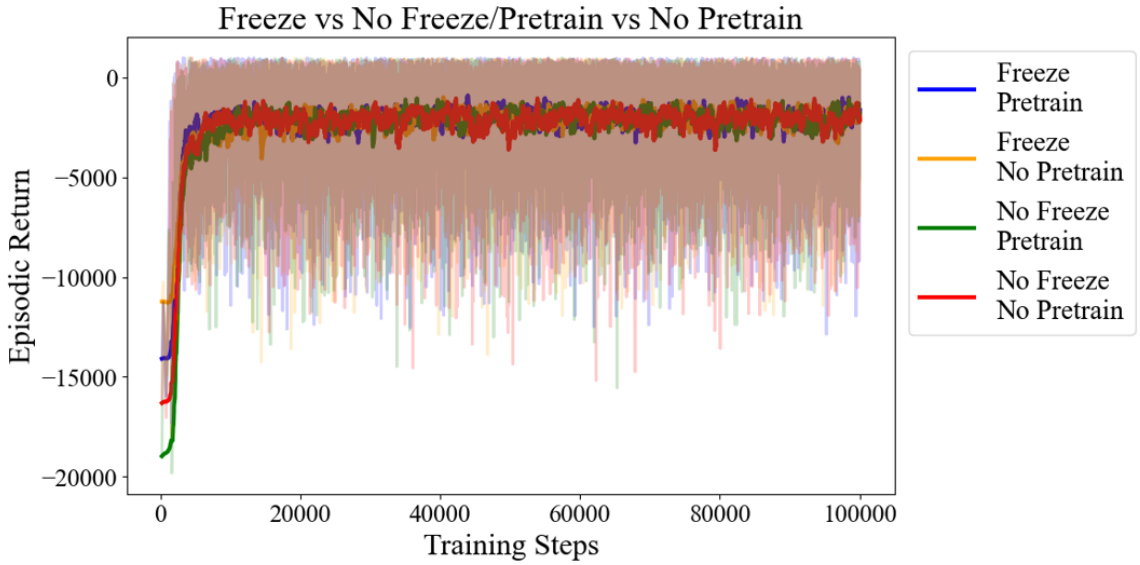


Figure A.2: The episodic returns over 100k training steps for policies trained with combinations of freeze and pretrained options. Due to the large fluctuations, it is extremely difficult to spot any differences even with smoothing.

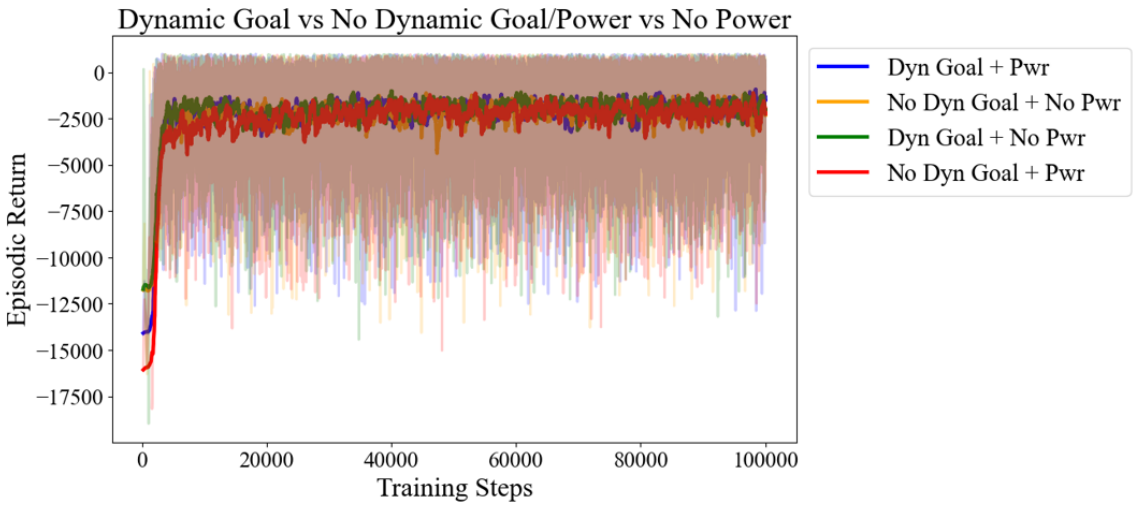


Figure A.3: The episodic returns over 100k training steps for policies trained with combinations of dynamic goal tolerance and power options. Due to the large fluctuations, it is extremely difficult to spot any differences even with smoothing.

