Explicit Enforcement of Desirable Action Space Properties in Deep Learning Models for Robotic Control

by

Michael Przystupa

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Faculty of Science Department of Computing Science
University of Alberta

Abstract

This dissertation investigates methods for incorporating robotic domain knowledge into deep learning systems to represent different action spaces. Robots can be controlled through action spaces spanning different levels of abstraction, from low-level joint torques to high-level Cartesian poses. Beyond physical action spaces, latent action spaces offer learned embeddings that coordinate atomic commands into coherent patterns—motivated by the manifold hypothesis that robot actions often lie on lower-dimensional manifolds. A key limitation of existing latent action space learning is that current models often ignore available prior knowledge, requiring substantially more data than approaches that explicitly encode motion structure. Our hypothesis is that encoding robotic priors into deep learning architectures improves performance for both autonomous agents and human operators interacting with the system.

We present three primary contributions toward this goal. First, we develop local-linear neural networks that explicitly encode human priors in teleoperation systems. This architecture has nonlinear transformations of robot-state variables but behaves linearly with respect to user inputs, enforcing scalability, consistency, reversibility, and controllability. Our user studies demonstrate significant improvements over alternative deep learning systems and provide a more interpretable framework for explaining human success rates. Second, we incorporate Bayesian aggregation with neural networks to develop more robust deep learning movement primitives that perform well with less data on real robot systems while offering greater flexibility in specifying motor-primitive skills. Third, we improve the transferability of cross-embodiment control policies that leverage a robot's topological structure, showing that transfer

often requires specialization to the target embodiment and can be adjusted to fit available compute resources.

Our work develops neural architectures with robot domain knowledge spanning spatial relations, temporal interactions, and robot topological structure. These contributions highlight the significant role domain knowledge continues to play in developing robot learning systems that generalize well to changing situations. This work represents a step toward better modular deep learning components that are interpretable, data-efficient, and reusable across different robotic platforms and tasks.

Preface

This thesis is based on the following previous publications:

- Chapter 3 and Chapter 4 are a combination of three previously submitted papers on the topic of action maps for teleoperation.
  - Przystupa, Michael; Johnstonbaugh, Kerrick; Zhang, Zichen; Petrich, Laura; Dehghan, Masood; Haghverd, Faezeh; Jagersand, Martin "Learning State Conditioned Linear Mappings for Low-Dimensional Control of Robotic Manipulators." IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023.
  - Przystupa, Michael; Gidel, Gauthier; Taylor, Matthew E.; Jagersand, Martin; Piater, Justus; Tosatto, Samuele "Local Linearity is All You Need (in Data-Driven Teleoperation)." 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2024.
  - Przystupa, Michael; Wang, Allie; Knote, Lino; Parker, Adam; Hollenstein, Jakob; Taylor, Matthew E.; Jagersand, Martin; Piater, Justus; Tosatto, Samuele "Investigating the Efficacy of Mode-Switching with Deep Learning Action Maps for Teleoperation", Accepted to Human Robotic Interactions Late-Breaking Results, 2026 but withdrawn.

My contributions in these chapters include:

1. Developing the deep learning architectures used studied in the across the different projects for teleoperation.

2. Deriving the theoretical results to verify the proposed systems enforce necessary teleoperation behaviors.

3. Leading the organization and execution of user study experiments across all projects in collaboration with coauthors.

Research for these research projects, of which this thesis is a part, received research ethics approval from the University of Alberta Research Ethics Board, Communication mechanisms for cooperative human-robot manipulation tasks in unstructured environments, ID Pro00054665, March 30, 2016.

- Chapter 5 is an edited version of our accepted paper: Przystupa, Michael; Haghverd, Faezeh; Jagersand, Martin; Tosatto, Samuele "Deep probabilistic movement primitives with a bayesian aggregator." 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2023.

- Chapter 6 is based on our previously published paper: Przystupa, Michael; Tang, Hongyao; Jagersand, Martin; Miret, Santiago; Phielipp, Mariano; Taylor, Matthew E.; Berseth, Glen "Efficient Morphology-Aware Policy Transfer to New Embodiments". Reinforcement Learning Journal 6. (2025): 1521–1539. The chapter modifies introduction to connect the material to the whole thesis.

"Sometimes you climb out of bed in the morning and you think, I'm not going to make it, but you laugh inside — remembering all the times you've felt that way."

-Charles Bukowski

To my aunt, Lydia Corleto, who passed away before I finished this dissertation.

Thank you for always believing in me, and I hope you are at peace now.

Acknowledgements

I thank my supervisors, Martin Jagersand, for his support from the beginning of my PhD. His numerical methods perspective on robotics helped me think critically on more effectively using machine learning to solve robotic problems. He was supportive throughout the process, and I am grateful for the opportunity to contribute to robotics research because of my involvement in his lab group.

I also thank my co-supervisor, Matthew E. Taylor, for his support from the beginning of my 3rd year of PhD. He provided a critical outlook on discussing research ideas and has helped me grow in my academic writing. I'm glad to have been a member of your group as well.

I thank my committee members, Dr. Samuele Tosatto and Dr. Glen Berseth. My collaborations with them helped expand my network to international collaborations. These experiences directly led to several chapters in this dissertation, and I am grateful for all I've learned from both of you.

There have been numerous co-authors throughout this process, whom I am grateful to. Thank you, Laura Petrich, for teaching me so much about assistive robotics. Thank you to Kerrick Johnstanbough, Allie Wang, and Lino Knote for your assistance with user study experiments. Thank you to Vincent Zhang and Gauthier Gidel for their help in developing the theory featured in this thesis. Thank you to Masood Deghan, who taught me a lot about control theory and introduced me to teleoperation.

To my partner, Sara Khademioureh, thank you for being with me throughout this degree. I love you to the moon and back, and I am grateful you've been in my life. Thanks for pushing me on the road to completing my dissertation, and I'm glad we

pursued our doctoral studies together.

I thank my family for their love throughout my degree. To my parents, Alicia Corleto and Pawel Przystupa, it was always great seeing you in the many places I visited throughout my education. To my sister, Paulina Przystupa, thanks for all the edits and critical outlook on writing over the years.

Table of Contents

List of Tables

List of Figures

List of Symbols

G	Discounted cumulative return
T	Number of discrete time steps in an episode
$\mathcal{C}$	Context space
$\mathcal{M}$	Set of robot morphologies
$\mathbf{\Sigma}$	Covariance matrix
$\mathcal{S}$	State space
$\mathcal{X}$	Robot configuration space
$\mathcal{Z}$	Latent space
$\mathbf{a}$	Action vector
$\bar{\mathbf{c}}$	Target morphology context vector
$\mathbf{c}$	Context variable vector
$\ddot{\mathbf{y}}$	Second time derivative of output trajectory (acceleration)
$\dot{\mathcal{X}}$	Robot velocity space
$\dot{\mathbf{y}}$	Time derivative of output trajectory
$\dot{\mathbf{x}}$	Time derivative of robot state (velocity)
$\boldsymbol{\epsilon}$	Noise or perturbation vector
$\mathbf{f}$	Dynamics function vector output

γ	Discount factor ($\gamma \in [0, 1)$)
$\mathbf{g}$	Goal vector
$\hat{A}$	Estimated advantage function
$\mathbf{h}$	Hidden state or embedding vector
$\boldsymbol{\mu}$	Mean vector of a probability distribution
$\mathbf{m}$	Morphology descriptor vector
$\boldsymbol{\omega}$	Blending coefficients for trajectory interpolation
$\mathbf{o}$	Observation vector
$\boldsymbol{\phi}$	Temporal basis feature matrix ($\Phi \in \mathbb{R}^{T \times h}$)
$\pi_\theta(s)$	Control policy parameterized by θ
$\boldsymbol{\sigma}$	Standard deviation vector of a probability distribution
$\mathbf{s}$	State input vector
$\boldsymbol{\tau}$	Trajectory or discretization step
θ	Learnable parameters of a neural network or policy
$\mathbf{v}$	Velocity vector
$\mathbf{w}$	Weight vector
$\mathbf{x}$	Robot state vector
$\mathbf{y}$	Output trajectory vector
$\mathbf{z}$	Latent variable vector

Abbreviations

AE Autoencoder.

CAE Conditional Autoencoder.

CMDP Contextual Markov Decision Process.

CNMP Conditional Neural Movement Primitive.

CNN Convolutional Neural Network.

DeepProMP Deep Probabilistic Movement Primitive.

DMP Dynamic Movement Primitive.

DOF Degrees of Freedom.

DPC Deep Principal Component Action Map.

ELBO Evidence Lower Bound.

KL Kullback-Leibler.

LfD Learning from Demonstration.

LN Layer Normalization.

LoRA Low-Rank Adaptation.

MDP Markov Decision Process.

MLP Multi-layer Perceptron.

MSE Mean Squared Error.

NASA-TLX NASA Task Load Index.

NOAH Nonlinear Odd Action Hypernetwork.

PCA Principal Component Analysis.

PEFT Parameter Efficient Fine-Tuning.

PPO Proximal Policy Optimization.

ProMP Probabilistic Movement Primitive.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

SCL State Conditioned Linear Map.

SGD Stochastic Gradient Descent.

STLC Small-Time Local Controllability.

TRPO Trust Region Policy Optimization.

VAE Variational Autoencoder.

VAE-CNMP Variational Autoencoder Conditional Neural Movement Primitive.

VLM Vision-Language Model.

Glossary of Terms

Action Map A function that maps inputs to system action commands.

Assistive Robotics A field of robotics concerned with developing robotic systems that support people with physical disabilities in performing daily tasks.

Autoencoder A neural network that reconstructs data from compressed representations.

Bayesian Aggregation A method for combining multiple observations into a single posterior estimate by treating each observation as a likelihood update.

Contextual Policy A policy conditioned on a context variable that characterizes task or environment variations.

Cross-embodiment Transfer The problem of transferring a learned policy from one robot embodiment to another with a different morphology or action space.

Degrees of Freedom The number of independent parameters required to describe the configuration of a system.

Demonstration A recording of a human performing a task, used as training data in learning from demonstration.

Embodiment The physical instantiation of a robot, including its morphology, actuator configuration, and sensor suite.

Evidence Lower Bound A tractable lower bound on the log-likelihood of observed data, used as the optimization objective in variational inference.

Fine-tuning The process of adapting a pre-trained model to a new task or domain by continuing training on task-specific data.

Hypernetwork A neural network that generates the weights of another neural network as a function of some input.

KL Divergence A measure of how one probability distribution differs from a reference distribution.

Latent Action A synonym for an action map used in machine learning research.

Layer Normalization A technique that normalizes activations across the feature dimension of a single training example to stabilize neural network training.

Learning from Demonstration A technique that enables robots to acquire skills by observing and imitating human demonstrations rather than through reward-based trial and error.

Low-rank Adaptation A parameter-efficient fine-tuning technique that approximates weight updates as the product of two low-dimensional matrices.

Markov Decision Process A mathematical formulation for discrete-time decision making problems to maximize a reward function.

Movement Primitive A parametric model that represents temporal motion patterns in continuous time.

Multi-layer Perceptron A neural network that stacks linear transformations with intermediate nonlinear operations in between.

NASA Task Load Index A multi-dimensional subjective workload assessment instrument measuring mental, physical, and temporal demands of a task.

Over-actuated System A system where the degrees of control are greater than the degrees of freedom.

Policy A function that predicts actions to interact with the world.

Prefix Tuning A parameter-efficient fine-tuning technique that prepends trainable token embeddings to the input of a pre-trained model.

Principal Component A direction of maximal variance in a dataset, computed as an eigenvector of the data covariance matrix.

Reinforcement Learning A research topic in machine learning that studies and develops algorithms for maximizing reward in sequential decision making problems.

Self-attention A function that returns the weighted sum of inputs based on their dot-product interaction with a desired query value.

Small-time Local Controllability A property of a dynamical system guaranteeing that any state near the current state is reachable within an arbitrarily small time.

Supervised Learning A machine learning research topic that studies and develops algorithms to learn function approximations given a set of input and output pairs of desired function.

Teleoperation The process of operating some system — such as a robot — through a remote control interface.

Transformer A class of neural networks that capture interactions between sequence data with a self-attention mechanism.

Variational Autoencoder An autoencoder that learns a structured probabilistic latent space by regularizing the encoder output toward a prior distribution.

Vision-Language Model A neural network trained jointly on visual and textual data to support tasks requiring grounded language understanding.

Chapter 1

Introduction

Generalist robot policies can perform a variety of tasks given only a goal image or text description of the target task [12, 103]. These capabilities emerge from data scaling laws, in which model performance improves proportionally with training dataset size [83]. The objective of training a monolithic deep learning model on large real-world robotic datasets is to mitigate generalization to novel scenarios [71, 105]. These deep learning models enable robots to operate in unstructured environments; however, they have two fundamental limitations: their generalization capability rely soley training data coverage, and their decision-making processes are difficult to interpret.

Large robotic datasets are difficult to collect because they require robots to perform the tasks in the real world as opposed to simulation. Existing large-scale datasets often aggregate data from multiple sources and require many days to manually collect and validate [71, 105]. Researchers have developed robot-agnostic interfaces to streamline this process, but these systems either require substantial real-world hardware [26] or assume specific robot morphological structure (e.g., serial kinematic chains) to map the collection data to the desired robot [19].

Autonomous data collection procedures could be a solution [94], but these too have limitations. Recent work shows that autonomous collection provides only modest improvements — up to 10% in task success rates — compared to collecting additional human demonstrations [96]. While human-in-the-loop and online learning approaches

offer greater adaptability by combining autonomous exploration with targeted human feedback [165], they face challenges in determining when and how to solicit interventions. Overall, these limitations suggest that simply scaling data collection is insufficient for effectively deploying robots in unstructured environments.

Regardless of whether data collection processes are improved for robotic applications, deep learning remains a largely black-box modeling approach. Neural networks perform stacked nonlinear transformations of data, making it difficult to identify the contributions from individual input values [37]. Even shallow neural architectures are difficult to understand despite simpler architectures [16]. Real-world deep learning models are substantially more complex [12], further exacerbating understanding model behavior.

These difficulties with interpreting model behavior are problematic in domains where safety guarantees are important, such as manufacturing or assistive care. Robot deployment in manufacturing is increasing globally, and reducing human-robot collision injuries has become paramount [39, 92]. Likewise, in assistive care, robots need to be robust handling delicate, life-saving tasks such as feeding [2]. These applications of robotics require accessible mechanisms to both prevent failures and troubleshoot issues seamlessly [64].

Furthermore, current deep learning models process images to perform tasks, which are computationally intensive due to the high-input dimensionality. Modern vision-based systems must not only perform tasks accurately but also detect humans in the scene and respond rapidly to their presence [126]. Rapid response remains challenging for monolithic neural networks, which require significant computational resources and are difficult to analyze when failures occur.

An alternative approach to end-to-end learning is employing modularized learning, where the robotic control system comprises specialized components organized hierarchically to perform tasks. Classical robotic learning used modularized systems to perform complex tasks with orders of magnitude fewer trajectories [36, 130]. Hierar-

chical decomposition has proven valuable in domains such as reinforcement learning, where explicit model decomposition improves sample efficiency [150].

A critical aspect of learnable modules is appropriately handling diverse action spaces. Even existing generalist policies require modular components when dealing with varying embodiments [103]. In these cases, researchers employ specialized neural networks for specific robots to adapt learned features to the target platform [139]. Different action space representations pose significant challenges for generalist policies, requiring additional considerations in model design [41]. Research on improving action abstraction modules is necessary to improve modularized learning systems for real-world deployment.

1.1 Overview

This dissertation investigates methods for incorporating robotic domain knowledge into deep learning systems to represent different action spaces. Robots are well-structured systems in which substantial prior knowledge exists that can be leveraged to simplify the learning process. Typical deep learning models are often black box models, but this research demonstrates that desirable domain-specific priors can be systematically encoded into neural network architectures and training procedures. Our approach focuses on developing deep learning models that serve specific roles and are usable in larger learning systems as importable modules. For example, in a teleoperation system, a high-level policy network can generate abstract motion commands that are then decoded by our learned action representation module into robot-specific joint trajectories.

Robots can be controlled through various action spaces spanning different levels of abstraction. These range from low-level commands such as individual actuator torques to high-level abstractions like Cartesian end-effector poses — the position and orientation of a robot's tool in the robot arm's 3D workspace. Higher-level action spaces assume access to dynamic and kinematic models of the target control

system. Vision-based robot control systems have been successful because of the availability of these high-level action-space representations. The limitation of high-level action spaces is requiring well-structured models, which are more difficult to acquire for reconfigurable or soft robotic systems. A policy that works well in low-level action spaces — such as configuration space (i.e., joint space) — can have better generalization capabilities by avoiding the need for available dynamic or kinematic models of the system in these cases.

Beyond physical action spaces, it is possible to learn *latent action spaces* that map to the robot's designated physical action space. Latent action spaces represent a learned embedding in which meaningful collections of robot commands are coordinated into coherent patterns. This approach is motivated by the manifold hypothesis [163], which posits that high-dimensional data often lie on lower-dimensional manifolds. Our claim is that robot action spaces follow the manifold hypothesis, meaning that atomic action commands can be represented in a lower-dimensional latent space. For instance, wiping a table is fundamentally a 2-dimensional task in Cartesian space (moving across the table surface), but a 7-DOF robot arm controls this motion through a 7-dimensional joint configuration space. A learned latent action space can capture the essential 2D motion pattern while automatically coordinating the redundant joint movements. Similarly, grasping motions across different objects share common patterns — approach, pre-grasp shaping, and closure — that can be encoded as coordinates in a compact latent space rather than independent commands for each joint at each timestep.

A key limitation of existing latent action space learning in robotics is that current models often ignore available prior knowledge about the target problem. This is problematic because incorporating appropriate inductive biases can dramatically reduce the data required to learn robust latent action spaces for control tasks. Dynamic movement primitives exemplify this principle: by modeling motion as a dynamical system, they can produce reusable movements that generalize to new goal conditions

from as few as a single demonstration [61, 62]. In contrast, deep learning models can struggle to capture motion structure reliably, even when trained on large datasets, which is problematic for a robot's ability to solve tasks in uncontrolled environments.

This research investigates multiple approaches for incorporating robotic domain knowledge to adapt deep learning systems to varying action space requirements. The general approaches we consider include explicitly encoding mathematical properties in the deep learning model or explicitly accounting for robot knowledge in the learning system. The central hypothesis is that encoding robotic priors into deep learning architectures improves the performance of both robotic agents and human operators interacting with the system. This work represents a step toward developing modular deep learning components that are interpretable, data-efficient, and readily reusable across different robotic platforms and tasks.

1.2 Thesis Outline

This dissertation examines three domains for imbuing robotic knowledge into deep learning models across different action abstractions: teleoperation, movement primitives, and cross-embodiment transfer. Each of these three domains represents a distinct challenge in action space design: user-centric control (tele-operation), motion representation (movement primitives), and cross-platform generalization (morphology-aware learning). Together, these approaches demonstrate that incorporating structured robotic knowledge into deep learning systems can achieve both improved model interpretability and generalization compared to purely data-driven approaches.

Deep Learning Teleoperation Systems The first domain is teleoperation, in which machine learning models directly map user inputs to atomic robotic commands. Robotic teleoperation systems have critical real-world applications in assistive robotics, where users with limited mobility require better interfaces to control robots effectively [116]. We demonstrate that encoding user-specific control characteristics

into deep learning models improves user experience and provides mathematical guarantees that the teleoperation interface achieves predictable, intended behaviors. This chapter has the following contributions:

1. Formalizes teleoperation control as a dynamical system and provides refined definitions of teleoperation properties using this framework. A key distinction of our definitions is that they have clear, interpretable connections to the underlying dynamical system and can be extended to define more complex teleoperation settings with appropriate additional assumptions.

2. Proposes several neural network architectures and loss function modifications that cause a deep learning system to behave as an odd function by design. We demonstrate that this property is a significant component in improving user experience when using neural network-based teleoperation interfaces.

3. Proves that several loss functions and architecture design choices are mathematically guaranteed to enforce our teleoperation properties. These mathematical guarantees differ from prior research, which focused on empirical evidence that these properties are enforced by neural networks.

Teleoperation User Study Experiments We validate our findings by developing teleoperation systems with a series of user study experiments. These experiments are treated separately because they show the practical uses of our teleoperation system as opposed to the theoretical aspects of our work. These user studies help demonstrate the applications of our system and the limitations of the deep learning teleoperation approach. This chapter has the following contributions:

1. Demonstrates that our systems outperform previously applied deep learning models for teleoperation settings. Users perform better both subjectively and quantitatively with our systems compared to alternative deep learning models.

2. Contains user studies that reveal nuanced results on the benefits of non-linear teleoperation, where task difficulty is a significant factor. For simple tasks, linear methods such as PCA can be quite effective.

3. Compares learning-based teleoperation maps to standard Cartesian mode-switching, which is the typical de facto approach in teleoperation control. The results suggest that learned teleoperation maps often perform comparably to mode-switching, contradicting prior claims about the benefits of learnable interfaces. However, for sufficiently complex tasks, our systems provide notable advantages.

Deep Probabilistic Movement Primitives The second domain is movement primitives, which represent temporal motions with parametric models. Their key advantage is employing structured mathematical models that enable operations such as temporal modulation — controlling the duration of movements; motion blending — mixing multiple primitives together; and context conditioning — specifying conditions on the primitives — to generate new motions from the trained primitive models. This research demonstrates that deep learning-based movement primitives can preserve these mathematical operations while leveraging deep learning expressiveness. This capability allows our model to synthesize novel movements beyond those directly observed during training. This chapter has the following contributions:

- Proposes a deep learning model that performs the same mathematical operations as the classical probabilistic movement primitives framework. Our approach uses Bayesian aggregation in a latent action embedding space to blend temporal action skills.

- Contains simulation experiments that demonstrate that the proposed framework is capable of handling several complex tasks in simulation and generalizes more effectively compared to other deep learning movement primitive frameworks.

- Contains real-robot experiments that are conducted to demonstrate our method can be directly deployed on physical hardware while more effectively using limited training samples compared to alternative deep movement primitive frameworks.

Morphology-Aware Transfer Learning The final domain is morphology-aware policy learning, in which I investigate transfer learning approaches for adapting policies across robot embodiments with different kinematic structures and action dimensions. Morphology-aware learning enables deep learning policies to control robots despite variations in morphology — such as different numbers of joints, link lengths, or end-effector configurations. Critically, this research quantifies the extent to which pre-trained policies can generalize across changes in action space dimensionality and robot morphology with different fine-tuning methods. This chapter has the following contributions:

- Includes the first investigation of parameter-efficient fine-tuning methods in the context of morphology-aware policies. A distinction of this research from prior transfer learning studies is its focus on online policy learning, which introduces additional challenges beyond supervised learning applications of fine-tuning techniques.

- Includes a large-scale empirical study to compare direct fine-tuning, input adapter techniques, and prefix-tuning across 18 agent-environment combinations using a previously proposed morphology-aware pre-training framework. These experiments reveal several trends in comparative performance for scenarios in which one can either directly fine-tune model components or only modify model inputs.

- Includes extensive ablation studies on fine-tuning techniques largely applied in vision and natural language domains, including low-rank adapter layers and

prefix-tuning. These experiments reveal — particularly for prefix-tuning — that hyperparameter choices substantially affect both early and late stage performance of fine-tuning.

Together, these chapters demonstrate that the limitations of purely data-driven robotic policies — namely, their dependence on large datasets and their lack of interpretability — can be mitigated by systematically encoding domain knowledge into deep learning architectures. Our teleoperation work provides mathematical guarantees on network behavior, directly addressing the interpretability gap in neural network control. Our movement primitive framework preserves the structured operations of classical methods while leveraging deep learning expressiveness, reducing the data requirements for learning reusable motion representations. Our cross-embodiment work quantifies the extent to which structured pre-training enables policy transfer across different robot morphologies. These contributions collectively advance the development of modular, interpretable deep learning components for real-world robotic deployment.