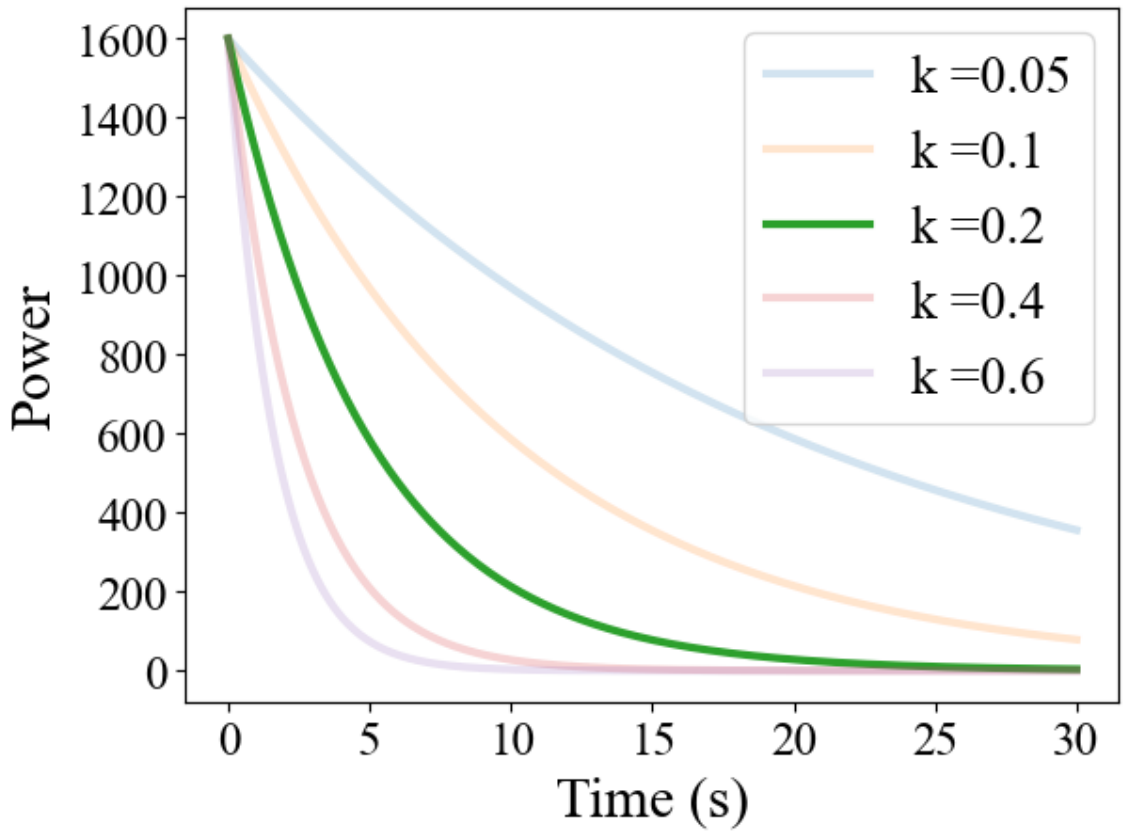


Figure A.4: The cool down curve of the power proxy using different values of k. We start with a power of around 1600 which is about constant on for an SMA for 40 seconds, then from empirical experiments, we notice the SMA cools down to around room temperature in 30 seconds, which corresponds to the $k = 0.2$ curve.

Bibliography

[1] Joshua Achiam and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. 2019. URL https://api.semanticscholar.org/CorpusID:208283920. 2.2.3

[2] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1999. 2.2.3

[3] Arduino. Arduino uno, 2010. URL https://www.arduino.cc/en/hardware/uno. Accessed on April 8, 2025. (document), 5.4

[4] Yarden As, Ilnura Usmanova, Sebastian Curi, and Andreas Krause. Constrained policy optimization via bayesian world models, 2022. URL https://arxiv.org/abs/2201.09802. 2.2.3

[5] Uljad Berdica, Matthew Jackson, Niccolò Enrico Veronese, Jakob Foerster, and Perla Maiolino. Reinforcement learning controllers for soft robots using learned environments. In *2024 IEEE 7th International Conference on Soft Robotics (RoboSoft)*, pages 933–939. IEEE, 2024. 2.2.1

[6] Charles M. Best, Morgan T. Gillespie, Phillip Hyatt, Levi Rupert, Vallan Sherrod, and Marc D. Killpack. A new soft robot control method: Using model predictive control for a pneumatically actuated humanoid. *IEEE Robotics and Automation Magazine*, 23(3):75–84, 2016. doi: 10.1109/MRA.2016.2580591. 1

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540. 5.1.1

[8] Daniel Bruder, Xun Fu, R. Brent Gillespie, C. David Remy, and Ram Vasudevan. Data-driven control of soft robots using koopman operator theory. *IEEE Transactions on Robotics*, 37(3):948–961, 2021. doi: 10.1109/TRO.2020.3038693. 2.1.2

[9] Agustin Castellano, Hancheng Min, Juan Andrés Bazerque, and Enrique Mallada. Learning safety critics via a non-contractive binary bellman operator, 2024. URL https://arxiv.org/abs/2401.12849. 2.2.3

[10] Andrea Centurelli, Luca Arleo, Alessandro Rizzo, Silvia Tolu, Cecilia Laschi, and Egidio Falotico. Closed-loop dynamic control of a soft manipulator using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):4741–4748, 2022. doi: 10.1109/LRA.2022.3146903. 4.1.1, 4.2.2

[11] Andrea Centurelli, Luca Arleo, Alessandro Rizzo, Silvia Tolu, Cecilia Laschi, and Egidio Falotico. Closed-loop dynamic control of a soft manipulator using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):4741–4748, 2022. doi: 10.1109/LRA.2022.3146903. 2.2.2

[12] Andrew Choi, Ran Jing, Andrew P. Sabelhaus, and Mohammad Khalid Jawed. Dismech: A discrete differential geometry-based physical simulator for soft robots and structures. *IEEE Robotics and Automation Letters*, 9(4):3483–3490, 2024. doi: 10.1109/LRA.2024.3365292. 1, 2.1.1

[13] E. Coevoet, T. Morales-Bieze, F. Largilliere, Z. Zhang, M. Thieffry, Mario Sanz-Lopez, B. Carrez, Damien Marchal, Olivier Goury, J. Dequidt, and Christian Duriez. Software toolkit for modeling, simulation, and control of soft robots. *Advanced Robotics*, 31:1–17, 11 2017. doi: 10.1080/01691864.2017.1395362. 1, 2.1.1

[14] William Coral, Claudio Rossi, Oscar M Curet, and Diego Castro. Design and assessment of a flexible fish robot actuated by shape memory alloys. *Bioinspiration & biomimetics*, 13(5):056009, 2018. 3

[15] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016. 2.1.1

[16] Zihang Dai. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019. 4.2.4

[17] Cosimo Della Santina, Robert Katzschmann, Antonio Bicchi, and Daniela Rus. Dynamic control of soft robots interacting with the environment. 04 2018. doi: 10.1109/ROBOSOFT.2018.8404895. 1

[18] Richard Desatnik, Zach J Patterson, Przemysław Gorzelak, Samuel Zamora, Philip LeDuc, and Carmel Majidi. Soft robotics informs how an early echinoderm moved. *Proceedings of the National Academy of Sciences*, 120(46):e2306580120, 2023. 3

[19] Richard Desatnik, Mikhail Khrenov, Zachary Manchester, Philip LeDuc, and Carmel Majidi. Optimal control for a shape memory alloy actuated soft digit using iterative learning control. In *2024 IEEE 7th International Conference on Soft Robotics (RoboSoft)*, pages 48–54. IEEE, 2024. 2.1.2

[20] Jingliang Duan, Zhengyu Liu, Shengbo Eben Li, Qi Sun, Zhenzhong Jia, and Bo Cheng. Adaptive dynamic programming for nonaffine nonlinear optimal control problem with state constraints. *Neurocomputing*, 484:128–141, May 2022.