Chapter 2

Background & Literature Review

This chapter provides background information relevant to the research, including reinforcement learning, feedforward neural networks, and conditional autoencoders. Later chapters include specialized background sections for topic-specific material. The chapter ends with a literature review of work related to each of the later chapters.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a problem setting that learns methods for optimizing control policies using only data from interacting with the system. Figure 2.1 visualizes the RL problem. An RL problem can be formalized as a Markov decision process (MDP) [149]. An agent takes actions $a \in \mathcal{A}$ that cause transitions between states $s \in \mathcal{S}$ in the environment. The agent starts with a distribution of states $s_0 \sim p(s_0)$ and selects actions based on a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, which maps states to a probability distribution $\mathcal{P}$ over actions. The transitions between states are represented as the conditional probability distribution $s' \sim p(s'|s, a)$ for the next state $s' \in \mathcal{S}$. The reward function, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, represents the immediate value of actions between sub-sequent states. In summary, an MDP is a tuple $M = (\mathcal{S}, \mathcal{A}, p(s'|s, a), r, p(s_0))$.

The goal of an MDP is to find policies that maximize the expected sum of rewards:

$$\pi^\star(s) = \arg\max_{\pi \in \Pi} \mathbb{E}_{p(\tau)}[G(\tau)].$$

The expected cumulative reward $G(\tau) = \mathbb{E}_{p(\tau)}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})]$, with discount

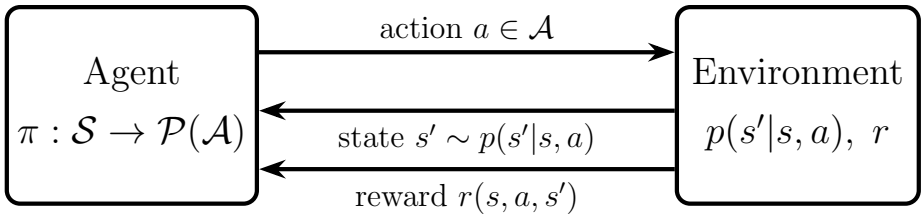


Figure 2.1: The reinforcement learning loop. An agent selects actions $a \in \mathcal{A}$ according to policy π based on the current state. The environment transitions to a new state $s' \sim p(s'|s, a)$ and returns a reward $r(s, a, s')$ to the agent.

factor $\gamma \in [0, 1)$, represents returns when following the trajectory distribution,

$$\tau \sim p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t).$$

Our work uses a finite horizon T because the robotics tasks we consider have finite durations, but the formulation is applicable to other settings.

2.1.1 The Atomic Action Space

There are many choices for action spaces $a \in \mathcal{A}$ in robotic control tasks, each of which can affect the performance of agents for the specified reward function [30, 111]. We define the *atomic action space* as the physical action space used to control the robot. The atomic action space is a continuous real space $\mathbf{a} \in \mathbb{R}^l$, where commands control each limb $l \in \mathbb{N}^+$ of the robot (e.g., the joint velocities). This is an assumption in many robot learning systems, particularly when working in Cartesian pose space, which are ultimately converted to joint space trajectories.

2.1.2 Value Functions

To efficiently find optimal policies, RL algorithms typically learn value functions that estimate the expected future rewards from different states and actions. One of the primary areas of research in RL is modeling and learning the Q-function,

$$Q^\pi(s, a) = \mathbb{E}[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k, s_{k+1}) \mid s_t = s, a_t = a],$$

which quantifies the *value* of a state-action pair for a task and corresponds to the optimal policy $\pi^\star(s) = \arg\max_{a \in \mathcal{A}}[Q(s,a)]$.

Another central RL quantity is the value function,

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(s)}[Q(s,a)],$$

which quantifies the value of a state. The value function can be derived from the Q-function as $V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}[Q^\pi(s,a)]$. The value function can be used to define the advantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, which measures the value of certain state-action pairs compared to the average expected returns in a given state.

Typically, the Q-function and value function are estimated with collected demonstration data from the environment and represented with a parametric model, $V_\theta(s)$ or $Q_\theta(s,a)$, with parameters $\theta \in \mathbb{R}^n$. These parameters are then optimized with collect transition tuples from the environment:

$$\arg\max_\theta ||\gamma \hat{V}(s') + r(s) - V_\theta(s)||. \tag{2.1}$$

A bootstrap estimate $\hat{V}(s')$ is used and is often a moving average of the learned value function. Value function models are important for the cross-embodiment fine-tuning research conducted in Chapter 6. This chapter discusses the effects fine-tuning methods have on policy performance in the RL context.

2.2 Multi-layer Perceptrons

The Multi-layer perceptron (MLP) is a central class of neural network architectures in the literature. An MLP is a universal function approximator capable of learning arbitrary continuous functions given sufficient width [24, 56]. Supervised learning applications of MLP assume available datasets $\mathcal{D} = \{(\mathbf{o}_i, \mathbf{a}_i) : i \in [0, N]\}$, which consists of $N \in \mathbb{N}^+$ observations $\mathbf{o}_i \in \mathbb{R}^n$ and targets $\mathbf{a}_i \in \mathbb{R}^l$. It is assumed observations and targets are drawn independently and identically distributed from distribution

$(\mathbf{o}_i, \mathbf{a}_i) \sim p(\mathbf{a}_i|\mathbf{o}_i)p(\mathbf{o}_i)$. In RL problems, this assumption is typically violated because data is generated sequentially over trajectories. The dataset is used to learn a function $\pi : \mathbb{R}^n \to \mathbb{R}^l$ that minimizes against some loss $L : \mathbb{R}^n \to \mathbb{R}$,

$$\min_{\pi \in \Pi} \mathbb{E}[L(\pi(\mathbf{o}), \mathbf{a})].$$

The choice of function class Π is typically user-specified, where specifying an MLP is just one of many choices. MLPs are particularly well-suited for learning complex nonlinear mappings from vector-valued inputs. However, they lack the inductive biases of specialized architectures such as convolutional neural networks (CNNs) for spatial data or recurrent neural networks (RNNs) for sequential data.

2.2.1 MLP Architecture

An MLP is a feedforward neural network that operates on vector inputs; Figure 2.2 shows a diagram of a typical model. A single layer in an MLP is a function $h : \mathbb{R}^{h_{l-1}} \to \mathbb{R}^{h_l}$,

$$h(\mathbf{x}) = \sigma(W\mathbf{x} + b),$$

where we denote layer $l \in \mathbb{N}^+$ with hidden layer dimensions $h_{l-1}, h_l \in \mathbb{N}^+$. Here, $h_0 = n$ corresponds to the input dimension. The term h_l often refers to the number of neurons in layer l. The learnable parameters $W \in \mathbb{R}^{h_l \times h_{l-1}}$ and $b \in \mathbb{R}^{h_l}$ define an affine transformation of the inputs. The activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear operation applied element-wise to each component of the vector $Wx + b$. Typical choices for σ include the sigmoid function $\sigma(z) = 1/(1 + e^{-z})$, the hyperbolic tangent $\sigma(z) = \tanh(z)$, or the Rectified Linear Unit (ReLU) $\sigma(z) = \max(0, z)$. Tanh activation functions are especially important to research discussed in Chapter 3 because they are odd functions $-f(\mathbf{a}) = f(-\mathbf{a})$.

The hidden layers of an MLP can be stacked on top of each other to learn increasingly complex representations,

$$\pi(\mathbf{o}) = W^L h^{L-1}(\cdots h^2(h^1(\mathbf{o}))\cdots) + b^L,$$

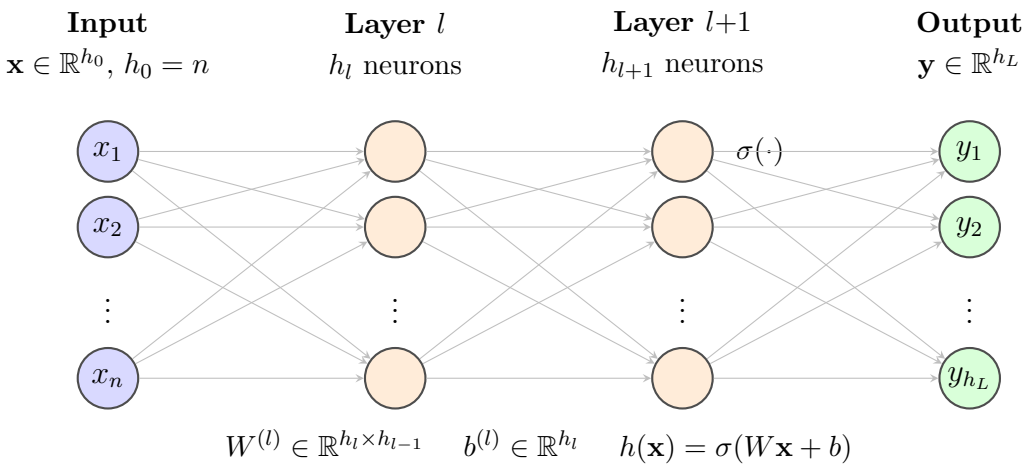


Figure 2.2: A feedforward neural network (MLP) with input $\mathbf{x} \in \mathbb{R}^n$, two hidden layers, and output $\mathbf{y} \in \mathbb{R}^{h_L}$. Each input node corresponds to a single scalar component x_i of the input vector. At each hidden layer l, every neuron computes an affine transformation of its inputs followed by a nonlinear activation $\sigma(\cdot)$, i.e. $h(\mathbf{x}) = \sigma(W^{(l)}\mathbf{x}+b^{(l)})$, where $W^{(l)} \in \mathbb{R}^{h_l \times h_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{h_l}$ are learnable parameters. Edges between layers represent the full matrix multiplication, with each neuron receiving weighted inputs from all neurons in the preceding layer.

where we denote the output layer parameters as $W^L \in \mathbb{R}^{d \times h_{L-1}}$ and $b^L \in \mathbb{R}^d$ for a model with $L \in \mathbb{N}^+$ hidden layers. Note that the output layer typically does not include an activation function for regression problems.

2.2.2 Training and Optimization

Neural networks are trained by minimizing a loss function with respect to the network parameters $\theta = \{W^1, b^1, \ldots, W^L, b^L\}$. For regression tasks, a common choice is the mean squared error (MSE),

$$L(\pi, \theta, \mathbf{a}) = \frac{1}{n} \sum_{i=1}^{n} ||\mathbf{a}_i - \pi_\theta(\mathbf{o}_i)||_2^2.$$

The parameters are optimized using gradient-based methods. The standard gradient descent update rule is

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\pi, \theta, \mathbf{a}),$$

where $\alpha > 0$ is the learning rate. Backpropagation [128] is employed to efficiently compute the gradients $\nabla_\theta L$ by applying the chain rule recursively through the net-

work layers. This algorithm has computational complexity linear in the number of parameters, making it tractable for MLPs with many layers. In practice, stochastic gradient descent (SGD) or mini-batch gradient descent is typically used, where gradients are computed on small subsets of the data rather than the entire dataset. Modern optimizers such as Adam [73] or RMSprop [1] incorporate adaptive learning rates and momentum to improve convergence. We use the Adam optimizer in this dissertation because it require minimal hyperparameter tuning and has fast convergence rates.

Although MLPs have shown strong empirical performance, several practical considerations need to be accounted for during training. The initial values of W and b significantly impact training dynamics. In this dissertation, neural network weights are sampled from a uniform distribution $\theta_i \sim U(-\sqrt{1/h_{l-1}}, \sqrt{1/h_{l-1}})$. Other common schemes include Xavier/Glorot initialization [35] and He initialization [49], which scale initial weights based on layer dimensions to maintain stable gradient magnitudes. To prevent overfitting, different types of regularization techniques can be employed, including L2 weight decay (adding $\lambda||\theta||^2$ to the loss), dropout [146] (randomly deactivating neurons during training), and early stopping based on validation set performance.

2.3 Conditional Autoencoders

One of the central learning frameworks employed in this dissertation is the conditional autoencoder (CAE) which is visualized in Figure 2.3. These models learn latent representations that can reconstruct the target data. The conditional input provides context for the latent codes, which might be ambiguous otherwise. This section describes a general formulation of the CAE framework relevant to the work in Chapters 3, 4, and 5.

The CAE model is optimized with a dataset of state-action tuples $D = \{(\mathbf{s}_i, \mathbf{a}_i) \mid i \in [1, N]\}$, for $N \in \mathbb{N}^+$ samples. Bold notation specifies real vectors $\mathbf{s} \in \mathbb{R}^d$ and $\mathbf{a} \in \mathbb{R}^l$ for $d, l \in \mathbb{N}^+$ dimensions. The dataset is assumed to be provided by some

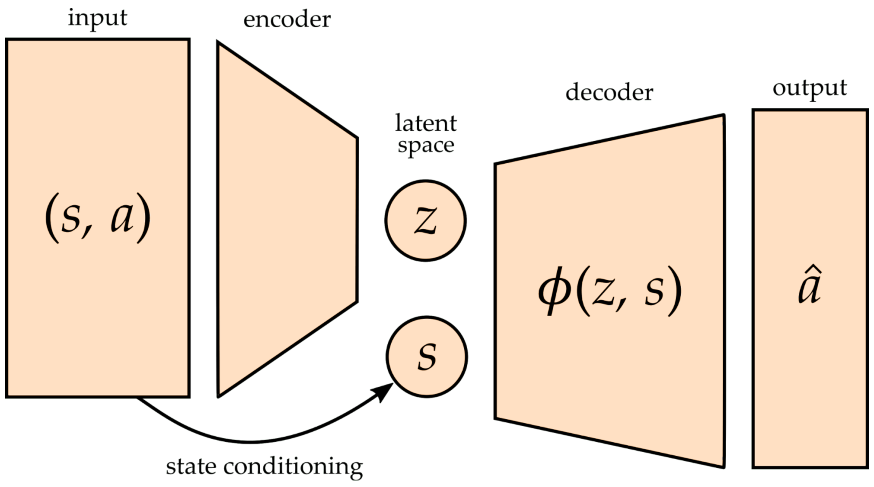


Figure 2.3: A conditional autoencoder architecture, the state and action are used to encode a latent space that reconstructs the original actions. Reproduced with permission from Losey et al. [87].

optimal policy $\mathbf{a}_t \sim \pi^\star(\mathbf{s}_t)$ for the desired robotic task. The CAE model learns to re-produce the actions given the state, but discrete time steps $t \in \mathbb{N}^+$ are not encoded in the model.

A CAE comprises two neural networks: an encoder that learns to compress the data and a decoder that reconstructs the data. The *encoder* $g_\theta : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$ learns to predict compressed representations $\mathbf{z} \in \mathbb{R}^m$ based on the current state and action, where $m \in \mathbb{N}^+$ and $m < l$. In certain cases, g_θ may generate distribution parameters $g_\theta(\mathbf{s}, \mathbf{a}) = \{\mu, \sigma\}$, which represent the mean and standard deviation of a diagonal Gaussian distribution, respectively. The decoder $f_\theta : \mathcal{S} \times \mathcal{Z} \to \mathcal{A}$ then reconstructs the original action based on the state information. The general loss function to optimize a CAE model is

$$L(\mathbf{s}, \mathbf{a}, f_\theta, g_\theta) = L^{recon}(\mathbf{a}, f_\theta(\mathbf{s}, g_\theta(\mathbf{s}, \mathbf{a}))) + L^{reg}(f_\theta, g_\theta, \mathbf{s}, \mathbf{a}).$$

The reconstruction term is the negative log-probability of a distribution of a Gaussian variable $L^{recon} = -\log N(\mathbf{a} \mid f_\theta(\mathbf{s}, g_\theta(\mathbf{a}, \mathbf{s})), I\sigma_a)$, with $\sigma_a \in \mathbb{R}^+$ is a hyperparameter and I is the identity matrix. The regularizer term L^{reg} can be the KL divergence $D_{KL}(P\|Q) = \int_{\mathcal{X}} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$, where $Q = N(\mathbf{z}; \mu, \sigma)$ and the prior distribution is $P = N(0, I)$. The KL divergence helps constrain latent codes to the support of

the prior distribution when sampling $\mathbf{z} \sim N(0, I)$ and encourages more semantically meaningful latent dimensions [53].

2.4 Literature Review

This section is a literature review of related works to each of the domains studied in this dissertation. The topics are organized into three domains — action maps (Chapter 3 and Chapter 4), temporal abstraction (Chapter 5), and morphology-aware learning (Chapter 6) — each of which connects to a component of our work. Action map research is the general area of research our work on teleoperation is most applicable, where intermediate models learn to map onto the robot's action space. Temporal abstraction is the general field of learning action sequences in which movement primitives are one class of approaches. Morphology-aware learning refers to methods adapting over changing action spaces.

2.4.1 Action Maps

Action maps are models that project agent actions to desired control system commands. The key distinction of action maps is they generate commands in the atomic action space at each step of control, as opposed to generating sequences of actions. Terminology varies between fields on the topic — over-actuated systems in control theory, or latent actions in machine learning and teleoperation. The underlying objective across fields is to define constraints on the action map for the target control problem. We distinguish these concerns of each of these domains and visualize a taxonomy of related works in Figure 2.4.

Over-actuated systems have more available actions than degrees of freedom in the target system; for example, a robotic manipulator with many actuated joints executing a low-dimensional Cartesian task admits multiple joint-space realizations of the same end-effector motion. Over-actuation can be beneficial to provide better fault tolerance, reduce cost, or perform varying control motions with the same actuators

[65, 135]. These redundancies arise in biological systems [27, 153], such as musculoskeletal systems [11, 14]. Identifying the optimal control inputs is known as the control allocation problem, where optimization is used to select appropriate actuator commands that realize a desired system-level action [48, 65, 76, 158, 159].

In machine learning, action maps for solving control allocation problems — identifying control inputs that realize a desired output action — are referred to as latent actions, and do not necessarily assume over-actuation in the target system. Robotic hand synergies can be controlled in joint space, but research has proposed methods to compress the control space into low-dimension manifolds [51, 129]. For example, planar wiping tasks in Cartesian space are two-dimensional problems, but may use the six to seven degrees of freedom (DOF) of a typical robot arm's actuated joints. Reducing the agent's number of actions is beneficial, as it can improve reinforcement learning sample efficiency [4, 17]. Latent action models can have additional constraints encoded in them, which can be helpful for control policies optimized with offline RL [3, 173].

Furthermore, latent action models are beneficial for robotic teleoperation, to project limited DOF input devices (e.g., joystick) to robot control axes (i.e., Cartesian pose). Early works investigated using a linear function to project user actions [21, 104]. The linear function has been explored for myoelectric interfaces to control poses of robotic hands [6, 93]. The limitation of linear functions is assuming all robot commands exist in a fixed linear sub-space reducing the relevant actions that the model can map.

Recent research suggest that conditional autoencoders (CAE) can improve the ability to accurately recover user actions. A CAE model uses robot state information to compensate for the limited available axis of control [68, 69, 87]. Studies also demonstrate that CAE models can be supported by action suggestion systems to improve user control accuracy [67, 88].

A significant challenge for latent action models in teleoperation is adapting to user expectations of the control system. A known issue of CAE action maps is user

confusion when operated initially [88]. This confusion stems from directly mapping user commands to task motions. A solution is querying users for expected behavior and learning a user preference model [81], but this requires more data collection from the user. One of our goals in this dissertation is to improve the CAE model design to improve user experience without additional data collection.

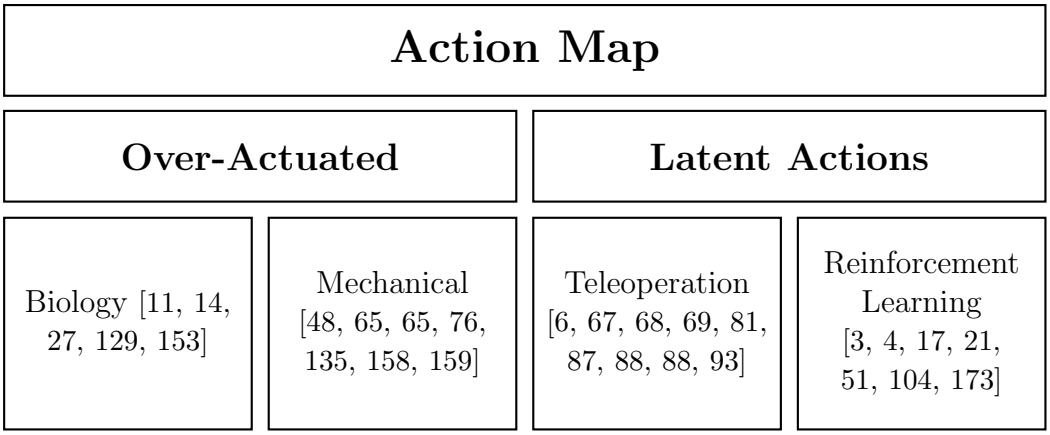


Figure 2.4: A taxonomy showing the applications of action map models in the literature. Research can broadly be separated into studying over-actuated systems in biology or mechanical systems, or otherwise as latent action models applied in teleoperation and reinforcemetn learning.

2.4.2 Temporal Action Abstractions

Temporal action abstractions are functions that generate *sequences of atomic actions* that are executed by the robotic system. Temporal models are useful in developing hierarchical control systems, where higher-level controllers make fewer decisions during a control task that are executed by lower-level control modules. The topic has been studied under different terminologies including *options* in reinforcement learning; *action chunks* in behavior cloning; and *movement primitives* in robotics. Figure 2.5 shows a taxonomy of different methods.

The options framework from reinforcement learning treat temporal abstraction as a semi-Markov decision process. At each decision point, a meta-policy selects high-level policies call options that executed low-level policies of actions which operates

autonomously until the next decision point in the system [150]. Prior research has focused mostly on discovering re-usable options to expedite the learning process of a system [147]. Option discovery methods include building hierarchical learning algorithms [8, 152, 170] and specifying sub-goal reward signals to train option policies [90, 91, 123]. Theoretical results suggest the option frameworks can improve sample learning efficiency [13, 58, 125]. The key distinction is that options are found *online* during the learning process of the control system.

In contrast, learning from demonstration (LfD) is a technique that enables robots to acquire complex, adaptive skills by observing and reproducing demonstrated trajectories [5, 124, 132, 174]. Recent works in video-language-action models favor *action chunk* predictions, where fixed sequential actions are predicted by the policy [70, 109, 113, 114, 162]. The choice in deploying action chunk models is often task-specific, where different action horizons can affect performance [20]. A limitation of these approaches is that the action chunk is fixed in duration and is not designed to easily synthesize between chunks.

A more flexible LfD learning framework users movement primitives (MPs), which model temporal actions as parametric models. Parameterized motions capture the essential features to allow for variations and modifications according to different situations. These models are useful to analyze movement patterns, such as in human beings [85, 168]. MPs can also be combined with RL to solve complex robot tasks [10, 45, 97, 112, 154]. Researchers have demonstrated MPs' utility across robotic embodiments, including manipulators [97], unmanned aerial vehicles [148], aquatic robotics [45], and quadrupeds [144, 172].

Researchers have developed a variety of frameworks to model movement primitives because of their general applications in robotics. One of the first MP frameworks are the dynamic movement primitives (DMP) approaches[62, 133]. This framework models motions as an evolving second order dynamical system. This model structure enables convergence to target end-points and cyclical motions in continuous time. The

adaptability of a DMP comes from learning a nonlinear forcing term that modifies the dynamics of the motion during control [62, 130].

Unfortunately, dynamical system-based MPs have limited generalization capabilities because they fail to capture the variance in movement patterns. A more powerful formulation has been the *Probabilistic movement primitive* (ProMP), which uses a linear-Gaussian model to model movements [107, 108]. An important factor in ProMPs is the basis function which encodes inductive biases better suited for different kinds of motions [77, 80, 171]. ProMPs are robust to limited demonstration data [36], and can be adapted by operations such as *blending*, where unseen movements are produced by interpolating two learned ones, and via-point conditioning, where users can specify desired target poses in a movement. These features of ProMPs make them a more powerful model of motions compared to the less flexible DMP approach.

However, Gaussian-linear models are limited to unimodal distributions and cannot represent rich, multimodal datasets or process high-dimensional variables such as images [22, 154]. Recent approaches overcome these limitations with deep learning techniques. Some researchers have proposed autoregressive neural-network-based movement primitives [40, 75, 102, 141, 142]. These models are *time-discrete*, leading to performance degradation with high frequencies [9]. Other research proposes continuous-time approaches that use a deep model to generate the parameters of movement primitive models [9, 10], but a linear model still represents the fundamental movements.

The pivotal work of Seker et al. [138] overcomes the limitations of previous work by proposing the conditional neural movement primitive (CNMP), which is a continuous-time, nonlinear representation of movement skills. These movement primitives utilize conditional neural processes (CNPs) [33, 34], an encoder-decoder model that allows aggregating multiple input-output pairs to form a latent representation of a function. A CNMP encodes motion as a latent embedding and a time input to output joint configurations of a CNP model. Aggregating multiple time-joint-configuration pairs

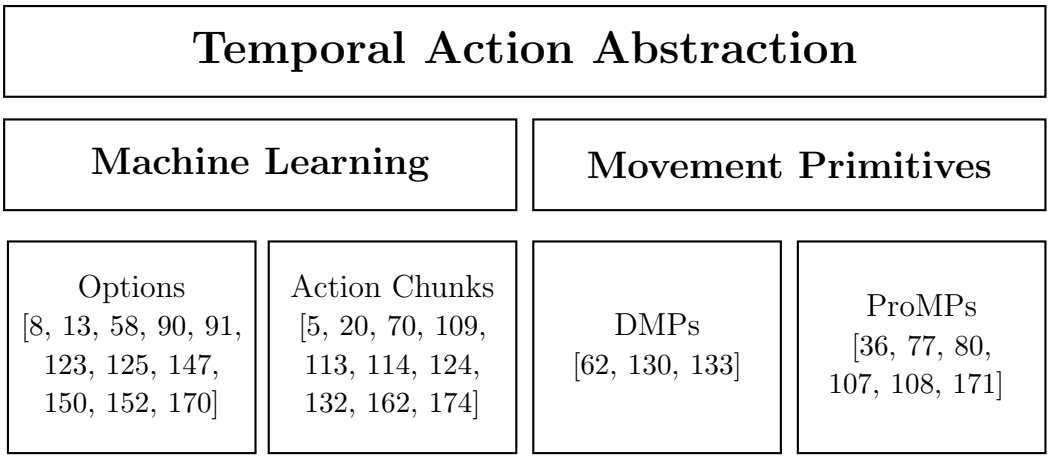


Figure 2.5: We distinguish machine learning and movement primitives as the significant categories of temporal action abstraction methods. Machine learning methods usually are deep learning focused, where as movement primitives have more structured models.

forms the latent representation of the desired motion. A nonlinear model then uses the latent representation to produce the continuous-time association between the two variables. These models influence the development of systems we propose in Chapter 5 of this dissertation.

2.4.3 Morphology-Aware Policy Learning

A general challenge applying deep learning to robotics is the effects of morphological differences between robots. Morphology differences can affect data reuse because policies trained on specific robots do not easily transfer to other robots [32, 60]. Even on the same class of robot, small deviations in dynamics and kinematics can affect deep learning-based policy performance [18, 134]. These issues motivate *morphology-aware* learning in which information about the robot's morphology is explicitly modeled in the mathematical formulation of the control policy. Morphology-aware learning is a form of contextual Markov decision process [44], in which context information (the morphology) informs the optimal control policy. Morphology-aware learning focuses on the affects of an embodiment to perform a task as opposed to solving a variety of different tasks, such as in goal-conditioned RL [98, 106, 117]. We visualize the

different means of generalizing over morphology discussed in Figure 2.6

One solution to these problems is learning policies on invariant spaces that generalize across different robot morphologies. By operating in end-effector space, the policy delegates morphology-specific computation to the robot's inverse kinematics solver to complete the task [23]. The inverse kinematics effectively factors out morphological differences between robots, enabling generalization in Cartesian space, where desired end-effector positions are specified to the robot. This approach has been widely used in large-scale data collection efforts for imitation learning [72, 105, 140] and helping to generalize RL policies between multiple robot arms [84, 89].

Unfortunately, controlling a robot in end-effector Cartesian space narrows the possible task specifications. Cartesian control is often employed for manipulation tasks, which assumes the only relevant point of contact is between the robot's end effector and objects. This assumption ignores a variety of more complex whole-arm manipulation problems in which multiple robot joints make contact with the manipulator [28, 31, 155]. Many desirable tasks beyond manipulation can be difficult to specify for other robot embodiments (e.g., quadrupeds are more difficult to control in this space). These limitations in task specification hinder the general application of Cartesian control as the solution for other robot embodiment classes besides manipulators.

An alternative to Cartesian space is learning policies directly in the robotic joint space, but this requires a model that adapts to varying numbers of actuators in the robot. Researchers address this by modeling the morphology graph of a robot at the level of each limb. Policies can then generalize over distributions of robots by processing the representation with graph neural networks [131] or transformers [157]. Researchers have considered several factors of morphology-aware policies, including the efficacy of transfer learning [41, 60, 161], tokenizing graph representations [54, 156], evaluating simulated robotic designs [110, 169], and more effectively defining inductive biases directly in the neural network [46, 79, 139, 166].

However, existing morphology-aware policy learning research has largely ignored

computational constraints when using robotics in the real world. Robots often have limited onboard computing power, so control algorithms must be efficient [59, 100]. Hardware constraints make incorporating deep learning difficult, especially as many neural network models employ foundation models (tens of millions to billions of parameters) trained on internet-scale text or image data sets [99, 120, 121]. Policy performance often degrades when structural variations occur between robots in the same morphology class, especially if the policy has not encountered that robot during training [66, 84, 86, 143]. These computational and performance limitations help motivate our interest in learning modularized components. By learning smaller components, we can more easily analyze failure scenarios when combining components to learn across morphology.

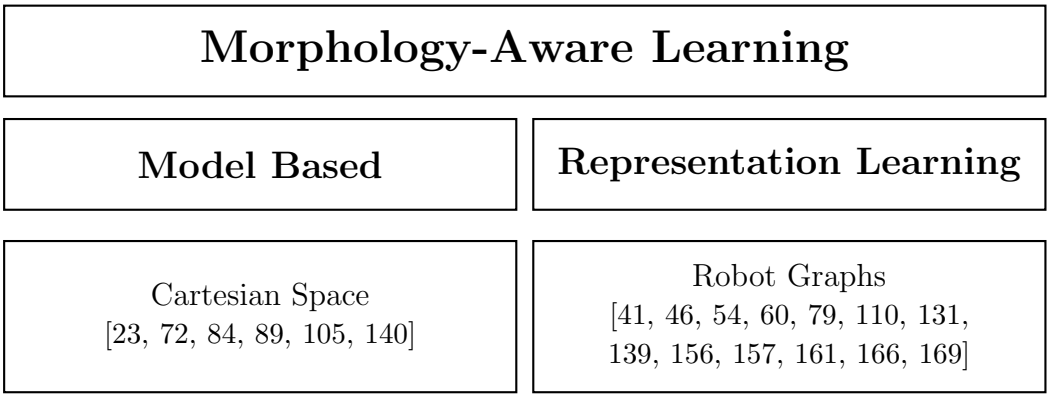


Figure 2.6: Overview of morphology-aware learning approaches. Model-based methods leverage Cartesian space control with inverse kinematics to generalize across robot morphologies, while representation learning methods use graph neural networks and transformers to process robot morphology graphs directly in joint space.

Chapter 3

Action Maps for Teleoperation Systems

This chapter discusses our contributions developing and improving the generalizability of neural action map models for teleoperation. An action map is a function that translates low-dimensional inputs to task-relevant low-level robot commands. These functions are essential in limited-degree-of-freedom (DOF) teleoperation settings. For example, users operating wheelchairs or wheelchair-mounted robot arms usually live with a disability that limits the DOFs available to their control interface relative to the robot's controllable DOFs, such as a 2D joystick controlling a 7DOF robot arm. Constraints on available DOFs are the most significant challenge in action map development.

The de facto action map in limited DOF teleoperation is Cartesian mode-switching, in which the user can decide which Cartesian axes the joystick controls, and inverse kinematics moves the robot in configuration space. Prior research suggests that Cartesian mode-switching can be tedious because requires understanding the end-effector coordinate frame, rotational movements can be especially confusing for the user [15]. These difficulties increases the user's mental effort when controlling robotic arms, even among non-disabled participants [52, 127]. Empirical results show that mode-switching accounts for 24% of task performance,and sub-optimal mode mappings contribute to slower completion times [151].

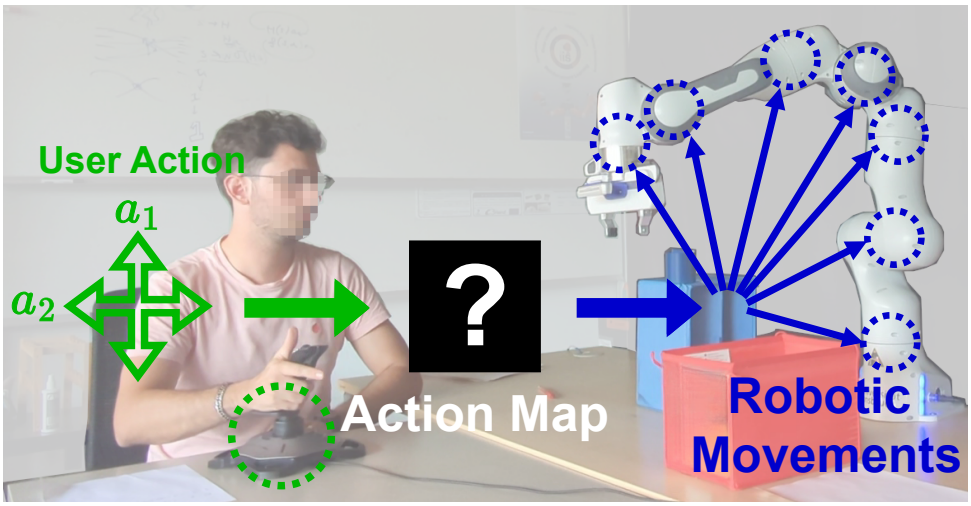


Figure 3.1: User teleoperating a 7-DOF robot using a 2D joystick with a data-driven spatial mapping interface.

In contrast, deep learning action maps learn latent representations to re-map agent actions to the original control space. These types of action abstractions are useful in teleoperation, where a human user controls an interface to operate a robot (see Figure 3.1). Previous teleoperation action maps rely on data distribution to capture necessary human-desirable behaviors, but the training distribution may not capture these properties. The uncertainty in relying on data distributions to encode necessary properties is problematic because there are multiple priors for users' expectations about how a teleoperation interface should operate. Our research contribution is showing that explicitly enforcing these teleoperation properties improves user experience when teleoperating a robot with deep learning-based teleoperation action maps

The following section describes the data-driven teleoperation problem, including definitions of teleoperation properties. We then discuss deep learning design decisions that enforce teleoperation properties, either through model design or through optimization objectives. The chapter ends with a theoretical analysis that verifies that our models enforce the defined teleoperation properties and identifies the limitations of developing more complex teleoperation systems.

3.1 Teleoperation Properties

We define the robotic teleoperation as a first-order ordinary differential equation,

$$\dot{\mathbf{x}}(t) = f_\theta(\mathbf{x}(t), \mathbf{a}(t)), \tag{3.1}$$

where $\mathbf{x}(t) \in \mathbb{R}^d$ represents the state of the robot (i.e., joint configurations or end-effector's pose), $\dot{\mathbf{x}}(t) \in \mathbb{R}^d$ is velocity, $\mathbf{a}(t) \in \mathbb{R}^n$ is the user action (e.g., the 2D position of the joystick as in Fig. 3.1), representing raw control inputs rather than augmented or homogeneous coordinates, and $t \in \mathbb{R}^+$ is the time. The function $f_\theta : \mathcal{X} \times \mathcal{A} \to \dot{\mathcal{X}}$ is referred to as the *action map* which is trained to map states $\mathcal{X}$ and user actions $\mathcal{A}$ to robot velocities $\dot{\mathcal{X}}$. We assume that the teleoperation dynamical system evolves following Euler's method,

$$\mathbf{x}(t + \tau_i) = \mathbf{x}(t) + \tau_i f_\theta(\mathbf{x}(t), \mathbf{a}(t)),$$

where $\tau_i \in \mathbb{R}^+$ is the discretization step. We consider these modeling assumptions reasonable because teleoperation is a closed-loop system with measurable control frequency. The robot's sampling rate constrains decisions to occur at discrete time intervals rather than continuously. The trajectory of the action map is determined online by the internal policy $\pi(\mathbf{o}_h(t)) = a(t)$ of an unknown human observation $\mathbf{o}_h(t) \in \mathbb{R}^m$. We could further define a Markov decision process incorporating the dynamical system into the transition operator, and derive the objectives to solve for the control policy and define an objective the control policy $\pi(\mathbf{o}_h(t)) = \mathbf{a}$ is solving, but this is outside the scope of the research.

Given the dynamical system described above, we provide definitions of previously suggested teleoperation properties [81, 87, 88, 118]:

1. **Controllability:** Users expect to be able to reach arbitrary robot configurations, or that from any $\mathbf{x}$ exists a sequence $A = \{\mathbf{a}(0), \mathbf{a}(t_1), \ldots, \mathbf{a}(t_{T-1}), \mathbf{a}(t_T)\}$ in the action mapping $f_\theta(\mathbf{x}(t), \mathbf{a}(t))$ to reach desired target $\mathbf{x}^\star$.

2. **Monotonicity of User Inputs Scaling:** As the user's action increases, the action map output magnitude increases, and as the user's action decreases, the magnitude of the action map decreases, i.e., for fixed $\mathbf{x}$ and $\mathbf{a}$ with $\alpha_1, \alpha_2 \in \mathbb{R}$, if $|\alpha_1| < |\alpha_2|$ then we have $\|f(\mathbf{x}, \alpha_1\mathbf{a})\| \leq \|f(\mathbf{x}, \alpha_2\mathbf{a})\|$.

3. **Consistency:** The dynamical system velocities should change smoothly w.r.t. to the state for fixed user inputs. This is achievable if we can assume f is Lipschitz continuous: $||f(\mathbf{x}, \mathbf{a}) - f(\mathbf{x}', \mathbf{a})|| \leq L\|\mathbf{x} - \mathbf{x}'\|$ for $L \in \mathbb{R}^+$.

4. **(User Input) Reversibility:** Users are able to *undo* their inputs; for state $\mathbf{x}(t)$ and $\mathbf{x}(t + \tau_i) = \mathbf{x}(t) + \tau_i f(\mathbf{x}(t), \mathbf{a}(t))$, if $\mathbf{a}(t + \tau_i) = -\mathbf{a}(t)$ users expect $\mathbf{x}(t + \tau_i + \tau_{i+1}) = \mathbf{x}(t)$.

We will show that a deep learning action map can explicitly enforce these teleoperation properties if certain features are guaranteed by the model. These results use tools from control theory to guarantee teleoperation properties are enforced. We are able to apply mathematical results from control theory because of our treatment of the action maps as a dynamical system.

3.2 Enforcing Human-Priors in Deep Learning

This section describes our contributions improving neural action maps for teleoperation. We have found that enforcing teleoperation properties requires explicit design choices in the optimization objective and the neural network structure. Prior works used the multi-layer perceptron (MLP) (see Chapter 2 for details). A limitation of MLP models is that they do not guarantee odd function behavior $-f(\mathbf{x}, \mathbf{a}) \neq f(\mathbf{x}, -\mathbf{a})$. This can be seen in affine models $\dot{\mathbf{x}} = W\mathbf{a} + b$, where due to the bias term, $-\mathbf{a}$ will not map to $-f(\mathbf{x}, \mathbf{a})$. This lack of antisymmetry occurs more generally in MLP architectures, even without explicit bias terms. Odd functions are helpful in teleoperation because users expect the systems to allow "undoing" a control command when performing the reverse action $-\mathbf{a}$. We have found that odd functions

can further allow the learning systems to enforce other teleoperation properties. A key contribution in our research has been determining how to increase user control by proposing both model architecture changes and optimization objective functions that enforce odd function behavior.

3.2.1 Hypernetwork Neural Action Maps

Our first approach to build odd neural action maps directly modifies the architecture. An MLP layer typically has a bias term b which make it challenging to enforce odd function behavior. Even if the bias is excluded, the state conditioning creates an adaptive bias dependent on the state — for a linear action map: $f(\mathbf{x}, \mathbf{a}) = W^a\mathbf{a} + W^x\mathbf{x}$. One solution is to exclude bias inducing components from both the state dependent $W^{\mathbf{x}}\mathbf{x}$) and independent (b) conditioning terms. These modeling restrictions reduce the models expressivity, but would make the action map behave as an odd function.

Alternatively, a hierarchical model can decompose state conditioning from action generation with a hypernetwork [42]. The hypernetwork predicts the weights of a sub-network based on the robot state. This sub-network is composed of n layers and only takes in user inputs. Each layer's weights can be seen as a 3D tensor product,

$$W^i(x) = H^i \otimes \phi_\theta(x) + B^i,$$

where $W(x) \in \mathbb{R}^{h \times n}$ is the subnetwork layer weights after the tensor product $\otimes$ between $H \in \mathbb{R}^{h \times w \times n}$ and state features $\phi_\theta(x) \in \mathbb{R}^w$.

Moreover, a hypernetwork bias $B^i \in \mathbb{R}^{h \times n}$ can also be learned, but is important to distinguish from typical bias terms. The hypernetwork bias changes the predicted subnetwork weights, not the sub-network predictions like an in MLP. If $W(\mathbf{x})$ were the 0 matrix, the layer would become $B^i\mathbf{a}$ representing the unconditional model described previously.

In addition to the hyper-linear weights, all activation functions σ of the subnetwork

must be odd in order for the action map to be odd end-to-end. This means activation functions that are asymmetrical such as rectified linear units or sigmoid cannot be chosen, and instead functions such as tanh, sinusoidal functions or odd polynomials must be used. Combining these two observations, we can construct the action map as an arbitrary depth subnetwork as follows:

$$\dot{x} = W^n(\mathbf{x}) \circ h_{\mathbf{x}}^{n-1} \circ h_{\mathbf{x}}^{n-2} \circ ...h_{\mathbf{x}}^2 \circ h_{\mathbf{x}}^1 \circ \mathbf{a},$$

where $h_{\mathbf{x}}^i(\mathbf{a}) = \sigma(W^i(\mathbf{x})\mathbf{a})$ is the i-th hidden layer of the subnetwork. We visualize this architecture in Figure 3.2. We refer to these conditional action maps as a *nonlinear odd action hypernetwork* (NOAH).

Local Linear Maps A special case of the NOAH model is predicting a single linear mapping $\dot{x} = W_\theta(\mathbf{x})\mathbf{a}$, where $\mathbf{a} \in \mathbb{R}^d$, which we call a *local linear action map*. Such models can also have the matrix $W(\mathbf{x})$ modified through post-processing transformations. A variation considered in our work applies the Gram-Schmidt process, in which case we refer to such model as a State Conditioned Linear (SCL) action map ([118]). A second variation we consider is learning $W_\theta(\mathbf{x})$ as a full rank matrix, from which we then extract eigenvalues and eigenvectors. We will expand on this later in Chapter 4. A significant was finding that theoretically and empirically such action maps are sufficient for teleoperation.

3.2.2 Teleoperation Mapping Objective Functions

We discuss our contributions developing loss function that enforce odd function behavior. Action maps are typically trained with supervised learning objectives as discussed in Chapter 2. The training data is collected kinesthetic demonstration data of desired tasks $\{\mathbf{x}, \dot{\mathbf{x}}\} \in \mathcal{D}$. We found alternative objectives can work well in practice under different model assumptions. These modifications allow typical MLP models to enforce desirable teleoperation properties approximately.

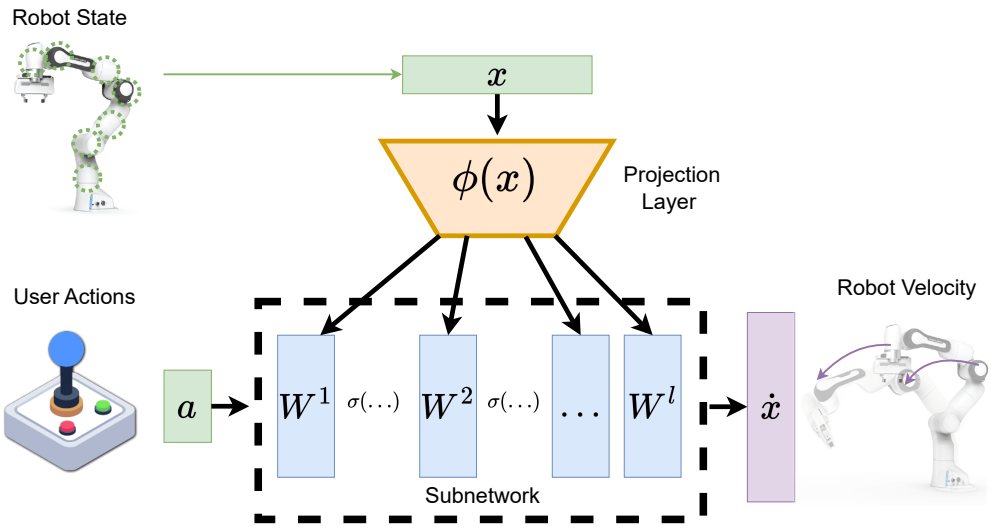


Figure 3.2: Nonlinear odd action maps ensure the action map is an odd function. NOAH generates controllable subnetworks, which are composed of a sequences of layers that preserve the odd property by using odd activations. Note that, during training, user actions are generated with an encoder.

The first objective function we proposed is an *odd function regularizer*,

$$L^{reg}(f, g, \mathbf{x}, \dot{\mathbf{x}}) = ||(-\dot{\mathbf{x}}) - f(\mathbf{x}, -\mathbf{a})||_2^2 + ||f(\mathbf{x}, g(\mathbf{x}, 0))||_2^2 + ||g(\mathbf{x}, 0)||_2^2.$$

Our proposed regularizer achieves two behaviors: (1) Enforcing odd function behavior to enable reversing and (2) encouraging that zero inputs map to zero outputs. We include this regularizer in each mini-batch by adding $\dot{\mathbf{x}} = 0$ and inverse tuples $(-\dot{\mathbf{x}}, -a)$ with the state inputs in the batch training a vanilla MLP model.

Unlike other general regularizer terms, our regularizer explicitly encourages desired teleoperation properties. The model learns to behave as an odd function, which is crucial for achieving a form of *(User Input) Reversibility*. Other reported regularizer terms only compress the action space [87] or constrain a black-box dynamics model [4], but otherwise never explicitly encourage any teleoperation properties.

Another objective we considered to improve *controllability* assumes the model is a local linear map. This assumption allows us to optimize our action map in various ways, leveraging the linear structure. We train the local linear map as a Gaussian

model by maximizing its log-probability w.r.t. data,

$$\arg\max_{\theta} \sum_{i=1}^{N} \log p(\dot{\mathbf{x}}_i|\mathbf{x}_i) = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \dot{\mathbf{x}}_i^{\intercal} \left(\boldsymbol{\Sigma}_\theta(\mathbf{x}_i)\right)^{-1} \dot{\mathbf{x}}_i - \log\det\left(\boldsymbol{\Sigma}_\theta(\mathbf{x}_i)\right), \quad (3.2)$$

where $\mathbf{x}_i$ and $\dot{\mathbf{x}}_i$ are acquired via demonstrations and $\boldsymbol{\Sigma}_\theta(\mathbf{x}) = \mathbf{H}_\theta(\mathbf{x})\mathbf{H}_\theta^{\intercal}(\mathbf{x}) + \sigma^2\mathbf{I}$ is the covariance matrix produced by the neural network. The objective in (3.2) induces the covariance matrix to capture the variability of the data, and can be minimized with standard stochastic gradient descent techniques. Our method does not require an explicit decoder to learn the control system like conditional autoencoder approaches described in Section 2. The user actions $\mathbf{a}$ are not explicitly considered during training this model.