ISSN 0925-2312. doi: 10.1016/j.neucom.2021.04.134. URL http://dx.doi.org/10.1016/j.neucom.2021.04.134. 2.2.3

[21] François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, Stéphanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, and Stéphane Cotin. *SOFA: A Multi-Model Framework for Interactive Physical Simulation*, pages 283–321. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-29014-5. doi: 10.1007/8415_2012_125. URL https://doi.org/10.1007/8415_2012_125. 2.1.1

[22] Jaime F Fisac, Neil F Lugovoy, Vicenç Rubies-Royo, Shromona Ghosh, and Claire J Tomlin. Bridging hamilton-jacobi safety analysis and reinforcement learning. In *2019 Int. Conf. on Robot. and Automat. (ICRA)*, pages 8550–8556. IEEE, 2019. 6.4.1

[23] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477. 5.1.2, 6.4.1

[24] M Gazzola, LH Dudte, AG McCormick, and L Mahadevan. Forward and inverse problems in the mechanics of soft filaments. *Royal Society open science*, 5(6): 171628, 2018. doi: 10.1098/rsos.171628. URL https://doi.org/10.1098/rsos.171628. 2.1.1

[25] Moritz A. Graule, Thomas P. McCarthy, Clark B. Teeple, Justin Werfel, and Robert J. Wood. Somogym: A toolkit for developing and evaluating controllers and reinforcement learning algorithms for soft robots. *IEEE Robotics and Automation Letters*, 7(2):4071–4078, 2022. doi: 10.1109/LRA.2022.3149580. 2.1.1

[26] Yang Guan, Yangang Ren, Qi Sun, Shengbo Eben Li, Haitong Ma, Jingliang Duan, Yifan Dai, and Bo Cheng. Integrated decision and control: Towards interpretable and computationally efficient driving intelligence, 2021. URL https://arxiv.org/abs/2103.10290. 2.2.3

[27] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290. 2.2.2, 5.1.2, 6.3

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. 4.2.4

[29] Homayoun Honari, Mehran Ghafarian Tamizi, and Homayoun Najjaran. Safety optimized reinforcement learning via multi-objective policy optimization, 2024. URL https://arxiv.org/abs/2402.15197. 2.2.3

[30] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. Cleanrl: High-quality single-file implementations of deep reinforcement learning

algorithms, 2021. URL https://arxiv.org/abs/2111.08819. 5.1.2

[31] Phillip Hyatt, Curtis C Johnson, and Marc D Killpack. Model reference predictive adaptive control for large-scale soft robots. *Frontiers in Robotics and AI*, 7: 558027, 2020. 1

[32] Dynalloy Inc. Technical characteristics of flexinol actuator wires. Rev J. 3

[33] Hu Jin, Erbao Dong, Min Xu, Chunshan Liu, Gursel Alici, and Yang Jie. Soft and smart modular structures actuated by shape memory alloy (sma) wires as tentacles of soft robots. *Smart Materials and Structures*, 25(8):085026, 2016. 3

[34] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. URL https://arxiv.org/abs/1509.02971. 5.1.2, 6.4.1

[35] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021. 4.2.4

[36] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074. URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074. 5.2

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602. 2.2.2, 6.3

[38] Ryota Morimoto, Satoshi Nishikawa, Ryuma Niiyama, and Yasuo Kuniyoshi. Model-free reinforcement learning with ensemble for a soft continuum robot arm. In *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*, pages 141–148, 2021. doi: 10.1109/RoboSoft51838.2021.9479340. 2.2.2

[39] Zach J. Patterson, Andrew P. Sabelhaus, Keene Chin, Tess Hellebrekers, and Carmel Majidi. An untethered brittle star-inspired soft robot for closed-loop underwater locomotion. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8758–8764, 2020. doi: 10.1109/IROS45743.2020.9341008. 3

[40] Zach J. Patterson, Andrew P. Sabelhaus, and Carmel Majidi. Robust control of a multi-axis shape memory alloy-driven soft manipulator. *IEEE Robotics and Automation Letters*, 7(2):2210–2217, 2022. doi: 10.1109/LRA.2022.3143256. 2.1.2

[41] Zach J. Patterson, Wei Xiao, Emily Sologuren, and Daniela Rus. Safe control

for soft-rigid robots with self-contact using control barrier functions. In *2024 IEEE 7th International Conference on Soft Robotics (RoboSoft)*, pages 151–156, 2024. doi: 10.1109/RoboSoft60065.2024.10522000. 2.2.3

[42] Steven I Rich, Robert J Wood, and Carmel Majidi. Untethered soft robotics. *Nature Electronics*, 1(2):102–112, 2018. 3