3.3 Theory Analysis Teleoperation Enforcement

We have discussed several methods for enforcing and training action map models in the teleoperation case. In this section, we provide a theoretical analysis, which verifies that our models enforce the desired properties. Some properties are easier to verify, including consistency and monotonicity of user input scaling. Below, we discuss enforcing each of the teleoperation properties we previously discussed.

3.3.1 Limitations of Nonlinear Odd Functions

An important consideration of our work has been determining if nonlinear models of user actions a are beneficial over some simple local linear model (e.g., $\dot{\mathbf{x}} = W(\mathbf{x})\mathbf{a}$). We hypothesized that *there would be no statistically significant difference between our proposed nonlinear action map methods and local linear models in terms of user experience*. One reason for this hypothesis is that local linear models achieve the previously listed teleoperation properties, giving users significant control over the system [118]. It is also plausible that during optimization, both hypernetwork and regularizer approaches converge to behave effectively as a local-linear function, which

is in the set of odd functions. This is observable in the first order Taylor series approximation at $a_0 = 0$:

$$\begin{aligned} f(\mathbf{x}, \mathbf{a}) &\approx f(\mathbf{x}, a_0) + J^{\mathbf{a}}(\mathbf{x}, a_0)(\mathbf{a} - a_0) \\ &\approx f(\mathbf{x}, 0) + J^{\mathbf{a}}(\mathbf{x}, 0)(\mathbf{a} - 0) \\ &= f(\mathbf{x}, 0) + J^{\mathbf{a}}(\mathbf{x}, 0)\mathbf{a}, \end{aligned}$$

where $f(\mathbf{x}, 0) = 0$ by being an odd function.

Furthermore, we show that as the discretization step decreases, the induced robotics' trajectory under the linearization and original model becomes the same. We show this by deriving the error between the final state of the linearization and the original model when following the same user action sequence. Proofs are included in the Appendix A. The result relies on the following lemma to bound the error between the predictions of linearization at $\hat{x}(t)$ and true model at $x(t)$:

Lemma 1 *For two states $\hat{\mathbf{x}}(t)$ and $\mathbf{x}(t)$, the action $\mathbf{a}(t)$ under a Lipschitz continuous, odd, action map f and linearization $\hat{f} = \nabla_{\mathbf{a}} f(\hat{\mathbf{x}}(t), \mathbf{a})|_{\mathbf{a}=0}\mathbf{a}(t)$ have $\|f(\mathbf{x}(t), \mathbf{a}(t)) - \hat{f}(\hat{\mathbf{x}}(t), \mathbf{a}(t)\| \leq M + LC$, where $M = \max_{\mathbf{a}} \|f(\mathbf{x}(t), \mathbf{a}(t))\|$, L is the Lipschitz constant, and $C = \max\{\|\mathbf{a}\| : \mathbf{a} \in A\}$.*

The intuition of this lemma is that in a robot's state space for an action map that is Lipschitz continuous and has bounded user inputs there exists some maximal distance of robot velocities between two arbitrary robot states. With this result, we can then prove that as the discretization step $\tau \to 0$, these errors disappear.

Theorem 1 *For an odd action map f that has Lipschitz constant L on a bounded action space, following Euler integration, the distance between trajectory $x(t)$ and $\hat{\mathbf{x}}(t)$, where $\hat{\mathbf{x}}$ follows linearization $\hat{f}$ of f, we have $\|\mathbf{x}(t) - \hat{\mathbf{x}}(t)\| \leq \tau(T-1)(M + LC)$, where τ is the discretization step and T is the number of discrete steps taken.*

This theorem suggests, particularly for shorter trajectories, users likely would not notice substantial changes in motion between the model's local linear approximation and original prediction.

An important consequence of this result is that teleoperation systems can be modeled using a local-linear structure. Since our local-linear models reduce to *driftless* affine control systems — a well-studied class in control theory [78] — we can leverage existing theoretical frameworks to analyze controllability. This connection allows us to rigorously establish the conditions under which an action map remains controllable in Subsection 3.3.4.

3.3.2 Consistency

Consistency is one of the easier properties to enforce in deep learning systems because our definition corresponds with guaranteeing the model is Lipschitz continuous. A multi-layer perceptron will be Lipschitz continuous so long as the activation functions are Lipschitz continuous — which is true for tanh and ReLU activations. The Lipschitz coefficient can then be upper bounded by a projection operation on the linear weights,

$$W^l \leftarrow W^l / \min(\lambda, 1)$$

where $\lambda = ||W^l||$ is the spectral norm of the weight matrix W^l in the $l \in \mathbb{N}^+$ layer of the model. This algorithm was proposed in the work of Gouk et al. [38].

3.3.3 Enforcing Action Monotonicity

We establish monotonicity properties for various architectural components. The monotonicity in user inputs is defined as: for scalars $\alpha_1, \alpha_2 \in \mathbb{R}^+$ with $\alpha_1 < \alpha_2$, a function f is monotonic if $||f(\mathbf{x}, \alpha_1 \mathbf{a})|| \leq ||f(\mathbf{x}, \alpha_2 \mathbf{a})||$ for any action vector $\mathbf{a}$.

Theorem 2 *A local linear action map $W(\mathbf{x})$ is monotonic in user inputs for any pair of $\alpha_1, \alpha_2 \in \mathbb{R}^+$ with $\alpha_1 < \alpha_2$.*

For some $\alpha \in \mathbb{R}^+$ such that $\hat{\mathbf{a}} = \alpha\mathbf{a}$, we have $W(\mathbf{x})\hat{\mathbf{a}} = \alpha W(\mathbf{x})\mathbf{a}$, directly supporting monotonicity. In the general case, as long as activation functions are monotonic and no bias terms are used, this condition extends to deeper architectures.

Theorem 3 *For a NOAH architecture with monotonic activation functions σ, the complete model maintains monotonicity in user inputs.*

For our regularization objective, we impose a strong structural constraint that leads to favorable monotonicity properties:

Theorem 4 *Suppose we optimize the function $f_\theta(\mathbf{x}, \mathbf{a})$ such that $f(\mathbf{x}, -\mathbf{a}) + f(\mathbf{x}, \mathbf{a}) = 0$. Then f is an odd function in $\mathbf{a}$ and exhibits monotonic behavior in the first-order approximation.*

Our main architectural approaches—including local linear maps, NOAH networks with monotonic activations, and functions regularized to be odd—all converge to monotonic behavior in user inputs. This convergence provides theoretical justification for the stability and predictability of our control systems. These results demonstrate that our design choices lead to reliable and consistent system behavior.

3.3.4 Determining Teleoperation Controllability

The originally proposed definition of teleoperation controllability requires that the model be *globally* controllable, meaning that action sequences exist to move between arbitrary robot states. Proving a nonlinear model achieves this is challenging for complex nonlinear systems. Towards this goal, we analyze whether our models are *small-time locally controllable* (STLC), for state $\mathbf{x}(0)$ and $\epsilon \in \mathbb{R}^+$, we have the reachable set $R(\mathbf{x}(0), \mathcal{A}, t', f_\theta) = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}(0)\| \leq \epsilon\|\}$. If a state is small-time locally controllable, it suggests that a user can move a robot in arbitrary directions with the teleoperation map.

Criteria to verify that an affine control system is STLC in a given state are well established[1], but identifying all states $\mathbf{x} \in \mathcal{X}$ that are STLC is non-trivial. We find the set of STLC states of a neural teleoperation map by analyzing the affine map, $\mathbf{H}_\theta(x) = [W_1\phi(\mathbf{x}); W_2\phi(\mathbf{x}); ...; W_n\phi(\mathbf{x})]$, for some basis function $\phi : \mathcal{X} \to \mathcal{H}$, and $W_i \in \mathbb{R}^{n \times h}$ $(h \in \mathbb{N}^+)$:

Theorem 5 *For an affine control system* $\dot{\mathbf{x}} = \sum_{i=0}^{n} \mathbf{h}_i(x)a_i$*, where* $\mathbf{h}_i(\mathbf{x}) = W_i\phi(\mathbf{x})$*, all states* $\mathbf{x} \in \mathcal{X}$ *are small-time controllable if (1) The matrices* W_i *are not linear dependent (i.e., there does not exist coefficients* $\alpha_j \in \mathbb{R}$ *such that* $\sum_{i \in \{1,...,n\} \setminus \{j\}} \alpha_j W_j = W_i$*) and (2) that for* $i \in [1, ..., n]$*, we have that* $\phi(\mathbf{x})$ *is not in the null space* W_i*.*

This theorem simply shows the conditions when $H(\mathbf{x})$ is a full rank matrix for systems with control inputs equal to the number of state dimensions. An interpretation of condition (1) shows the exact user action that could produce redundant motions under a low-rank $H(\mathbf{x})$. For the condition (2), a consequence is that for linear neural networks, $\phi(x) = x$, one can identify the set of non-trivial states $(\mathbf{x} \neq 0)$ which will induce low-rank basis by finding the null space of each basis matrix W_i. We provide other results in the appendix A.3.1.

More importantly, as typical neural networks have higher hidden dimensions than the state, $h \gg n$, the null space of W_i will be non-trivial, so it is likely in practice that $\mathbf{H}_\theta(\mathbf{x})$ contains at least one basis $\mathbf{h}_i(x) \approx \mathbf{0}$. The system noise parameter helps maintain a full-rank matrix, even when the matrix prediction is low rank. This hyperparameter thus helps users to maintain maximum control while teleoperating the robot.

3.3.5 Trajectory Reversibility

In this section, we provide results on user action reversibility on an action sequences, $A = [\mathbf{a}(0), ..., \mathbf{a}(\sum_{i=1}^{T} \tau_i), -\mathbf{a}(\sum_{i=1}^{T} \tau_i), \ldots, -\mathbf{a}(0)]$, as opposed to just a single step

[1]See Chapter 15 in Lavalle [78] for full discussion, and specifically equation 15.110 for the full conditions of verifying small-time local controllability.

in a trajectory. These inverse trajectories represent the action sequence from state $\mathbf{x}(\sum_{i=0}^{T} \tau_i)$ to $\mathbf{x}(\sum_{i=0}^{T-k} \tau_i)$ for some previous state $k \in \mathbb{N}^+$. We first show that in continuous time

Theorem 6 (Trajectory Reversibility) *In the interval* $0 \leq t \leq 1$ *following* f *in Equation 3.1, and assuming that* $\mathbf{a}(t+1) = -\mathbf{a}(t-1)$*, we have* $\mathbf{x}(2-t) = \mathbf{x}(t)$*.*

We show that the induced vector field with the inverse action sequence will return close to the initial state. The ability of the system to arrive *close* to the initial state is called *soft* trajectory reversibility. Throughout this section, we assume that the system follows the dynamics of (3.1) and that the map f is odd, i.e., $f(\mathbf{x}, -\mathbf{a}) = -f(\mathbf{x}, \mathbf{a})$. Due to discretization errors, reversing trajectories may deviate from any initial state. We denote the visited states because of discretization as $\mathbf{x}_T(K) = 1/T \sum_{i=0}^{k} \mathbf{x}(i/k)$ for duration $T \in \mathbb{N}^+$. We account for discretization by bounding the distances from an initial state over a trajectory of states.

Theorem 7 (Trajectory Soft Reversibility) *Suppose there is an odd and bounded function* $f(\mathbf{x}(t), \mathbf{a}(t))$ *that is Lipschitz continuous* $\|f(\mathbf{x}(t), \mathbf{a}(t)) - f(\mathbf{x}'(t), \mathbf{a}(t))\| \leq L\|\mathbf{x}(t) - \mathbf{x}'(t)\|$*, and* $M = \max_{\mathbf{x}\in\mathcal{X}, \mathbf{a}\in\mathcal{A}} \|f(\mathbf{x}, \mathbf{a})\|$*. For points,*

$$\mathbf{x}_T(k) = \mathbf{x}(0) + \nu \sum_{i=0}^{k-1} f(\mathbf{x}_T(i), \mathbf{a}_T(i)),$$

such that $\mathbf{a}_T(T+k) = -\mathbf{a}_T(T-k)$*,* $\nu \in \mathbb{R}^n$ *for a fixed duration* $T \in \mathbb{N}^+$*, we have*

$$\|\mathbf{x}_T(0) - \mathbf{x}_T(2T)\| \leq \nu M \left(\exp\left(TL\nu\right) - 1\right). \tag{3.3}$$

This result further establishes a connection between the accuracy of reversibility and the sampling rate. The faster we can update the velocities from an action map, the easier it should be to reverse user actions. In Appendix A.3.2, we include results that show, in continuous time, this is indeed the case.

3.4 Conclusion

In this chapter, we presented a formalization of learning teleoperation mapping functions. Our formulation allows us to more clearly define the desirable properties for a teleoperation mapping. We then proposed several design choices that can be mathematically demonstrated to enforce these desirable properties. These results show that it is possible to build deep learning teleoperation systems with verifiable properties. Our work differs from previous efforts, which only show empirical evidence on their system's ability to encode teleoperation properties. At this point, we have only theoretically verified that our systems enforce consistency, monotonicity, reversibility and controllability in a teleoperation system. We will discuss our experimental evaluations in the next chapter.

Chapter 4

Teleoperation User Studies

In this chapter, we show a number of practical results of the benefits and limitations of our teleoperation systems. We conducted a variety of user studies to understand different components of our teleoperation models. As we investigate different models for each set of experiments, which we will make clear on these model distinctions. In particular, we will first establish the benefits of the state conditioned linear map over traditional MLP architectures in the first section, and then a simplification of this system we refer to as the *DeepPCA* approach. They are just variations of ideas discussed in Background (Chapter 2) and Teleoperation (Chapter 3). All user studies conducted with the approval by the University of Alberta Research Ethics and Managements (Pro00054665).

4.1 Comparing Modeling Choices for Teleoperation

In this section, we discuss user study experiments that investigate comparing neural network architecture and loss function choices. The goal of this series of user studies was to determine the most effective means of building a learnable teleoperation mapping. In our experiments, we seek to address the following questions:

Q1) How does an action map enforcing teleoperation properties compared to prior methods?

Q2) How do action maps compare to *mode-switching* teleoperation maps?

Q3) When learnable teleoperation maps *enforce* teleoperation properties, are some techniques superior to others?

Throughout this section, we will refer to *state conditioned linear* mapsl (SCL) when discussing a local linear model. This is a specific architecture we proposed in Przystupa et al. [118]. The key characteristics of this model include,

1) Optimizing the action map as an autoencoder as discussed in Background 2 optimzizing only the reconstruction error.

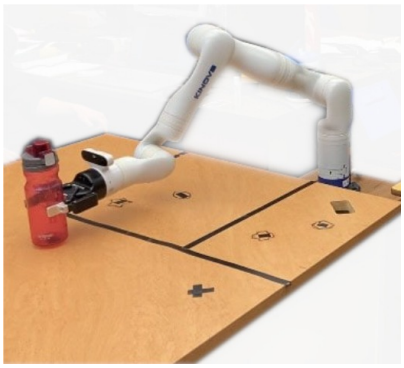
2) using a *single control mode* like in prior work using MLP models [87, 88].

3) Post-processing the linear map $W(\mathbf{x})$ with the Gram-Schmidt process to orthonomalize the linear basis. Orthonormalizing the basis makes to define a maximum user action norm applicable to every state ($||\mathbf{a}|| \leq \delta$ for some $\delta \in \mathbb{R}^+$)‘.

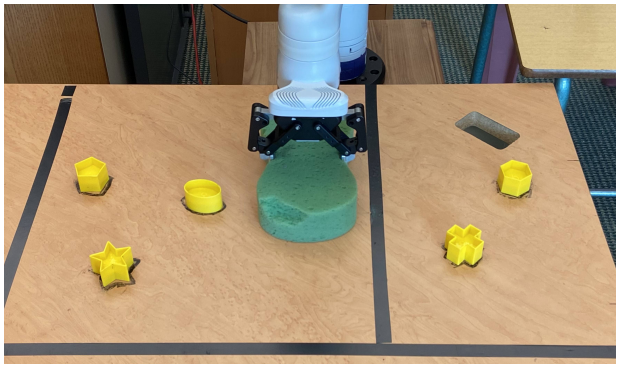
4.1.1 SCL perform as well or better than prior learned action maps (Q1)

The first user study establishes the efficacy of SCL compared to other function classes to learn an action map. We chose tasks that can be solved with 2D and 3D Cartesian control, which we refer to as `Table Cleaning` and `Pick and Place` in this section. Despite being simple, these tasks allow us to empirically evaluate the efficacy of SCL in general compared to existing learnable action mapping approaches. See Figure 4.1 for setup.

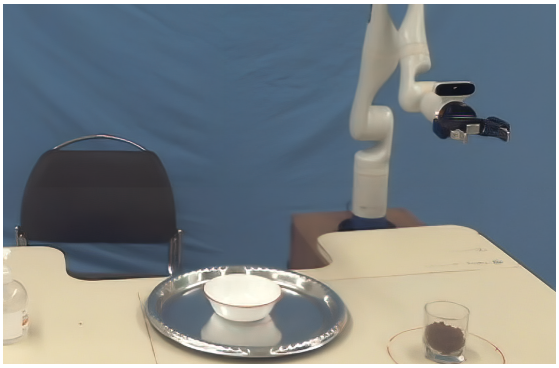
In the `Table-Cleaning` task, participants were asked to push five small objects on a table out of a designated boundary in 60 seconds. We collected 5 demonstrations of circular scrubbing motions on a table to train the models. Three trajectories were used to train the PCA, SCL and CAE models, and the remaining two were held out for validation. The total number of training data was 509 ($\mathbf{x}$,$\dot{\mathbf{x}}$) tuples.



(a) Pick and Place Tasks



(b) Table Wipe Setup



(c) Pouring Task

Figure 4.1: Table Cleaning and Pouring Experiments. The figure shows three different experimental setups used in our evaluation.

Afterward completing trials for the first task, participants performed a `Pick and Place` task. Participants teleoperate the robot to reach from a home configuration to grasp the bottle, which was initially placed on the black "x" marked on the table within 90 seconds. The participants were then instructed to move the bottle to another fixed, marked position on the table. Grasping was controlled by a trigger on the joystick controller. We collected eight demonstration trajectories for `Pick and Place`. We used six trajectories for training and two for validation. The six training trajectories accounted for 2001 ($\mathbf{x}$,$\dot{\mathbf{x}}$) tuples.

For each task, participants were given five minutes to practice, and two attempts at the task. We had 13 participants (4 females and 9 males, age 23 – 33). We followed a single-blind experimental set-up, randomizing the order of models each participant used for each task.

We compare SCL against other learnable action mapping methods in the literature including principle component analysis (PCA) [6, 104] and conditional autoencoders (CAE) [87, 88]. Respectively, they represented a linear and non-linear approach for learning low-to-high dimensional mappings for control. Our PCA models were trained using only observed velocities $\dot{\mathbf{x}}$ where we ignore normalizing by the mean. This choice avoids the issues with adding a bias term discussed in Chapter 3. Our CAE model is based on the open-source version of Karamcheti et. al. [68] — see Chapter 2 for more details on CAE.

The conditional autoencoder and SCL maps were trained on demonstration data to fit the mean-squared error between the recorded and reconstructed joint velocities (i.e. with unsupervised learning). The SCL and CAE models used neural networks with two hidden layers (256 units) and tanh activation functions. For each algorithm we chose the best model (out of ten) based on a validation loss for deployment. Participants used a 2-DOF joystick to provide low-dimensional control commands with inputs mapped as actions $\mathbf{a} = [a_0, a_1]^\top$, with $\|\mathbf{a}\| \leq 1$.

Results and Discussion Our empirical results are included in Figure 4.2 where we report success rates. Participants used all three systems, so we compare inter-user results. Across the trials, we observed distinct patterns of behavior for each method: in successful trials with PCA and SCL, user inputs translated predictably to robot motions, enabling smooth and intuitive control; CAE trials, however, frequently exhibited problematic behaviors including non-zero motion when no input was provided and unpredictable actions when the robot reached configurations outside the training distribution, making task completion difficult even for motivated participants.

We find that PCA worked well for participants in the `Table Cleaning` task, but it failed to perform consistently in the `Pick and Place` task. There was a qualitative difference in the motions required to reach the bottle versus the motions needed to move the bottle to its destination (i.e. different motions were needed to "pick" vs. to "place"). PCA was forced to capture all variation in joint velocities for both pick and place motions with just two principal components. SCL, on the other hand, was free to adapt the subspace of joint velocities, given the current configuration of the manipulator. This meant that the joint velocity commands available to the participants (via output of action map $g(\mathbf{x}, \mathbf{a})$) could be entirely different when reaching for the bottle, versus after the bottle had been grasped. We observed that the ability of SCL to use the current state to adapt the action maps for locally useful actions resulted in a statistically significant advantage in the pick and place task, in terms of the number of successful trials for each participant (Figure 4.2).

Based on our observations, the low success rate of the CAE model could be attributed to two reasons. First, the CAE internally uses affine mappings as opposed to linear mappings. When a user's input is $\mathbf{0} \in \mathbb{R}^2$ (no input), the joint velocity generated will be determined by the bias terms in the hidden layers of the model, causing the robot to drift even without user commands. The SCL method does not suffer from this problem, because it instead transforms actions with a linear map. Second, we observed that many CAE trials resulted in failure if users navigated to

joint configurations outside of the training trajectory distribution. In these states, the robot behavior became unpredictable, and in many cases counter-productive (e.g. all actions appeared to move the end-effector away from the bottle). These issues of unpredictable behavior have been previously seen in the literature [87]. In contrast, the soft reversibility property of SCL enabled participants to recover from configurations outside of the training distribution.

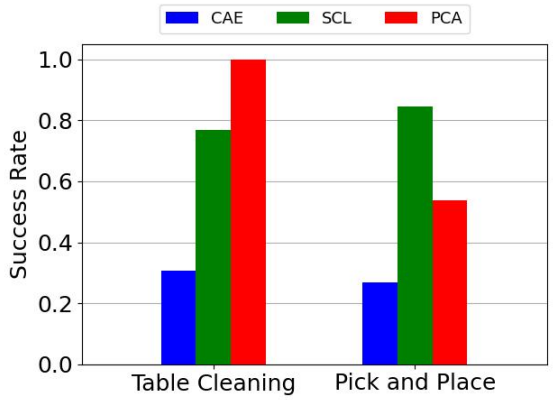


Figure 4.2: Success rates for the Table Cleaning and Pick and Place tasks. SCL outperforms PCA in the more complex pick-and-place task due to its ability to adapt action maps based on robot configuration, while CAE shows poor performance in both tasks.

4.1.2 SCL works comparably to mode-switching (Q2)

In our second user study, we were interested in comparing SCL to systems that have previously been used for assistive robotics including CAE and a mode switching system (MS) [101]. We set-up a `Pouring` task (Figure 4.1), where participants had to pick-up a cup full of beans, pour them into a bowl, and then replace the cup in a designated location on the table within two minutes. This task was highly nonlinear, consisting of several motions that involved both translation and rotation of the robot's end-effector. We collected 10 demonstration trajectories, where 8 were used for training (5200 training tuples) and 2 for validation (1191 validation tuples).

This study included 14 participants (ages 22 – 51, 2 female). Participants were

given four two-minute practice trials with each of the control systems to familiarize themelves with the action mappings. We then collected results for each method over three test trials.

We report the average success rates of participants in Figure 4.3. We found significant differences in mean success rates between interfaces (p¡0.05 by Kruskal-Wallis test). Post-hoc pairwise comparisons using Dunn's test" revealed significant differences between CAE and both SCL and MS, but not between SCL and MS. Generally, users could pick up the glass with any interface but struggled to pour with the CAE model. Generally we found users could pick up the glass with any interface, but struggled to pour with the CAE model.

In addition, we also collected user subjective opinions which included a 5 question Likert scale survey (1 low and 7 high) featured in Figure 4.4. We found that only Control and Reversibility were statistically significant by Dunn's Test. We also collected the NASA Workload Index [47] featured in Figure 4.7. Again, by Dunn's test we found statistical significance between CAE and both SCL and MS.

Although our results show the promise of SCL in assistive robotic settings, further work is necessary to improve the user experience. With a single mode, SCL performed comparably to mode switching, which exposed three control modes (x-y, z-yaw, and roll-pitch, following Newman et al. [101]). On average, users switched modes 17.55 ± 3.91 times while using the mode switching interface in the pouring task. We conjecture that on even more complex tasks (e.g. dynamic tasks, or tasks requiring simultaneous orientation and position control), the benefits of SCL would be more pronounced over typical assistive robotic systems. In previous works, tasks have often consisted of several sub-tasks and were achievable with CAE models. Interestingly, our results would seem to disagree with existing work on CAE models for shared autonomy. One explanation could be that previous research included additional heuristics to account for CAE limitations. We again found similar to the previous user study that CAE models move without user input, because the model architecture does prevent this

behavior by design. It is possible to address this heuristically (e.g. send 0 velocity if the action norm is small), but our focus was on achieving properties of the interface end-to-end.

4.1.3 Action maps trained as odd functions behave similarly (Q3)

In this section, we report the results of a user study comparing different teleoperation enforcing variants we discussed in Chapter 3. We compare SCL maps as a baseline [118]. We refer to hypernetwork models introduced in Chapter 3.2.1 as *NOAH* models, and refer to CAE models trained with odd-function regularizer (see Chapter 3.2.2) as AE (Loss).

We first evaluated the quality of action maps with empirical metrics in simulation. We measured each model's prediction accuracy and the effect of the local linear approximation away from $a_0 = 0$. We then conducted simulated teleoperation experiments to investigate how well the models could track trajectories. We finish the section with results from our user study that evaluated our hypothesis on non-linearity's benefit with able-body participants, testing whether statistical significance exists in subjective and objective metrics.

In all experiments, we used the same hyperparameters to train each system. Each model had three linear layers with tanh activations. NOAH models had 32 units in the subnetwork layers and 48 in the hypernetwork layer, whereas SCL and AE models had 256 neurons per layer. These layer parameters keep the total parameters similar between models, but NOAH models had slightly fewer parameters (approximately 65,000 vs. 70,000 for other models). We trained each model for 1000 epochs for regression and simulated teleoperation experiments, and 500 epochs for the user study experiments. Models are trained with mini-batches of 256 with the Adam optimizer with a learning rate of 0.001.

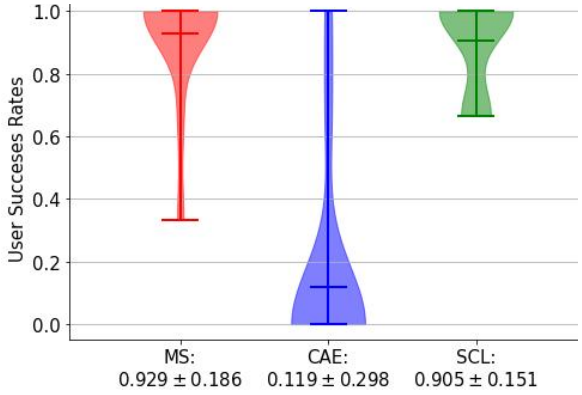


Figure 4.3: Percentages User Completions

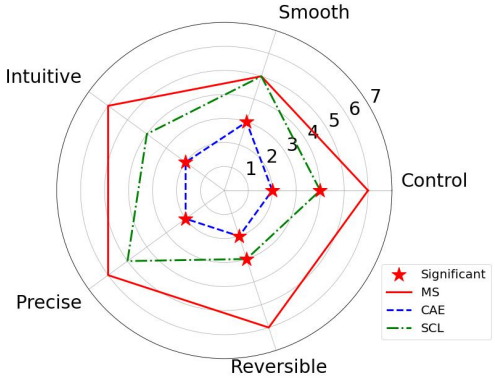


Figure 4.4: Median Likert Scale results

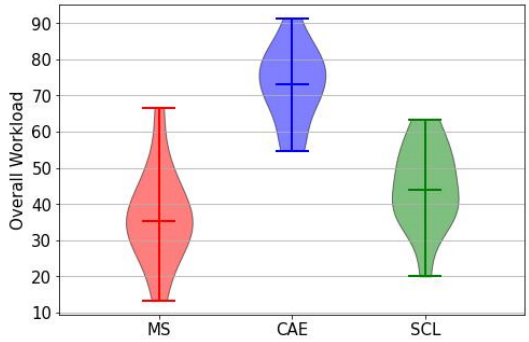


Figure 4.5: NASA Workload Index

Pouring User Study Results. Figure 4.3 shows violin plots of task completion rates with mean values, demonstrating significant differences between interfaces (p¡0.05, Kruskal-Wallis), with CAE performing significantly worse than both SCL and MS. Figure 4.4 presents radar plots of 5-question Likert scale responses (1-7 scale), with stars indicating statistical significance relative to mode switching; only Control and Reversibility dimensions showed significant differences. Figure 4.7 displays violin plots of NASA TLX workload scores with median values, showing significant differences between CAE and both SCL and MS (Dunn's test), while SCL and MS performed comparably across metrics.

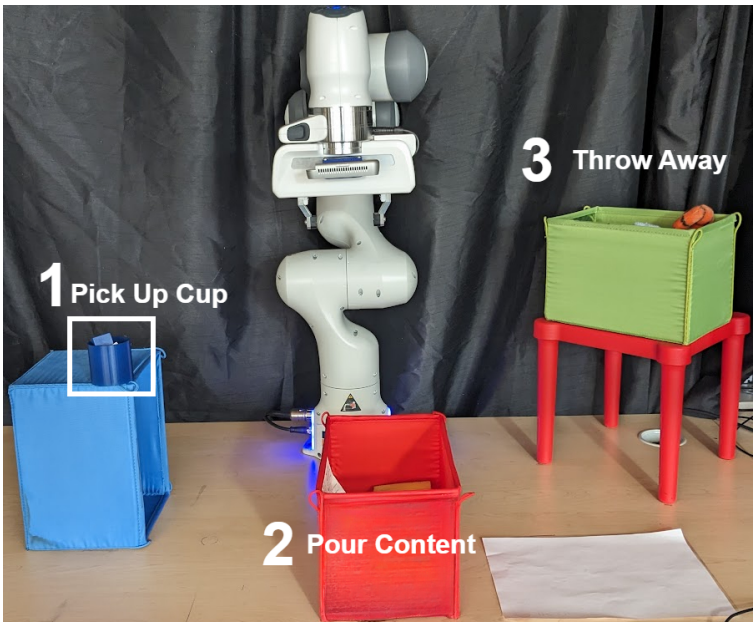


Figure 4.6: The user study task consists of three phases: (1) Pick up cup, (2) pour its content in the red bin, and (3) dispose of the cup in the green box.

Experiment Set-up

To validate our hypothesis on the impact of action map structure, we conducted a user study where participants completed a sequential multi-stage task using an *Emika Franka Robot* [43]. Figure 4.6 shows the experimental setup. We asked participants to pick up a cup from the top of a box (blue), empty its contents in the middle container (red), and throw the cup away in the corner on the opposite side of the testing space (green). This task transitioned between multiple aspects of the Cartesian pose for each subtask. We collected 32 trajectories for a total of 9270 interaction tuples, or approximately 289 steps per trajectory to train the data-driven teleoperation mappings.

We had 15 participants (13 male, 2 female) ages 20 – 55. Each participant used four different teleoperation mappings during the experiment: mode switching, NOAH, AE (Loss), and SCL. Participants completed four trials with each mapping function, where the first two trials were two minutes (training trials) and the latter two were three minutes (testing trials). Mode switching consisted of six modes: three for

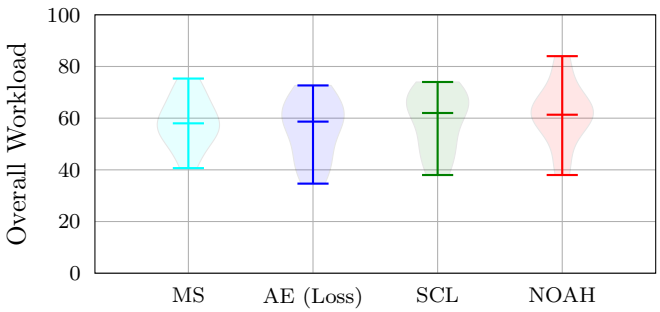


Figure 4.7: Participant NASA Task Load Index scores for the `Pour` task across teleoperation action map systems. Dunn's test revealed no significant differences between models ($p > 0.05$), suggesting comparable workload.

translation and three for orientation. In our evaluation process, every participant first completed the mode switching trials, and afterwards performed trials with the three data driven mappings in a randomized single-blind set-up. We chose to have mode switching first to help familiarize users with the joystick interface and establish baseline expectations across users to fairly evaluate the data-driven mappings. We included an option to reset the robot to the initial pose at any time during the trial, which meant participants could reset between subtasks instead of performing the movement end-to-end. This feature was included to prevent participant frustration and task abandonment when difficulties arose with a particular teleoperation system, ensuring all systems could be evaluated across the full task sequence. This strategy was not necessarily optimal because no demonstrations were collected to perform the second and third subtasks from the start pose.

Discussion

We collected several subjective metrics, including the NASA-TLX score and a subjective experience survey on a 7-point Likert scale. The keywords in our Likert questions were: intuitive, precise, smoothness, undo, and control. Figure 4.7 shows violin-plots of participants' NASA-TLX scores for each action map. We found no statistical significance between the four action maps by the Dunn median statistical test for a p-value of 0.05. We show the statistically significant Likert scale results: control be-

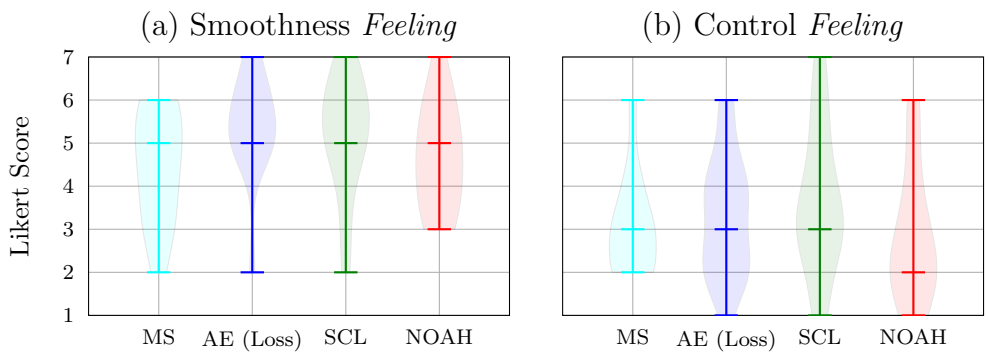


Figure 4.8: Subjective Likert score results from user study. We found statistical significance ($p < 0.05$) for *smoothness* (a) between SCL and NOAH and *control feeling* (b) between MS and AE (Loss). All other pairs were insignificant. No statistical differences were found for other questions.

tween SCL and NOAH and smooth between mode switching and AE in Figure 4.8.

Furthermore, we measured success rates of the entire task and subtasks. Complete success with any system required successful completion of each of the three subtasks. We break down results into several scenarios based on our findings. The significant categories included full success or just missing pouring, which was the usual failure case. We counted partial successes where participants completed some portions of the subtasks successfully and otherwise reported failures if no subtask was completed. We show the percentage of trials for reported scenarios in Figure 4.9.

User study results suggest that, compared to the previously proposed SCL method and mode switching, users had similar experiences with both NOAH and AE approaches. We found no statistical differences in our survey results or the NASA-TLX score, suggesting users had similar experiences across mapping functions. If we considered missing the pour as tolerable, the AE method could be considered objectively superior to NOAH and SCL where users only struggled pouring with the AE model.

However, the data used in the experiment consists of three separately collected sets of similar movements that we combined in order for the AE model to learn a deployable mapping. In contrast, both NOAH and SCL were generally able to learn better

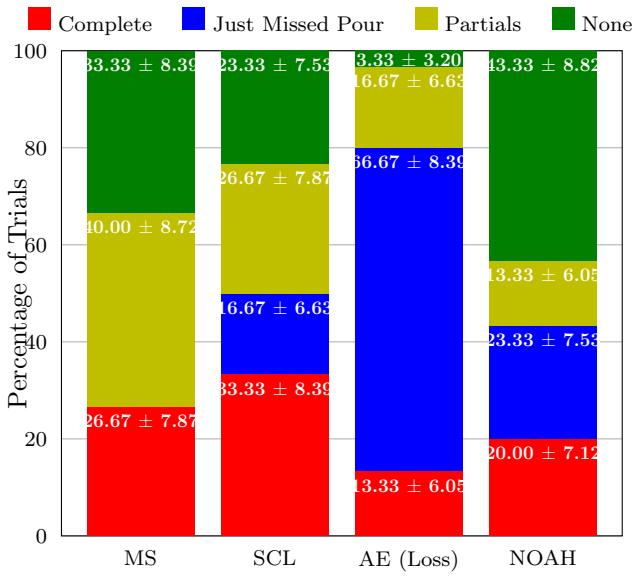


Figure 4.9: Participant success rates for different subsets of tasks completed during the teleoperation experiment. Standard deviation is calculated by treating each outcome as a Bernoulli variable and is shown at the top of each bar. "Complete" represents trials where participants succeeded from beginning to end. "Just Missed Pour" represents trials where participants successfully grasped the cup but then dropped it. "Partial" represents trials where participants completed at most one subtask (e.g., grasping the cup).

mappings with any of the three datasets we had combined. Each of these datasets only took around ten minutes to collect, but required several iterations to validate the mapping before experimental trials. This is an open challenge of our research as data-driven interfaces are sensitive to data quality and initialization. Model initialization could also affect the mapping between training runs, and investigating these affects in deployment is a promising future work.

4.2 Controllability User Studies

This section discusses experiment to understand the effects of local linear models in improving user control when providing additional control modes. Chapter 3 shows that local-linear systems can be small time locally controllable (STLC) in all states, when users are provided an action map with the same DOF as the system. This theory

assumes users can control all directions simultaneously, but with limited DOF control interfaces, *mode-switching* is a necessary feature to expand user control. Consider a robot with $\mathbf{x} \in \mathbf{R}^4$ but a user only had $\mathbf{a} \in \mathbb{R}^2$ actions. We can arrange the following two controllers:

$$\dot{\mathbf{x}} = f^1(\mathbf{a}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{a}, \quad \dot{\mathbf{x}} = f^2(\mathbf{a}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{a}.$$

In this example, the first controller controls $\dot{\mathbf{x}}_1$ and $\dot{\mathbf{x}}_2$, while the second controls $\dot{\mathbf{x}}_3$ and $\dot{\mathbf{x}}_4$. In these experiments, we study providing user's a model which provides a set of state-dependent control modes $f \in \{f^1, ..., f^n\}$ that a user can mode-switch between for a target task.

Specifying Control Modes: We study action map mode-switching with a novel variation of the SCL framework we call the Deep Principle Component action map (DPC). This model simplifies the original SCL model by optimizing the the linear map as a covariance matrix, $\Sigma(\mathbf{x}) = W(\mathbf{x})W(\mathbf{x}^T) + \lambda I$ in as described in Chapter 3.2.2. Control modes are generated from the eigenvectors of the covariance matrix,

$$\dot{\mathbf{x}} = f_\theta(\mathbf{x}, \mathbf{a}) = \mathbf{V}_i(\mathbf{x})\mathbf{a},$$

where i is the chosen mode and $\mathbf{V}_i(\mathbf{x}) = [\mathbf{v}_{id}, \ldots, \mathbf{v}_{(i+1)d}]$ are the $i-(i+d)$ eigenvectors $\mathbf{v}_i$. The eigen-modes are ordered in decreasing importance (i.e., according to the eigenvalues) at a given robot state $\mathbf{x}$.

One challenge we have identified when using the eigendecomposition is the *eigen flipping problem* in which the subsequent states have opposite direction eigenvectors — $\mathbf{v}_i(\mathbf{x}(t)) \approx -\mathbf{v}_i(\mathbf{x}(t+\nu))$. The flipping problem can causes the robot to jitters back and forth between nearby states for the same action $\mathbf{a}$. A heuristic solution is to check the the value of the cosine similarity ($c : \mathbb{R}^\kappa \times \mathbb{R}^n \rightarrow [-1, 1]$) and flip the

(a) Barrier Pick and Place (b) Shelf Pick and Place (c) Pick, Pour and Place

Figure 4.10: We used two different robots (Kinova for a and b, and Emika-Franka Panda for c) and collected data from a total of 52 participants. We used the following number of trajectories for each task: 50 trajectories for `Barrier`, 60 trajectories for `Shelf`, and 39 trajectories for `Pour`. The goal of these experiments were to analyze the benefits of Deep Principle Components for mode-switching against alternative mode-switching action maps.

sign of $\mathbf{v}_i$ if distance is near -1,

$$\mathbf{v}_i = \begin{cases} -\mathbf{v}_i & \text{if } c(\mathbf{v}_i(\mathbf{x}(t)), \mathbf{v}_i(\mathbf{x}(t+\tau_t)) < -0.95 \\ \mathbf{v}_i & \text{otherwise} \end{cases}.$$

We study the affects of controllability with this new local linear model, we were interested in addressing several research questions in understanding the controllability of DPC action maps: **Q1**) What are the benefits of adding additional control modes to learned action maps? **Q2**) What are the benefits of including state dependency when more modes are available? **Q3**) What are the benefits of learning compared to Cartesian mode-switching? We address these three questions through a series of user study experiments. Hyperparameters for neural networks used in experiments are reported in Appendix B.1.1

4.2.1 Experimental Set-up

We address research questions Q1 - Q3 with user study experiments to compare DPC against alternative action map algorithms. Our baselines include Cartesian mode-switching, PCA action maps, and variations of DPC with fewer available modes. Participant used a *Logitech G Extreme 3D PRO Joystick*, controlling 2 DOFs. Our Cartesian mode-switching mapping consisted of one translation-only mode, one rotation-

translation mode, and one rotation-only mode. DPC and PCA had three modes to be comparable to Cartesian mode-switching, and the modes were always mapped in order of eigenvalues (mode 1 had the highest-valued eigenvalue components, mode 3 had the lowest).

In this series of user studies, we followed a similar process to evaluate the efficacy of DPC for increasing controllability. We first explained the experimental procedure to participants, which included describing the task and how the joystick interface worked, including using the trigger to open/close the gripper, showing how the joystick moved the robot, which buttons changed modes, which button reset the task, and how to begin/end an experiment. Participants were then given three minutes to practice with a held-out action map, task combination not used in the main experiment. The training phase was followed by three experimental trials with a given action map model. Allotted trial times were either 3 or 5 minutes depending on the task complexity. Participants could reset the task if they felt it was too challenging to continue or due to task disruptions (e.g., dropping an object or colliding with an obstacle). We recorded resets, success or failure, duration, number of mode-switches, and joystick and robot trajectory smoothness.

After completing all three trials for a given action map, participants were asked to complete a perceived competency survey, a Likert scale questionnaire, and the NASA TLX [47, 145, 164]. These surveys measured their subjective experience of controlling the robot. For each task, we repeated this procedure until all action maps had been tested, conducting three trials and filling out surveys for the next action map interface. Action maps were assigned in a randomized single-masked setup, where participants did not know which action map they were evaluating on the task. Experiments were conducted on a *Kinova Gen-3 Lite* (Kinova) and *Emika Franka Panda* (Franka).

On the Kinova, participants completed two tasks including: *Barrier Pick and Place* (`Barrier`), where the user needs to pick a cube and move it to the placement platform while avoiding the barrier and *Shelf Pick and Place* (`Shelf`) where the user needs to

pick a cube placed in an above shelf and move it into a box on the table. The `Barrier` task is a standardized task in assistive robotic teleoperation settings [29], and the `Shelf` task represents a typical pick and place task with an added verticality component [115]. The training task for Kinova experiments was a pick and place task with cubes using a mode-switching action map in joint space.

We tested different model combinations between the `Barrier` and `Shelf` tasks. Participants performed trials with the DPC model trained with either one, two, or three available modes on the `Barrier` task because the goal was to validate the effects varying numbers of modes had on user performance. We randomly divide our cohort into three separate groups for the `Shelf` task asking participants to complete the task with DPC and either: PCA, Cartesian mode-switching, or single mode DPC. We chose to use a single baseline to minimize participant fatigue and provide participants with more time with each action map [55, 122].

The Franka experiments used a different cohort of participants at another institution. Participants in these experiments performed a *Pick, Pour, and Place* (`Pour`) task, where the user needs to pick up a jar, pour its contents into a container, and place it back on the platform. The training task was a block pick and place task with Cartesian mode-switching. Subjects were asked to compare DPC against either Cartesian mode-switching or PCA on the task. These experiments were conducted at another institution for several reasons. (1) We had exhausted our pool of participants and wanted further to validate our findings for mode-switching with DPC. (2) Participants from the other institution provide diversity in experience generalizing our results across cultures. (3) The tasks on the Kinova are easy to decompose into separate phases, where combining translation and rotation (as is necessary in `Pour`) was not tested. (4) These experiments demonstrate that DPC is generally applicable across different robot embodiments.

Summary details for the user involvement between tasks are shown in Table 4.1. Visualization of all tasks are provided in Figure 4.10. Participants were non-disabled

individuals with varying amounts of experience with teleoperation. Our participants ranged in age from 20 to 43 years old. Three participant results were excluded from the `Shelf` task results due to an error in data collection. We aggregate results between tasks when discussing results that address specific questions. We use the Mann-Whitney hypothesis test with a significance threshold of $p < 0.05$ in our results. Independent statistical calculations per-task and action-map model training information are in the Appendix.

Table 4.1: Summary details on number of participants who perform each task included in the user study. Parentheses show number of male (M) and female (F) identifying participants in each cohort. We had 52 total participants across all tasks (Kinova: 33, Franka: 19). Participants in the `Barrier` task also completed the `Shelf` task.

Task	Participants
1-mode vs 3-modes DPC	
`Barrier` (Kinova)	33 (M:23, F:10)
`Shelf` (Kinova)	10 (M:9, F:1)
PCA vs. DPC	
`Pour` (Franka)	10 (M:8, F:2)
`Shelf` (Kinova)	10 (M:7, F:3)
DPC vs. Cartesian MS	
`Pour` (Franka)	9 (M:6, F:3)
`Shelf` (Kinova)	10 (M:6, F:4)

4.2.2 Adding Modes Does Not Enhance nor Hinder Performances (Q1)

To answer whether additional modes are helpful, we looked at results from the `Barrier` and `Shelf` tasks providing users either one or three control modes with DPC. In Figure 4.11, we show results looking at completion times, success rates, and perceived competence along with p-values from running a Mann-Whitney hypothesis test. We found statistical significance ($p < 0.05$) between success rates and com-

pletion times in favour of the single control mode, but not perceived competence. These suggest that tasks with clearer mode separation — both tasks were solvable performing rotation and translation motions separately — users perform better with *fewer* available control modes. We find from the user's experience that users felt equal amounts of competence with both systems. These suggest additional modes do not negatively impact users perceived capabilities with DPC.