[43] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987. 4.2.4

[44] Andrew P Sabelhaus, Rohan K Mehta, Anthony T Wertz, and Carmel Majidi. In-situ sensing and dynamics predictions for electrothermally-actuated soft robot limbs. *Frontiers in Robotics and AI*, 9:888261, 2022. 2.2.1, 4.1.1, 4.2.2

[45] Andrew P. Sabelhaus, Zach J. Patterson, Anthony T. Wertz, and Carmel Majidi. Safe supervisory control of soft robot actuators. *Soft Robotics*, 11(4):561–572, 2024. doi: 10.1089/soro.2022.0131. URL https://doi.org/10.1089/soro.2022.0131. PMID: 38324015. 2.2.3

[46] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017. URL https://arxiv.org/abs/1502.05477. 2.2.2

[47] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. 2.2.2, 5.1.2, 6.4.1

[48] Jing Shu, Junming Wang, Sanders Cheuk Yin Lau, Yujie Su, Kelvin Ho Lam Heung, Xiangqian Shi, Zheng Li, and Raymond Kai-yu Tong. Soft robots' dynamic posture perception using kirigami-inspired flexible sensors with porous structures and long short-term memory (lstm) neural networks. *Sensors*, 22(20): 7705, 2022. 2.2.1

[49] Smooth-On. Moldstar 15,16, and 30 1a:1b mix by volume platinum silicone rubbers. https://www.smooth-on.com/, 2023. 3

[50] Smooth-On. Smooth-sil series additional cure silicone rubber compound. https://www.smooth-on.com/, 2024. 3

[51] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic, 2020. URL https://arxiv.org/abs/2010.14603. 2.2.3

[52] Guo Ning Sue, Yogita Choudhary, Richard Desatnik, Carmel Majidi, John Dolan, and Guanya Shi. Q-learning-based model-free safety filter, 2024. URL https://arxiv.org/abs/2411.19809. 1, 6.2.2

[53] I Sutskever. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014. 4.2.4

[54] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html. 6.1.2

[55] Daniel CH Tan, Fernando Acero, Robert McCarthy, Dimitrios Kanoulas, and Zhibin Li. Value functions are control barrier functions: Verification of safe policies using control theory. 6.3.1

[56] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017. 4.2.2

[57] NittoBend Technologies. Soft angular displacement sensor theory manual, 2018. 3, 4.4

[58] Clark B. Teeple, Grace R. Kim, Moritz A. Graule, and Robert J. Wood. An active palm enhances dexterity of soft robotic in-hand manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11790–11796, 2021. doi: 10.1109/ICRA48506.2021.9562049. 2.2.2

[59] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization, 2018. URL https://arxiv.org/abs/1805.11074. 2.2.3

[60] Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robot. and Automat. Letters*, 6(3):4915–4922, 2021. 2.2.3, 6.4.1

[61] Garrett Thomas, Yuping Luo, and Tengyu Ma. Safe reinforcement learning by imagining the near future, 2022. URL https://arxiv.org/abs/2202.07789. 2.2.3, 2.2.3, 6.2.1

[62] Thomas George Thuruthel, Benjamin Shih, Cecilia Laschi, and Michael Thomas Tolley. Soft robot perception using embedded soft sensors and recurrent neural networks. *Science Robotics*, 4(26):eaav1488, 2019. 2.2.1

[63] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. 2.2.2

[64] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 4.2.4, 4.2.4

[65] Tao Yang, Youhua Xiao, Zhen Zhang, Yiming Liang, Guorui Li, Mingqi Zhang, Shijian Li, Tuck-Whye Wong, Yong Wang, Tiefeng Li, et al. A soft artificial muscle driven robot with reinforcement learning. *Scientific reports*, 8(1):14518,

2018. 2.2.2

[66] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J. Ramadge. Projection-based constrained policy optimization. *CoRR*, abs/2010.03152, 2020. URL https://arxiv.org/abs/2010.03152. 2.2.3

[67] Moritz A. Zanger, Karam Daaboul, and J. Marius Zöllner. Safe continuous control with constrained model-based policy optimization, 2021. URL https://arxiv.org/abs/2104.06922. 2.2.3

[68] X Zhang, FK Chan, T Parthasarathy, and M Gazzola. Modeling and simulation of complex dynamic musculoskeletal architectures. *Nature Communications*, 10 (1):1–12, 2019. doi: 10.1038/s41467-019-12759-5. URL https://doi.org/10.1038/s41467-019-12759-5. 2.1.1