4.2.3 Informing Action-Maps with the Robot's State Helps (Q2)

Although results in our investigation of providing more control suggest that this can be disadvantageous to users, a practical consideration is whether simply providing users a single full-rank linear action map is sufficient. Such an action map does not require state conditioning and hypothetically should be able to reach much of the configuration space. We investigated this setting by analyzing user results on the `Shelf` and `Pour` tasks comparing DPC against classical global PCA models. We report results in Figure 4.12, which include success rates, perceived competence, and survey results from our Likert-scale questionnaire. We found statistical significance in both success rates and perceived competence in favour of DPC. These findings would suggest that state conditioning *does matter* across quantitative (success rates) and qualitative (competence and Likert questions) metrics to *improve* user teleoperation experience with DPC for completing tasks.

4.2.4 Participants are as Successful with DPC with Fewer Switches than Cartesian Mode-Switching (Q3)

This experiment challenges the benefits of mode-switching with learned actions maps compared to Cartesian mode-switching — an engineered action mapping approach — where as previous experiments only show the trade-offs with relevant factors between different learned action maps. Figure 4.13 shows the average completion time, the average number of mode-switches, and the aggregate NASA TLX score compar-

ing DPC and Cartesian mode-switching. Only average mode-switches is statistically significant in favour of DPC, as found by the Mann-Whitney hypothesis test.

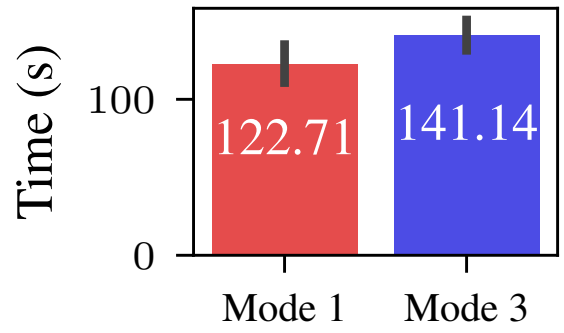
Results suggest that although DPC and Cartesian mode-switching achieve similar completion times, DPC reduces the number of mode-switches. This suggests that DPC secondary and tertiary modes provide fine-tuning motions that support the primary mode, and is supported by the per-task analysis in Table B.12 and B.19 of Appendix **??**. For Cartesian control, mode-switching is a *necessity* for completing certain tasks, particularly if there are changes in more than 2 axis of the robot's pose as shown in the results. These findings support conclusions in our previous experiments addressing Q1 where a single-mode can solve simpler tasks because it captures most of the movement variance, which corroborate findings by previous research on learned action maps [87].

4.3 Conclusions

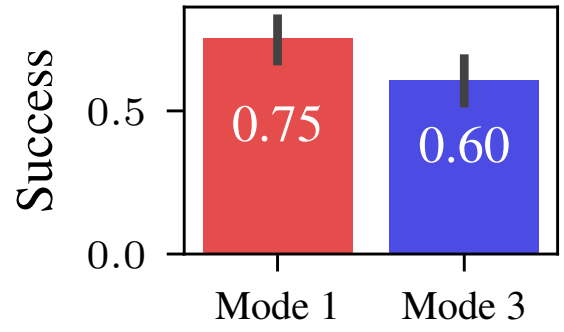
In this chapter, we have presented a variety of user study results demonstrating the potential of learned action maps that explicitly enforce teleoperation properties. When compared to neural networks trained without these properties, our approaches are measurably better on both simpler and more complex tasks. SCL maps significantly outperformed CAE models, which suffered from unintended drift when users provided no input and unpredictable behavior when the robot reached configurations outside the training distribution. In contrast, SCL's soft reversibility property enabled participants to recover from such configurations, achieving comparable performance to mode-switching systems (which required an average of 17.55 mode switches) while using only a single control mode. Among the different techniques to achieve teleoperation properties—including SCL, NOAH, and regularized autoencoders—we find that there is not a consensus on a "best" approach, suggesting that once a model behaves as an odd function it is sufficient for teleoperation. This result is supported by our theoretical results in Chapter 3, where participants reported similar NASA-TLX

scores and perceived competence across all methods that enforced these properties.

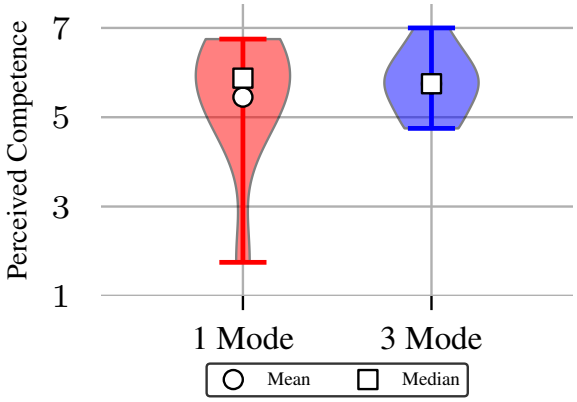
Our controllability experiments with DPC revealed nuanced insights about mode-switching in learned action maps. Surprisingly, providing additional control modes did not enhance performance—users achieved better success rates and completion times with fewer modes on tasks with clear motion separation, such as the `Barrier` and `Shelf` tasks. However, state-dependent action maps proved essential: DPC significantly outperformed global PCA models in both objective metrics (success rates, $p \leq 0.001$) and subjective measures (perceived competence, p ¡ 0.005; six of ten Likert dimensions, $p \leq 0.05$). When compared to Cartesian mode-switching, DPC achieved similar completion times and success rates while requiring significantly fewer mode switches ($p \leq 0.001$), suggesting the learned modes naturally captured task-relevant motion combinations that reduced the cognitive burden of manual mode selection. These findings indicate that state-conditioning is critical for complex tasks, but the optimal number of modes depends on task structure—simpler is often better when demonstrations already capture the essential task variations.



(a) Completion Time ($p = 0.016$)

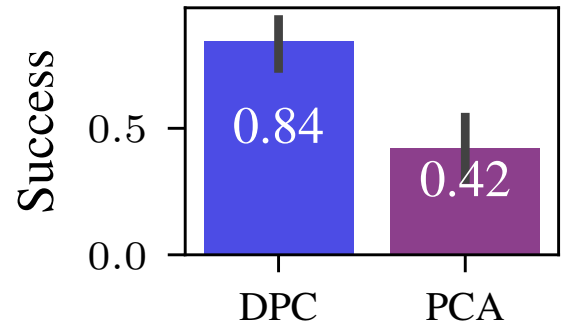


(b) Success Rates ($p = 0.015$)

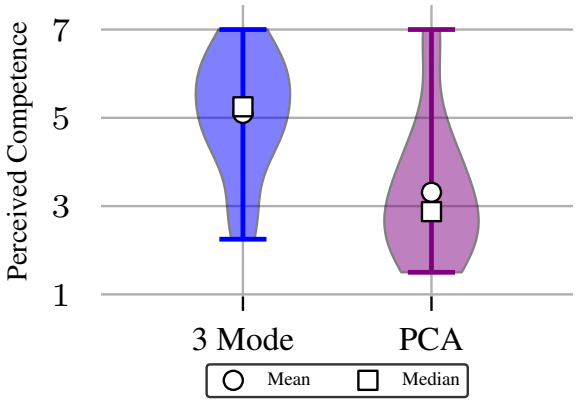


(c) Percieved Competence ($p \geq .999$)

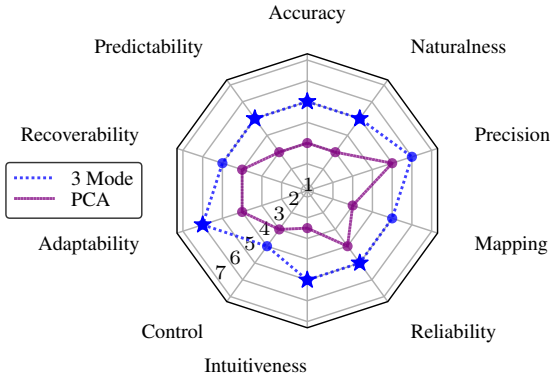
Figure 4.11: Aggregated results comparing the effects of increasing the available control modes on the `Barrier` & `Shelf` tasks to address Q1. Increasing the number of modes does not appear to increase the ability of the users to solve the tasks. Black bars show 99% confidence intervals.



(a) Success Rates ($p \leq 0.001$)

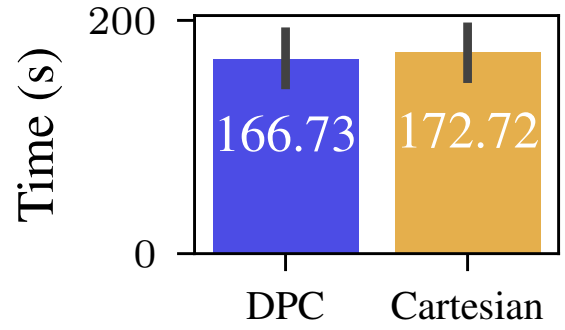


(b) Perceived Competence ($p < 0.005$)

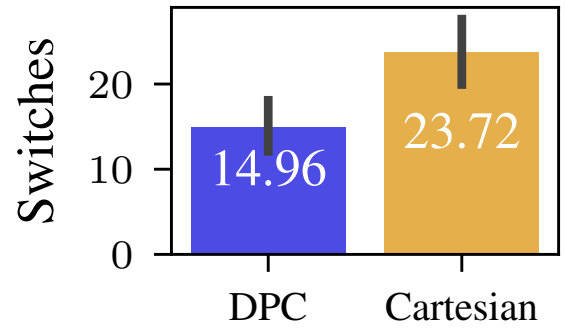


(c) Likert Scale

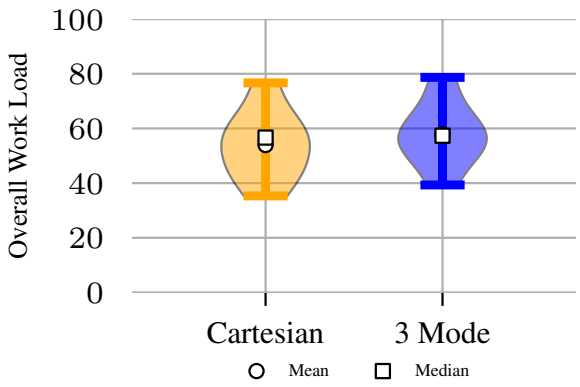
Figure 4.12: Aggregated Results of `Shelf` and `Pour` tasks. State-dependent action maps (DPC) significantly improved overall task success, compared to learning global, i.e., non state-conditioned action maps (PCA) and improved user experience — statistically significant by Mood Median test in 6 of 10 dimensions ($p \leq 0.05$) indicated by stars.



(a) Completion Time (p = 0.0838)



(b) Mode-Switches ($p \leq 0.001$)



(c) Nasa TLX Score ($p = 0.52$)

Figure 4.13: Aggregated results over the `Shelf` and `Pour` tasks. Learned action-maps significantly reduced the number of mode-switches, while maintaining similar task success nor the overall workload.

Chapter 5

Deep Probabilistic Movement Primitives

Learning from demonstration (LfD) enables robots to acquire motor skills by observing human demonstrations [5, 124, 132, 174]. A central challenge in LfD is representing demonstrated motions in a way that captures their essential structure while allowing flexible generalization to new situations. Movement primitives (MPs) address this challenge through parametric models that encode the features of a motion while permitting variations and modifications. The effectiveness of these models depends critically on their ability to represent uncertainty, handle diverse demonstration data, and generate motor skills that can adapt to different contexts through conditioning on via-points or external variables.

This chapter discusses our contributions towards developing a novel temporal movement skill model that builds off of properties of probabilistic movement primitives (ProMPs) [107, 108]. ProMPs learn a model of the distribution of motions (trajectories) shown by a human demonstrator. Once trained, the model can perform various operations to generate new movement distributions depending on the provided specifications. The ability to generate novel movement patterns is one of the significant advantages of ProMPs, which helps them generalize beyond the training movement distribution. A limitation of ProMPs is their restriction to uni-modal movements because of the Linear-Gaussian model used in the original framework.

In this chapter, we propose the *Deep Probabilistic Movement Primitives* (DeepProMP) as a deep learning equivalent of linear ProMPs. A DeepProMP represents motion primitives as a latent variable distribution, which can be manipulated to generate motions similar to the mechanisms used in ProMPs. A decoder neural network generates the desired motions from these latent variables, enabling the model to capture relationships between encoded conditioning variables and the movements. The model is optimized with variational inference [74]. This chapter provides a mathematical background and experimental evidence supporting the applications of DeepProMPs.

5.1 Background

This chapter develops a method to solve the problem of learning robotic movements from robot trajectory demonstrations. A robot trajectory includes a set of via-points $\mathcal{A} = \{\mathbf{a}_i = (\tau_i, \mathbf{y}_i)\}_{i=1}^{n}$, where via-points $\mathbf{a}_i$ indicate the configuration $\mathbf{y}_i \in \mathbb{R}^d$ of the robot at the normalized time τ_i. Each demonstration is accompanied by a variable number $m \in \mathbb{N}^+$ of *context variables* $\mathcal{C} = \{\mathbf{c}_i\}_{i=1}^{m}$ that describe content of task specific information in the environment (e.g., the pose of an object in the scene) or user specified goals (e.g., a robot pouring a specified amount of liquid in a glass). MPs aim to learn the relation between the time variable τ_i and the robot configuration $\mathbf{y} \in \mathbb{R}^d$ by imitating the user's behaviour in relation to the context variables $\mathbf{c}$.

Before describing the proposed DeepProMP framework to address modeling robot action trajectories, we provide additional background information. We begin with information on the original probabilistic movement primitive's framework that includes the definitions of operations that enhance movement primitive generalization. The section ends on describing Bayesian aggregation [160], which is mechanism that allows our model to generalize to new movements.

5.1.1 Probabilistic Movement Primitives

The probabilistic movement primitive (ProMP) is a probabilistic formulation of movement primitives that captures the variability of the demonstration data [107, 108]. A model that can encode time varying-motion provides a means to quantify critical (low-variance) or compliant moments (high-variance) in motions. ProMPs model motions as a Gaussian-linear model, $\mathbf{y} = \phi(\tau_i)\mathbf{w}$, where $\mathbf{w} \in \mathbb{R}^{d\times h}$ are the weights that are sampled from the distribution to represent the motion. The temporal features $\phi(\tau_i) : \mathbb{R} \to \mathbb{R}^h$ are a function of time, adjusted for representing either stroke-based or rhythmic motions. For stroke-based movements, normalized Gaussian radial basis functions (RBFs) are used: $\phi_i(\tau) = b_i(\tau)/\sum_{j=1}^{h} b_j(\tau)$, where $b_i(\tau) = \exp\left(-\frac{(\tau-c_i)^2}{2h}\right)$ with centers $c_i \in \mathbb{R}$ and bandwidth $h \in \mathbb{R}^+$. For rhythmic movements, Von Mises basis functions are employed: $b_i(\tau) = \exp\left(\frac{\cos(2\pi(\tau-c_i))}{h}\right)$, which naturally capture periodicity through their periodic structure. The centers c_i are typically uniformly distributed over the normalized time interval, and the bandwidth h controls the width of each basis function's influence. The normalization $\sum_{j=1}^{h} b_j(\tau)$ ensures constant summed activation across time, improving the stability of trajectory representation and enabling smooth interpolation between basis function centers.

The weights $\mathbf{w}$ are modeled as a Gaussian distribution $p(\mathbf{w}|\mu, \Sigma)$, with mean $\mu \in \mathbb{R}^{n\times h}$ and covariance $\Sigma \in \mathbb{R}^{nh\times nh}$ that capture correlations between robot joints. The model is then optimized to maximize the log-probability of the data,

$$\arg\max_{\mu,\Sigma} \mathbb{E}_{p(\mathbf{w})}[\log N(\mathbf{w}|\mu, \Sigma)].$$

The optimal distribution parameters can be found through a form of expectation maximization. First, the set of optimal weights are found per trajectory i, $\mathbf{w}_i^\star = \min_{\mathbf{w}_i} \frac{1}{2}||\mathbf{a} - \mathbf{w}_i\Phi||_2^2$, where $\Phi \in \mathbb{R}^{T\times h}$ are the stacked temporal features. These optimal weights are then used to find the distribution parameters, $\mu^\star = \frac{1}{n}\sum_{i=1}^{n} \mathbf{w}_i^\star$ and $\Sigma^\star = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}_i^\star - \mu)(\mathbf{w}_i^\star - \mu)^T$. The covariance matrix, $\Sigma \in \mathbb{R}^{dh\times dh}$, is particularly important as it encodes the correlations between different degrees of freedom, enabling

coordinated movements where the robot can achieve high precision in task-relevant dimensions (e.g., end-effector position) while individual joints exhibit larger, coordinated variability. The formulation described previously can include context variable information by concatenating the context variable $\mathbf{c}$ to each solution weight matrix, $\mu^{\mathbf{wc}} = \frac{1}{n}\sum_{i=1}^{n}[\mathbf{w}_i^\star; \mathbf{c}_i]$ and $\Sigma^{\mathbf{wc}} = \frac{1}{n}\sum_{i=1}^{n}([\mathbf{w}_i^\star; \mathbf{c}_i] - \mu_{\mathbf{wc}})([\mathbf{w}_i^\star; \mathbf{c}_i] - \mu_{\mathbf{wc}})^T$.

The advantage of encoding movement primitives with Gaussian distributions are the closed form solutions to synthesize and combine the motions via probabilistic operations. ProMP models are able to generalize movement beyond the demonstration data with just a few skill representations because of these probabilistic operations. We formalize each of these features and operations below.

Time Modulation Time modulation is the ability to modify the duration of a movement primitive. For some sampling rate $t \in [0, 1]$ we have a function $\tau(t)$ which represents the time features. More complex representations of time are considered in dynamic movement primitives (e.g. $\tau(t) = \exp(-t\lambda)$ for some scalar $\lambda \in \mathbb{R}^+$), but in ProMPs they are just the identity $\tau(t) = t$. These representations can help control how much of a movement duration is spent in certain portions of the temporal features $\phi(\tau(t))$. The temporal features $\phi(\tau)$ provide a multi-dimensional space of the time-modulation variable and allow us to define different types of motions as previously discussed.

Motion Blending Motion blending combines different proportions of multiple motion primitives into a new trajectory by computing the product of their distributions:

$$p(\mathbf{w}_1, \mathbf{w}_2) = p(\mathbf{w}_1|\mu_1, \Sigma_1)^{\omega_1} p(\mathbf{w}_2|\mu_2, \Sigma_2)^{\omega_2} \tag{5.1}$$

where $\omega_1, \omega_2 \in \mathbb{R}^+$ are blending coefficients. This operation identifies trajectories where both primitives have high probability mass, effectively finding their intersection in trajectory space rather than simply averaging them. The benefit of the Gaussian case is that we can rewrite the new parameters in closed form as:

$$\Sigma^* = \left(\left(\frac{\Sigma_1}{\omega_1} \right)^{-1} + \left(\frac{\Sigma_2}{\omega_2} \right)^{-1} \right)^{-1}$$
$$\mu^* = \Sigma^* \left(\left(\frac{\Sigma_1}{\omega_1} \right)^{-1} \mu_1 + \left(\frac{\Sigma_2}{\omega_2} \right)^{-1} \mu_2 \right).$$

Via-Point and Context Conditioning Motion conditioning is the ability to specify desired motion modifications that generate plausible motions under different specifications. Via-point conditioning specifies actions we want the primitive to perform at specific moments during the execution of a primitive. Specifying via-points can be helpful for avoiding obstacles, for example. Context-conditioning specifies an overall modification to the whole motion given desired task information. An example is a reaching task, where the context are the target positions. The key utility of the ProMP formulation is that conditioning generates motions that stay in-distribution, maintaining the structure and coordination learned from demonstrations.

For **via-point conditioning**, we specify desired configurations $\mathbf{y}^* \in \mathbb{R}^d$ at specific time points τ^*. The via-point observation is $\mathbf{a}^* = (\tau^*, \mathbf{y}^*)$, which relates to the weights through the basis functions: $\mathbf{y}^* = \phi(\tau^*)\mathbf{w}$. We then condition the weight distribution on this observation to obtain modified parameters $\mu_{\mathbf{w}|\mathbf{a}^*}$ and $\Sigma_{\mathbf{w}|\mathbf{a}^*}$.

For **context conditioning**, we leverage the joint distribution over weights and context variables learned during training. Given a desired context $\mathbf{c}^*$ (e.g., target object position), we condition the weight distribution to obtain parameters appropriate for that context.

In either case, because we assume a Gaussian distribution, we can compute the conditioning using the standard formulas for Gaussian conditionals:

$$\mu_{\mathbf{w}|\mathbf{c}} = \mu_{\mathbf{w}} + \Sigma_{\mathbf{w},\mathbf{c}} \Sigma_{\mathbf{c},\mathbf{c}}^{-1} (\mathbf{c} - \mu_{\mathbf{c}})$$
$$\Sigma_{\mathbf{w}|\mathbf{c}} = \Sigma_{\mathbf{w},\mathbf{w}} - \Sigma_{\mathbf{w},\mathbf{c}} \Sigma_{\mathbf{c},\mathbf{c}}^{-1} \Sigma_{\mathbf{c},\mathbf{w}},$$

where $\mathbf{c}$ represents either the via-point observation $\mathbf{a}^*$ or the context variable $\mathbf{c}^*$, and $\Sigma_{i,j}$ corresponds to the block matrix of the corresponding variables in the joint

distribution $\mu^{[\mathbf{w},\mathbf{c}]}$ and $\Sigma^{[\mathbf{w},\mathbf{c}]}$. For via-point conditioning, the covariance terms are computed directly from the basis functions: $\Sigma_{\mathbf{w},\mathbf{a}} = \Sigma_{\mathbf{w},\mathbf{w}}\phi(\tau^*)^T$. For context conditioning, these covariances are learned from the demonstration data as described previously.

5.1.2 Bayesian Context Aggregator

A crucial component of Deep ProMPs is to generate latent embeddings of movements. One approach to this is the *mean aggregator* $\mathbf{z} = \frac{1}{n}(\sum_{i=1}^{n} \phi(\mathbf{a}_i, \mathbf{c}_i)$, where $\phi : \mathbb{R}^l \times \mathbb{R}^c \in \mathbb{R}^d$ is a neural network that encodes conditional data $l, c, d \in \mathbb{N}^+$. A neural network decoder can then generate movements from these latent movement embeddings. Mean aggregation cannot represent epistemic uncertainty of movements, which is a significant limitation. Via-points and context variables can be associated with multiple motions, which can be difficult to capture with point-estimate representations of motions.

An alternative is *Bayesian aggregation*, which can model epistemic uncertainty by applying Bayesian inference directly to the latent variable. We define a prior distribution $p_0(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mu_0, \sigma_0^2)$, with mean $\mu_0 \in \mathbb{R}^d$ and diagonal covariance $\sigma_0^2 \in \mathbb{R}^{d\times d}$.[1] We define $p(\mathbf{a}_i|\mathbf{z}) = \mathcal{N}(\phi(\mathbf{a}_i)|\mathbf{z}, \sigma^2(\mathbf{a}))$ where ϕ is a deterministic mapping that projects $\mathbf{a}$ in the same dimension of $\mathbf{z}$. We can infer $p_{i+1}(\mathbf{z}) = p(\mathbf{z}|\mathbf{a}_{i+1}) \propto p(\mathbf{a}_{i+1}|\mathbf{z})p_i(\mathbf{z})$ by using Bayes' rule. Combining these distributions leads to the following posterior,

$$p(\mathbf{z}|\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n) = p_0(\mathbf{z}) \prod_{i=1}^{n} p(\mathbf{a}_i|\mathbf{z}),$$

which is computable in closed form when using normal distributions.[2] We use this posterior distribution to enable DeepProMPs to weigh different conditioning points depending on their variance adaptively.

[1] We keep this convention through the rest of the chapter.

[2] Notice that one can equivalently rewrite the latent distribution as $p(\mathbf{z}|\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n) = p_0(\mathbf{z}) \prod_{i=1}^{n} p(\mathbf{z}|\mathbf{a}_i)$, since, due to the symmetry of the Gaussian distribution, $\mathcal{N}(\phi(\mathbf{a}_i)|\mathbf{z}, \sigma^2(\mathbf{a})) = \mathcal{N}(\mathbf{z}|\phi(\mathbf{a}_i), \sigma^2(\mathbf{a}))$.

5.2 Model Architecture and Training

Deep ProMPs use the following probabilistic model to generate a distribution of movements,

$$p(\mathbf{y}|\mathbf{x}, \mathcal{A}, \mathcal{C}) = \int p(\mathbf{y}|\mathbf{x}, \mathbf{z}) p(\mathbf{z}|\mathcal{A}, \mathcal{C}) \mathrm{d}\mathbf{z}, \tag{5.2}$$

where $\mathbf{x}$ is a normalized time (or phase), $\mathbf{z}$ is the latent representation of the motion, and $\mathbf{y}$ is the desired robot configuration. The formulation (5.2) represents a form of encoder-decoder architecture, where latent variables $\mathbf{z} \sim p(\mathbf{z}|\mathcal{A}, \mathcal{C})$ are decoded by a neural decoder $p(\mathbf{y}|\mathbf{z}, \mathbf{x})$. The decoder $p(\mathbf{y}|\mathbf{z}, \mathbf{x})$ is implemented using a Gaussian distribution, i.e., $p(\mathbf{y}|\mathbf{z}, \mathbf{x}) = \mathcal{N}\left(\mathbf{y}\middle|\mu_y(\mathbf{z}, \mathbf{x}), \sigma_y^2\right)$, where μ_y is a nonlinear embedding from a neural network and σ_y is a hyperparameter.

A more important factor is the structure of the *encoder* $p(\mathbf{z}|\mathcal{A}, \mathcal{C})$ which applies Bayesian aggregation to encode the latent distribution $p(\mathbf{z}|\mathcal{A}, \mathcal{C})$ from a varying number of via-points and context variables. The encoder processes two different typologies of inputs: the set of via-points, denoted with $\mathcal{A} = \{\mathbf{a}_i\}_{i=1}^n$ and the set of context variables, denoted with $\mathcal{C} = \{\mathbf{c}_i\}_{i=1}^m$. The latent distribution is

$$q(\mathbf{z}|\mathcal{A}, \mathcal{C}) \propto p_0(\mathbf{z}) \prod_{i=1}^{n} q(\mathbf{z}|\mathbf{a}_i) \prod_{i=1}^{m} q(\mathbf{z}|\mathbf{c}_i), \tag{5.3}$$

where $p_0(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$, $q(\mathbf{z}|\mathbf{a}_i) = \mathcal{N}(\mathbf{z}|\mu_a(\mathbf{a}_i), \sigma_a^2(\mathbf{a}_i))$, and $q(\mathbf{z}|\mathbf{c}_i) = \mathcal{N}(\mathbf{z}|\mu_c(\mathbf{c}_i), \sigma_c^2(\mathbf{c}_i))$, where $\mu_a, \mu_c, \sigma_a, \sigma_c$ are neural network encodings. In practice, the mean and the variance of the latent distribution are computed by:

$$\mu_z(\mathcal{A}, \mathcal{C}) = \sigma_z^2(\mathcal{A}, \mathcal{C}) \left(\sum_{i=1}^{n} \frac{\mu_a(\mathbf{a}_i)}{\sigma_a^2(\mathbf{a}_i)} + \sum_{i=1}^{m} \frac{\mu_c(\mathbf{c}_i)}{\sigma_c^2(\mathbf{c}_i)} \right),$$

$$\sigma_z^2(\mathcal{A}, \mathcal{C}) = \left(1 + \sum_{i=1}^{n} \sigma_a^2(\mathbf{a}_i)^{-1} + \sum_{i=1}^{m} \sigma_c^2(\mathbf{c}_i)^{-1} \right)^{-1}.$$

Notice that we assume a unique pair of functions (i.e., neural networks) to process the mean μ_c and the variance σ_c^2 associated with the context variables.

5.3 Training the Model via Variational Inference

A challenge with our probabilistic model in (5.2) is the inability to derive a closed-form posterior distribution. We address this by training the model with *variational inference*. Similar to classical variational autoencoders (VAEs) [74], our model is composed of three entities: model likelihood $p(\mathbf{y}|\mathbf{z}, \mathbf{x})$, variational posterior $q(\mathbf{z}|\mathcal{A}, \mathcal{C})$, and prior distribution $p_0(\mathbf{z})$. We rewrite the variational posterior using the reparameterization gradient estimator,

$$\mathbf{z} = \mu_z(\mathcal{A}, \mathcal{C}) + \boldsymbol{\epsilon}\sigma_z(\mathcal{A}, \mathcal{C}) \quad \text{with } \boldsymbol{\epsilon} \sim \mathcal{N}(0, I), \tag{5.4}$$

to allow gradient training. The evidence lower bound of Deep ProMPs is then,

$$\mathcal{L}(\mathcal{A}, \mathcal{C}) = \mathop{\mathbb{E}}_{\mathbf{z}\sim q(\cdot|\mathcal{A},\mathcal{C})}\left[\sum_{i=1}^{n} \log p(\mathbf{y}_i|\mathbf{z}, x_i)\right] - \mathop{\mathbb{E}}_{\mathbf{z}\sim q(\cdot|\mathcal{A},\mathcal{C})}\left[\sum_{i=1}^{n} \frac{\log q(\mathbf{z}|\mathbf{a}_i)}{\log p_0(\mathbf{z})} + \sum_{i=1}^{m} \frac{\log q(\mathbf{z}|\mathbf{c}_i)}{\log p_0(\mathbf{z})}\right],$$

since terms like $\log p_0(\mathbf{z})/\log p_0(\mathbf{z}) = 1$ do not play a role in the optimization process. We minimize the ELBO with respect to the parameters of μ_a, σ_a, μ_c, σ_c, and μ_y.

5.4 Movement Primitive Operations

One of ProMPs' advantages is the ability to use probabilistic operations to generate novel movements. Here, we explain these operations that make ProMPs such a flexible model of movement primitives. Our discussion includes how DeepProMPs achieve these properties. The following sections present experiments that demonstrate DeepProMPs performing these operations.

Via-Points and Context Conditioning: After training a DeepProMP model, new trajectories are predicted by providing a set of via-points $\tilde{\mathcal{A}} \equiv \{\tilde{\mathbf{a}}_i\}$ and a set of external variables $\tilde{\mathcal{C}}$ to the DeepProMP model. Notice that one of the two sets can be empty (if both are empty, our model acts as an unconditioned generative model). We use the variational posterior as in Equation 5.3 to sample $\mathbf{z}$, and then

we use Equation 5.4 to generate the whole trajectory. The contributions of each via-points and external variable can be modulated by weighting their importance. This reweighting can be useful as the size of via-points conditioning and context conditioning is usually unbalanced, and the reweighting can help correct this bias. The variational posterior combining these two components — the conditioning variables and weighting coefficients $\omega_i^A, \omega_i^C \in \mathbb{R}^+$ — has the following formulation,

$$q(\mathbf{z}|\tilde{A}, \tilde{C}, \boldsymbol{\omega}^A, \boldsymbol{\omega}^C) = \prod_{i=1}^{n} q_A(\mathbf{z}|\tilde{\mathbf{a}}_i)^{\omega_i^A} \prod_{i=1}^{m} q_C(\mathbf{z}|\tilde{\mathbf{c}}_i)^{\omega_i^C}, \tag{5.5}$$

obtaining a Gaussian distribution again. Figure 5.1 shows the effects of different types of via-point conditioning.

In our experiments, we found the generated motions can arbitrarily violate the desired via-points. This issue can be addressed by gradient descent on the predicted variational distribution parameters, i.e.,

$$\min_{\tilde{\mu}_z, \tilde{\sigma}_z} \sum_{i=1}^{n} \left\| \mathbf{y}_i - \frac{1}{k} \sum_{j=i}^{k} \mu_y(\mathbf{z}_j, \mathbf{x}_i) \right\|_2^2 + \|\sigma^* - \tilde{\sigma}_z\|_2^2, \tag{5.6}$$

where $\mathbf{z}_{j=1,\dots,k} \sim \mathcal{N}(\tilde{\mu}_z, \tilde{\sigma}_z)$, k is the number of Monte-Carlo samples and σ^* is the target variance. We solve (5.6) by improving the initial solution $\tilde{\mu}_z = \mu_z(\mathcal{A})$ and $\tilde{\sigma}_z = \sigma_z(\mathcal{A})$ via gradient descent (Fig. 5.2). The optimization objective is to minimize the error between the specified via-points and the generated movement distribution and is carried out at deployment time.

Blending: As proposed in ProMPs, the blending of motions consists of smoothly transitioning from one movement to another. Consider a time-varying weight $\omega(\mathbf{x}) \in [0, 1]$ and two movement distributions $q(\mathbf{z}|\mathcal{A}_1, \mathcal{C}_1)$ and $q(\mathbf{z}|\mathcal{A}_2, \mathcal{C}_2)$; we derive the blending of the two movements

$$q_{\mathrm{b}}(\mathbf{z}|\mathcal{A}_1, \mathcal{A}_2, \mathcal{C}_1, \mathcal{C}_2, \mathbf{x}) = q(\mathbf{z}|\mathcal{A}_1, \mathcal{C}_1)^{\omega(x)} q(\mathbf{z}|\mathcal{A}_2, \mathcal{C}_2)^{1-\omega(x)}, \tag{5.7}$$

using the exponential weighting introduced by ProMPs. A potential problem is that the distribution is dependent on time, which could cause jerky motions if $\mathbf{z}$ — the

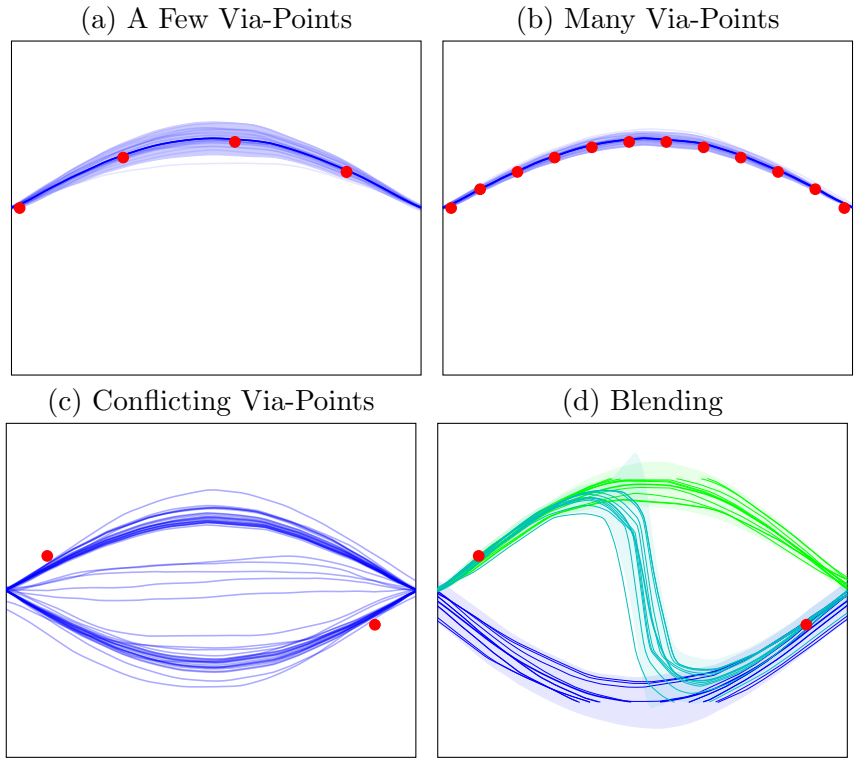


Figure 5.1: DeepProMPs used in different situations. (a) Distribution of trajectories using 3 via-points conditioning. (b) With more via-points, the variance of the distribution decreases. (c) With inconsistent via-points, the model chooses to stay close to the given dataset and violate the via-point constraints in order to avoid unseen behavior. Note the ability of our model to generate a bi-modal distribution. (d) Generalization can be enhanced using blending.

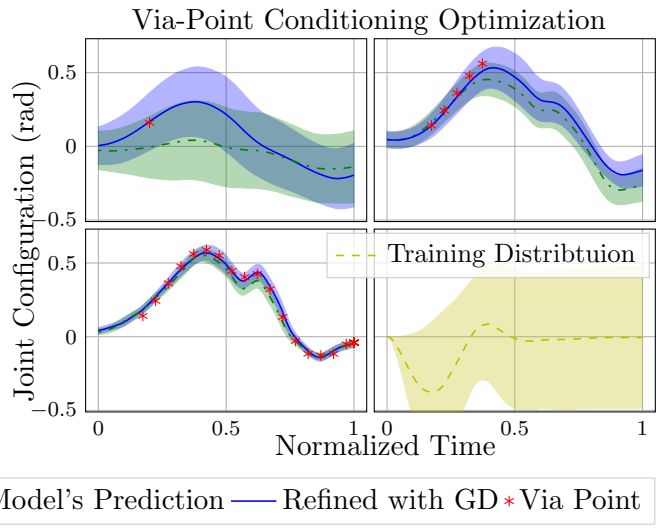


Figure 5.2: Results of the deployment-time via-point error minimization on one of the joints from our real-robot data. Green is our model's predicted distribution; blue is refined with gradient descent. Benefits are more pronounced with fewer via-points, while more via-points improve the overall quality of the prediction. The bottom-right plot shows the distribution of the training data.

latent motion embedding — is sampled at different times leading to jerky action sequences. We address this problem with the the reparameterization distribution by considering the mean $\mu_{\mathrm{b}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{C}_1, \mathcal{C}_2, \mathbf{x})$ and standard deviation $\sigma_{\mathrm{b}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{C}_1, \mathcal{C}_2, \mathbf{x})$ of the blended latent representation of the motion (5.7). By first sampling a standard noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$ and then computing

$$\mathbf{z}(\mathbf{x}) = \mu_{\mathrm{b}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{C}_1, \mathcal{C}_2, \mathbf{x}) + \boldsymbol{\epsilon} \cdot \sigma_{\mathrm{b}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{C}_1, \mathcal{C}_2, \mathbf{x}),$$

where $\cdot$ is the element-wise product, we obtain a smooth transition of $\mathbf{z}$. We refer to Figure 5.1 of DeepProMPs performing blending.

Time Modulation: Time modulation is a crucial component for movement primitives. During training, time is always normalised between 0 and 1; therefore, a time modulation function $\tau(t) : [0, T] \to [0, 1]$ can be used to translate the real-time $t \in [0, T]$ to the *phase* space $\mathbf{x} \in [0, 1]$. In our implementations, we employ the linear time modulation $\tau = t/T$, where T is the movement duration. This linear modulation

provides a straightforward method for achieving proportional velocity profiles with faster or slower movement. A monotonic τ can arbitrarily accelerate or decelerate a movement during its execution. If τ is non-monotonic, it can generate repeated action distributions seen earlier in the movement.

Rhythmic Movements: Rhythmic movements are repeated movement segments over time. Rhythmic movements require these segments to have the same initial and endpoint so that their repetition does not cause jumps. The time-modulation function must have the same position and velocity to obtain this property at $t = 0$ and $t = T$. We achieve this property by using the phase $\mathbf{x} = [\sin(2\pi t/T), \cos(2\pi t/T)]^\intercal$ both during training and deployment.

5.5 Empirical Analysis

The experiments described in this section compare DeepProMPs against alternative motor primitive models. We evaluate each model's reconstruction capabilities on demonstrations provided from several simulation tasks and a real robotic problem. We choose the mean squared error as our metric of choice to highlight how well we can recover demonstrations depending on the conditioning. We also include experiments demonstrating the time-modulation formulation for cyclical tasks on a real robot.

Baselines We choose ProMPs, CNMPs, and VAE-CNMPs as our baselines. We include ProMPs because it is the foundational framework that motivates our work. CNMP can be viewed as a version of our model with fixed variance in the latent distribution, but that predicts output variance $\sigma_y(\mathbf{z}, \mathbf{x})$ as done in previous work [138]. VAE-CNMP is a variation of our model without Bayesian aggregation, the via-point, and context variable independence assumption, but still uses the isotropic Gaussian as the prior in the KL divergence regularization. For both CNMPs and VAE-CNMPs, we use a single encoder that concatenates all inputs (via points and context

variables) when passed to the encoder. We also include variations of the CNMP and VAE-CNMP models trained with zero-padding inputs to replicate the conditioning capabilities of DeepProMPs. For example, to simulate inputting only via-points with our baselines, the variational encoder's input would be $q(\mathbf{z}|\mathbf{a}, \mathbf{c}^0)$, where $\mathbf{c}_0 = \overrightarrow{0}$, or context only as $q(\mathbf{z}|\mathbf{a}^0, \mathbf{c})$ where $\mathbf{a}^0 = \overrightarrow{0}$ for context only inputs. We refer to these versions as CNMPs (Indep) and VAE-CNMPs (Indep) in our results.

Datasets We conduct experiments with demonstrations collected in three simulated tasks from RLBench [63], and two sets of demonstrations performed on physical robotic manipulators. The three simulation tasks we use are: (1) `Reach`, a task where the manipulator goes to a designated target, (2) `Close Box`, a task where the manipulator closes a container, and (3) `Pour Water`, a task where the robot must pick up and pour the water from a container. Our physical robot experiments include demonstrations of a pick-and-place task where a Kinova Gen-3 lite moves a designated object to a specified location. We refer to the Kinova pick-and-place task as the `Kinova` task in this section. We use the previous four tasks (three simulations and one physical robot) to compare reconstruction performance. The second robot task uses a Barrett WAM® Arm to shake a container, which we use to verify training and execution of cyclical tasks. Figure 5.4 show the setup for the simulation tasks and physical robot tasks.

In our reconstruction experiments, we vary the number of trajectories used in each experiment. Our choices in the simulation task were motivated by preliminary experiment results and the complexity of the task. In the physical robotics experiments, we used a kinesthetic demonstration to collect the data. Respectively, we use 500 training examples for the Reach task, 1000 demonstrations for the Close Box and Pour Water task, and 100 demonstrations for the Kinova task. For all simulation tasks, we select the best models using a set of 100 validation examples and report results on 200 test examples. In the Kinova experiments, we report the best validation results

using 10 demonstrations for all models.

Training Configurations For each reconstruction experiment, we train all models to take via-points, low-dimensional context variables, and images as inputs. The dimension of the low-dimensional context varies for each task. In the Reach task, the context is the target location (3 dimensions); in Close Box, the context includes the angle of the box's lid and box location (54 dimensions); for Water Plants, the context is the location of the watering can and object to be watered (84 dimensions). In the Kinova experiments, the context is the normalized pixel location of objects (2 dimensions) and one-hot encoding for the sub-task (pick or place). In the simulation tasks, each trajectory is associated with five images of the scene. We always use these images together at deployment. For our Kinova data, we take a picture of the object to be moved and augment this with three additional pictures that contain markers on the desired object. We found that this mitigated issues with the limited training data used.

5.5.1 Task Reconstruction.

We evaluate ProMPs, DeepProMPs, CNMPs, VAE-CNMPs, CNMP (Indep), and VAE-CNMP (Indep). For optimization, we used the Adam optimizer [73] with all models using a learning rate of 10^{-4}. We train each model for 8,000 epochs in our simulation experiments and 10,000 epochs for our Kinova experiments, while always using mini-batches of size 16.

The network architecture is consistent across all models to ensure fair comparison. All models use two hidden layer multi-layer perceptrons with ReLU activations for vector inputs (via-points and low-dimensional context variables). Each layer has 128 neurons. All models have a single decoder with two hidden layers, each with 128 neurons that take in the latent variable and time-modulation variable as input. For images, we use a Resnet-50 architecture [50] to embed images in the latent space. We

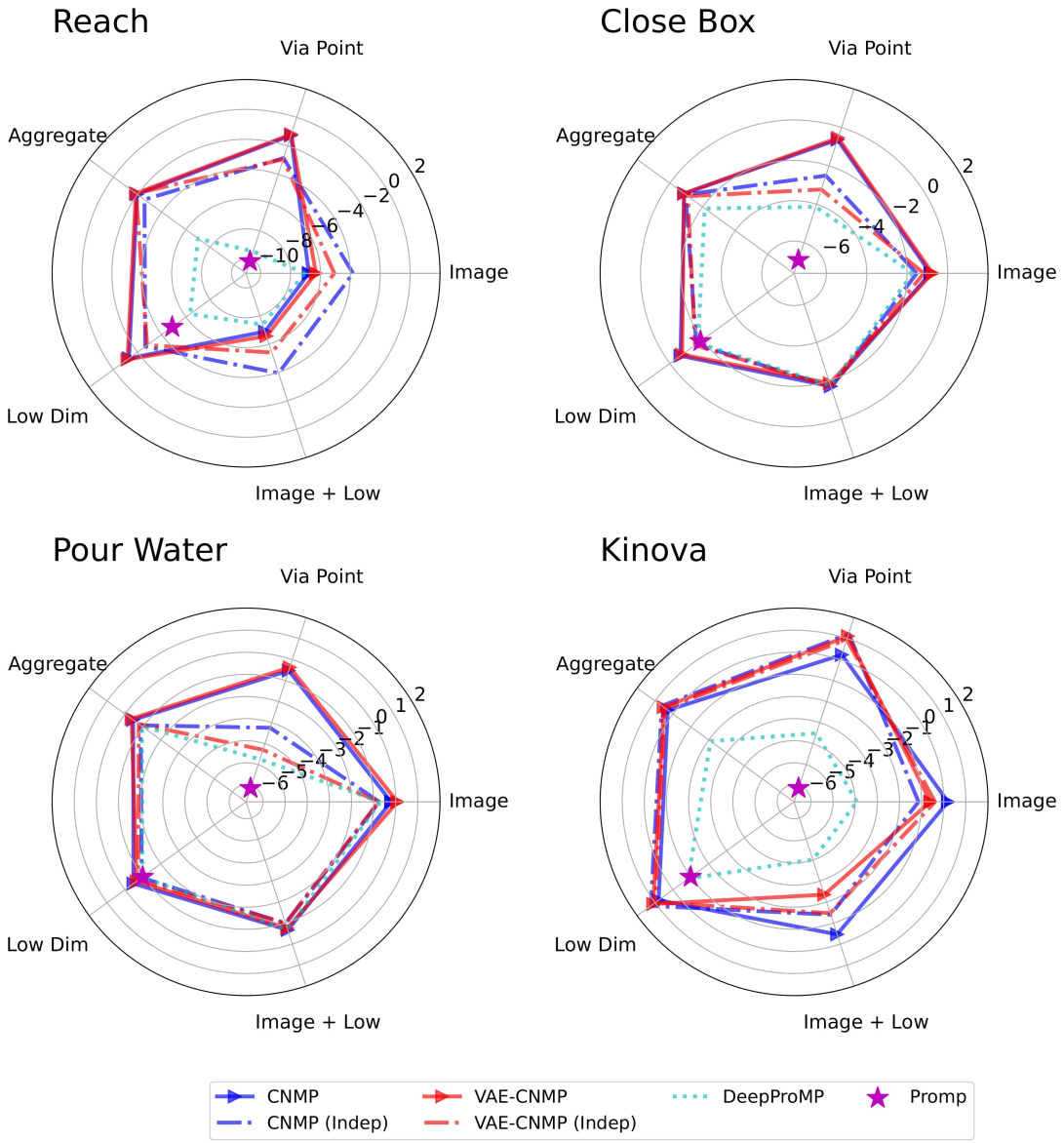


Figure 5.3: Radar plots comparing reconstruction performance across different motor primitive models. Smaller circles are indicative of better performance across potential data types. We consider conditioning on images, low-dimensional context variables, the combination of both, the full trajectory as via points and the average across the four former types of inputs. Measurements are in log scale of the mean square error.

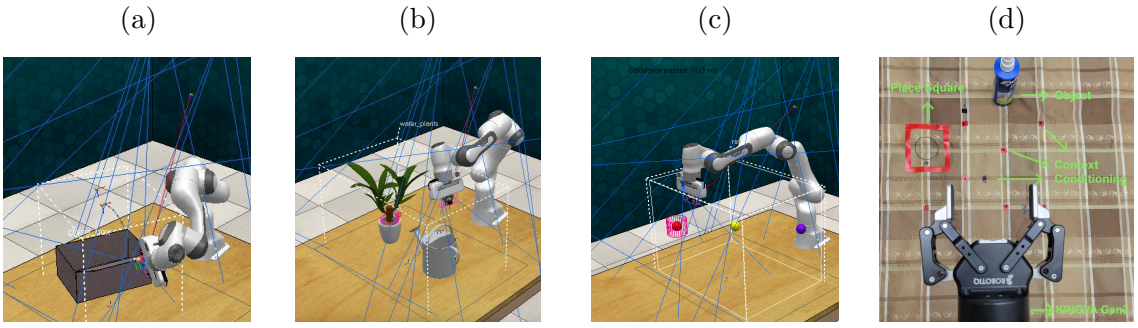


Figure 5.4: (a), (b), (c): Close box, pour water, and reach from RLBench. (d) A top view of our testing setup: The robot should grab the object and place in the designated square. Positions are encoded with 2D context variables.

aggregate the representation for all images associated with a trajectory using a mean operation.

Our training approach varies slightly depending on the model architecture. Both CNMP and VAE-CNMP concatenate all context variable types to each via point as inputs. When using DeepProMP, all encoders share a single affine head to produce latent distribution parameters. We use latent dimensions of 16 for Reach and Kinova and 32 for Close Box and Pour water for all models. During training, we sub-sample combinations of input data for each model. There is an additional sub-sampling step to train models conditioned only via-points, low-dimensional context, and image context information for the DeepProMP, CNMP (Indep), and VAE-CNMP (Indep) models. For example, training with via-points and images on one update and via-points only on another. Both CNMP and VAE-CNMP models always receive a via-point and context variable combination as input because of the concatenation operation. To ensure robust results, we average results using the best performing checkpoints from five independent training runs for each model. The one exception is ProMP, which has a deterministic solution, and we instead bootstrap results to average results.

We report the average performances with radar plots in Figure 5.3. Smaller circles imply better performance across all metrics. We show the log-scale mean squared

error of reconstructing demonstrations given different combinations of inputs. For CNMP and VAE-CNMP variants, we achieve this with zero padding as previously described to mask out the excluded inputs. `Via Point` refers to reconstructing the trajectory without any context variables, `Image` uses only images, `Low Dim` uses only the low dimensional context variable, and `Image + Low` uses both context variables. We also include `Aggregate`, which is the average performance across the previous four combinations of inputs. We note that for each context variable input, we include the initial robot position as well.

In via-point conditioning, we see that ProMP is superior across tasks. This result is expected because ProMP's analytic conditioning preserves all the information in the trajectory. Due to compressing the representation in a latent space, all deep models lose some information when reconstructing the model.

Interestingly, we see that zero-padding training is crucial for improving the baselines' ability to perform via-point conditioning across tasks, but it remains inferior to Deep-ProMPs. The zero-padding results suggests that modularizing encoder representations to data types is better than using a single module. One advantage DeepProMPs has is great capacity with independent encoders for each data type. We trained several models with double of the neurons reported in the experiments for both independent CNMP variants (256 neurons vs. 128 neurons per layer), but these yielded inconsistent performance gains and losses across metrics despite having a comparable number of parameters. This is a sensible result because all models are bottle-necked by the dimensions of the latent space, despite the size of the encoders.

Our results suggest that despite the independence assumptions in DeepProMP, the model provides competitive results with different input combinations and in certain cases provides better performance compared to baselines. We surprisingly find that ProMP produces good results on low-dimension reconstruction across tasks, but it is not as easy to extend to image data. We find that despite the independence assumption, our model still performs comparably to CNMP and VAE-CNMP, which

are trained to always see all context information during training. In certain cases, the zero-pad training seems to worsen performance in scenarios we would not expect for baseline methods (see Reach results using only images). We conclude that even when baselines are trained with techniques similar to DeepProMPs they come with trade-offs across conditioning choices, whereas DeepProMPs do not.

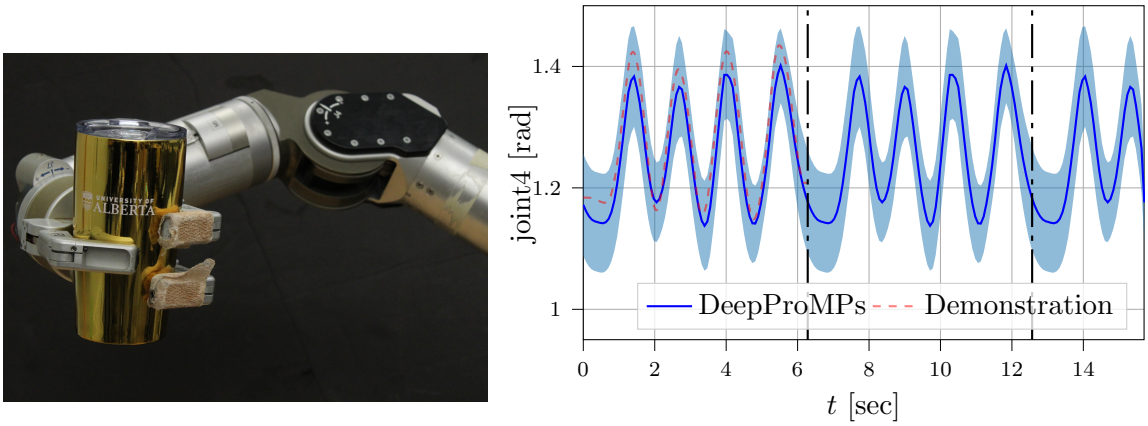


Figure 5.5: **Left**: Experimental setup of `Make Mojito`. **Right**: Demonstrated motion and learned cyclical behaviour. Results demonstrate that DeepProMPS can smoothly repeat cyclical motions indefinitely.

5.5.2 Real-Robot Make Mojito.

This experiment aims to highlight the ability of DeepProMPs to learn and replay cyclical behaviors. To this end, we collect a small data set of cyclical shaking motions and train DeepProMPs using the sinusoidal phase modulation proposed in Section 5.2. In Figure 5.5, we report the most salient dimension of the learned motion, corresponding to the robotic manipulator's second joint. Our model generalized the provided demonstration to perform it in a smooth, cyclic pattern. We find that our model realizes a smooth transition at the end of phase — 2π in Figure 5.5 — even if the gap between the demonstration's starting point and the endpoint is large.

5.6 Conclusions

In this chapter, we proposed Deep Probabilistic Motor Primitives as a deep learning variant of ProMPs. Our model is capable of both via-point and context conditioning

independently. DeepProMPs are more robust to this feature compared to the baseline method trained with zero padding to achieve the same behavior. Our model can also blend motor skills because of the Bayesian aggregation we incorporated in our model. We demonstrated DeepProMPs' capability of rhythmic motion modulation, which has otherwise been ignored in previous works. Our work is a step towards improving deep motor primitive models for applications in robotics. Future work could include incorporating other complex data types in deep motor primitives like text, considering alternative latent distribution choices to the Gaussian distribution in our Bayesian aggregator, and deploying DeepProMPs in downstream robotic applications.

Chapter 6

Morphology Aware Transfer

In this chapter, we discuss our contributions to investigating the efficacy of PEFT algorithms for morphology-aware online RL. Control policies can require immense computation and physical resources to learn for real-world systems (e.g., robotics). If policies can explicitly account for agent morphology, this can reduce computation costs by aggregating knowledge between morphologies and improve the policy's generalization capabilities to new embodiments.

Unfortunately, a morphology-aware policy may not elicit the optimal performance of a target morphology due to these generalization capabilities. For real-world applications, pretrained model components will need to continue to learn to maximize task performance. Reducing learnable parameters matters because, at deployment, computing power for updates may be limited. These limitations make PEFT solutions appealing, since they work under a range of resource constraints at deployment.

6.1 Background

We now formalize the technical framework for our approach. The morphology-aware MDP extends the standard MDP formulation to explicitly represent variations in robot body structure. In this framework, morphology acts as a context that determines the dimensionality of states and actions, allowing us to reason about policies that generalize across different robot designs while maintaining the ability to spe-

cialize to specific configurations. We then review actor-critic methods, particularly proximal policy optimization (PPO), which provide the stable and sample-efficient policy learning necessary for our parameter-efficient fine-tuning approach.

6.1.1 Morphology-Aware Markov Decision Process

The goal in our research is not to learn an optimal policy for a single MDP M but a distribution of MDPs over different robot morphologies. One interpretation of morphology-aware policy learning is as a form of *contextual* Markov decision process (CMDP) [44]. We define a CMDP using a context distribution where $c \sim p(C)$ instantiates a specific MDP. A context represents variations between MDPs that otherwise have overlapping state and action space information. For example, in robotics, this could involve replacing the robot that is performing the desired task.

In morphology-aware learning, c refers to morphology information about the agent, such as the limb adjacency matrix that shows which limbs connect to other limbs, the joints' angles, positions, and velocity, and other dynamics information. Under each context, we have an induced tuple $M(c) = (\mathcal{S}^c, \mathcal{A}^c, p^c(s'|s,a), r^c, p^c(s_0))$. The most important change is that now the objective is to learn a policy that does well over $p(c)$, $\pi^\star(s,c) = \arg\max_{\pi\in\Pi} \mathbb{E}_{p(c)}[G_c]$, where $G_c = \mathbb{E}_{p^c(\tau)}[\sum_{t=0}^{T} \gamma^t r^c(s_t, a_t, s_{t+1})]$ is the expected cumulative reward for a given morphology context information. This work focuses on continuous value action and state spaces where morphology affects the dimension of these variables, $\mathbf{a} \in \mathbb{R}^{l(c)}$ and $\mathbf{s} \in \mathbb{R}^{l(c)\times d}$, where $l(c) \in \mathbb{N}^+$ are the number of limbs in the morphology and $d \in \mathbb{N}^+$ is the feature dimension.

The context c is the morphology information, which we represent as a sequence $\mathbf{c} \in \mathbb{R}^{l(c)\times d^c}$ which has $l(c) \in \mathbb{N}^+$ limbs and $d^c \in \mathbb{N}^+$ features. Each token $\mathbf{c}_i$ contains information such as the limb adjacency matrix, link dynamic values (mass, friction, etc.), and link kinematic information (e.g., joint limits and values). We optimize MDPs with continuous action and state spaces, $\mathbf{a} \in \mathbb{R}^{l(c)}$ and $\mathbf{s} \in \mathbb{R}^{l(c)\times d^s}$ with $d^s \in \mathbb{N}^+$ are state features. This differs from typical CMDPs, which usually assume

a fixed dimensionality of states and actions.

We formalize the parameter efficient fine-tuning problem by first assuming access to a trained policy $\pi(\mathbf{s}, \mathbf{c}\ ; \theta^\star)$ with optimized parameters $\theta^\star$. For a new morphology $\bar{c} \sim p(C)$, we expect this policy to have some initial performance $\mathbb{E}_{\pi(s,\bar{c};\theta^\star)}[G_{\bar{c}}]$, which we call the *zero-shot* policy performance. The goal of our work is to optimize new parameters ϕ to maximize the cumulative reward objective,

$$\phi^\star = \arg\max_{\phi} \mathbb{E}_{\pi(s,\bar{c}\ ;\theta^\star \cup \phi)}[G_{\bar{c}}(s)],$$

where the new parameters are optimized only for the specific morphology $\bar{c}$. We hypothesize that learning a small set ϕ will perform measurably better than the base policy's zero-shot performance, $\mathbb{E}_{\pi(s,\bar{c}\ ;\theta^\star \cup \phi^\star)}[G_{\bar{c}}(s)] > \mathbb{E}_{\pi(s,\bar{c}\ ;\theta^\star)}[G_{\bar{c}}(s)]$ where $|\phi| \ll |\theta|$. We provide details of our approach to address this problem and our experiments in the following sections.

6.1.2 Actor-Critic Methods

One of the major classes of reinforcement learning algorithms is actor-critic methods. These methods optimize a policy π (the actor) guided by one of the previously discussed models of cumulative reward (the critic). Many state-of-the-art RL algorithms use this framework. We outline the general approach here.

The actor is typically updated using the policy gradient update,

$$\theta_{t+1} = \theta_t + \alpha \mathbb{E}_{p(\tau)}[G_t \nabla_\theta \log \pi_\theta(a|s)],$$

where $\alpha \in \mathbb{R}^+$ is the learning rate and θ parameterize the policy. The expectation is difficult to compute because either the transition dynamics $p(s'|s, a)$ are unknown or the distributions have intractable structure for analytical integration. The solution for both cases is to approximate the gradient with Monte Carlo rollouts from environment interactions,

$$\mathbb{E}_{p(\tau)}[G_t \nabla_\theta \log \pi_\theta(a|s)] \approx \frac{1}{n} \sum_{i=1}^{n} G_i \nabla_\theta \log \pi_\theta(a_i|s_i).$$

The trajectories the model learns from are generated by directly interacting with the environment. The policy gradient updates as written above use estimates of return G directly from sampled trajectories. While these returns are unbiased estimates of the true expected return, they can have high variance in practice due to the stochasticity of the environment and the randomness in trajectory outcomes. This high variance makes it difficult to reliably improve the policy.

The solution is to replace the returns G with learned estimates from the critic. The critic is trained (typically using temporal difference methods or regression) to approximate value functions from observed transitions. Using these learned value estimates as control variates, we obtain the actor-critic policy gradient update,

$$\theta_{t+1} = \theta_t + \alpha \frac{1}{n} \sum_{i=1}^{n} [A^{\pi}(s_i, a_i) \nabla_{\theta} \log \pi_{\theta}(a_i|s_i)].$$

We show the update using the advantage function as it is closer to the algorithm we used in our research. The Q-function or value function could also be used, but the advantage function acts as a better baseline, helping to reduce variance and stabilize learning overall. By subtracting the state value $V^{\pi}(s)$ from the Q-function, the advantage centers the gradient estimates around zero, reducing the variance of the policy gradient while introducing only minimal bias.

Proximal Policy Optimization

A key challenge in actor-critic methods is limited sample reuse. In on-policy methods, where we make updates based on the current policy, a single update shifts the policy distribution, invalidating the collected experience. This makes learning sample-inefficient, as new data must be collected after each update. This issue can be mitigated by using trust region methods, which constrain how much the policy can change in each update.

In trust region methods, a KL divergence constraint is included to stabilize learning.

The Trust Region Policy Optimization (TRPO) algorithm [136] formulates this as:

$$\theta^* = \arg\max_{\theta \in \Theta} \mathbb{E}_{s\sim d^{\pi_{\theta_{old}}}, a\sim\pi_{\theta_{old}}(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A^{\pi_{\theta_{old}}}(s,a) \right] \quad \text{s.t. } \bar{D}_{KL}(\pi_{\theta_{old}}, \pi_\theta) \leq \delta, \tag{6.1}$$

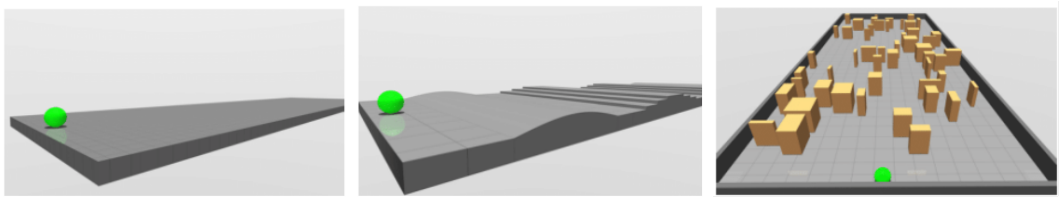
where $\pi_{\theta_{old}}$ is the policy used to collect the trajectories, $d^{\pi_{\theta_{old}}}$ is the state visitation distribution under that policy, and δ is a trust region threshold. The KL divergence constraint ensures the new policy does not deviate too far from the old policy. With a sufficiently conservative threshold δ, the algorithm achieves monotonically non-decreasing returns.

The developers of TRPO [136] derive a lower bound on policy improvement of the form $\frac{L_\pi(\pi')}{1-\gamma} - \frac{4\gamma\epsilon}{(1-\gamma)^2} \max_{s\in\mathcal{S}} D_{KL}(\pi, \pi')$, which shows that limiting KL divergence bounds the potential performance loss. They propose constraining the average KL divergence $\bar{D}_{KL}(\pi, \pi') = \mathbb{E}_{s\sim d^\pi}[D_{KL}(\pi(\cdot|s), \pi'(\cdot|s))] \leq \delta$ as a practical approximation for the max operator in the bound.

However, the constrained optimization objective in TRPO can be computationally intensive, making it challenging to scale the algorithm to large models and complex tasks. Proximal Policy Optimization (PPO) [137] simplifies this objective by replacing the hard KL constraint with a clipping mechanism. PPO optimizes the following clipped surrogate objective:

$$L^{CLIP}(\theta) = \mathbb{E}_{p(\tau)}[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies, $\hat{A}_t$ is the estimated advantage, and ϵ is the clipping hyperparameter (typically 0.1 or 0.2). The clipping operation restricts the ratio to the range $[1-\epsilon, 1+\epsilon]$, and the minimum operation creates a pessimistic bound that prevents destructively large policy updates. This simple modification achieves similar stability to TRPO while being much easier to implement and computationally efficient, making PPO one of the most widely used deep RL algorithms in practice.



(a) Flat Terrain (b) Variable Terrain (c) Obstacle Avoidance

Figure 6.1: Experiments comparing parameter efficient fine-tuning methods were conducted on different terrain types with varying task difficulty. Flat terrain was easiest, allowing the agent to run uninterrupted, while variable terrain and obstacle avoidance presented barriers the agent must navigate. Diagrams reproduced from Gupta et al. [41].

6.2 Minimally Invasive Morphology Adaptation

This research builds on several tools and models proposed in prior research on morphology-aware learning. The significant contribution is investigating means of *reducing* the total number of parameters to adapt a pre-trained policy to new robot embodiments. Transfer learning use weight from a previously trained policy to provide a better initialization. Our work differs from prior research by focusing to transfering policies with limited learnable parameters as opposed to fine-tuning all the parameters. This section discusses important transformer architecture components, the learning framework used, and the variety of fine-tuning techniques evaluated.

6.2.1 Transformers

An essential component of the morphology-aware policies in previous works is transformer models [41, 157, 166]. We treat our data as an observation sequence $\mathbf{o} \in \mathbb{R}^{l(c) \times d}$ with $l(c)$ limb embeddings with $d \in \mathbb{N}^+$ features. Each token $\mathbf{o}_i = [\mathbf{s}_i; \mathbf{c}_i]$ contains limb-level state and context variables of a morphology for $i \in [1, 2, ..., l(c)]$. A morphology-independent linear transformation projects the limb-specific features to a shared embedding space $\bar{\mathbf{o}} = \text{LN}(\mathbf{o}W^{embed} + W^{position}[1 : l])$, where $W^{embed} \in \mathbb{R}^{d \times h}$ is a linear projection operation that transforms the input features to the hidden dimension $h \in \mathbb{N}^+$. $W^{position} \in \mathbb{R}^{L \times h}$ represents the positional embeddings up to some

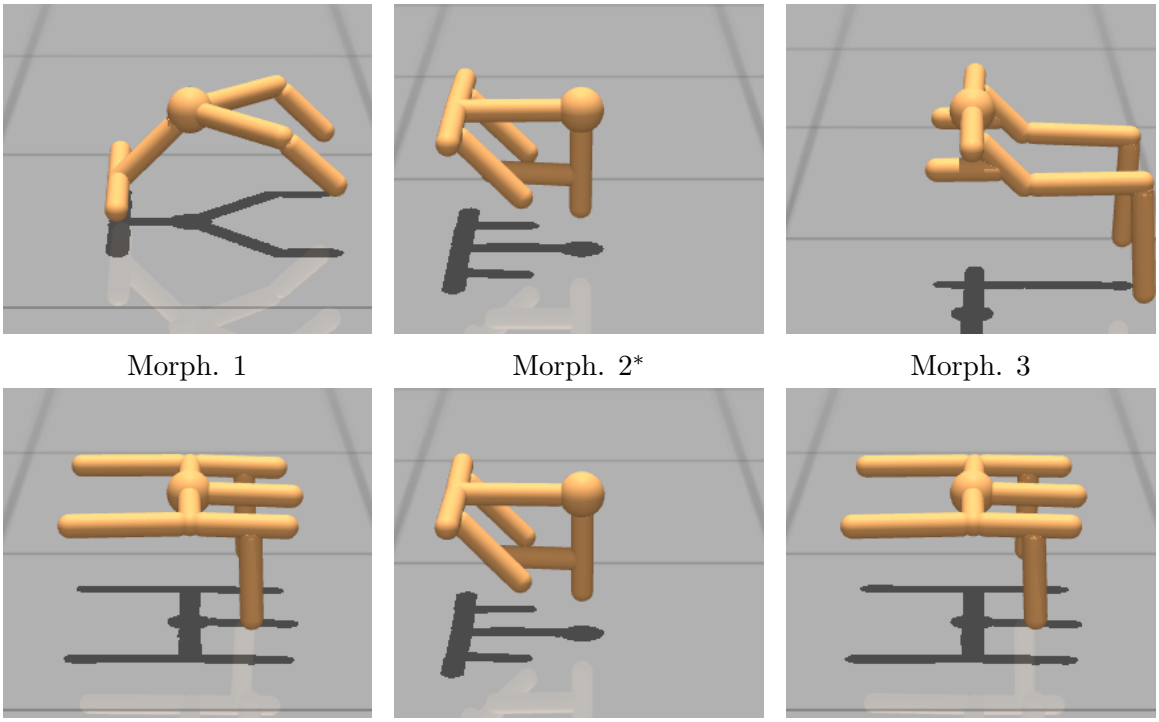


Figure 6.2: The six testing morphologies used in our evaluation. Morphology numbers correspond to those shown in relevant results. Morphologies $\{2, 4\}^*$ and $\{4, 6\}^{**}$ have the same limb configurations but different kinematic and dynamic parameters.

assumed max sequence length $L \in \mathbb{N}^+$, where only the first l columns of $W^{position}$ are used. LN refers to the LayerNorm function [7].

The major component of transformers is the *self-attention* mechanism, which generates weighted combinations of the sequence $\bar{\mathbf{o}}$, $f(\bar{\mathbf{o}}) = \text{softmax}(\epsilon QK^T)V$. We call $Q = \bar{\mathbf{o}}W^Q$, $V = \bar{\mathbf{o}}W^V$, and $K = \bar{\mathbf{o}}W^K$ the query, key, and value, respectively, and $\epsilon = 1/\sqrt{h}$ is a constant chosen to prevent the dot products from causing extremely peaked softmax distributions. The softmax operator, which converts vectors of real numbers to vectors of probabilities, $\text{softmax}(\mathbf{o})_i = \exp(\mathbf{o}_i)/\sum_{j=1}^{l}\exp(o_j)$, defines the weight each vector $\mathbf{o}_i$ contributes. The parameter set $W^{attn} = \{W^Q, W^V, W^K\} \in \{\mathbb{R}^{h\times h}, \mathbb{R}^{h\times h}, \mathbb{R}^{h\times h}\}$ are linear projections. We learn model parameters with gradient descent. Self-attention is followed by a nonlinear transformation function $f(\bar{\mathbf{o}})$ and residual connection to form transformer layer $T_i(\bar{\mathbf{o}}) = W^{out}\sigma(W^{in}(\text{LN}(\bar{\mathbf{o}} + f(\bar{\mathbf{o}}))) +$

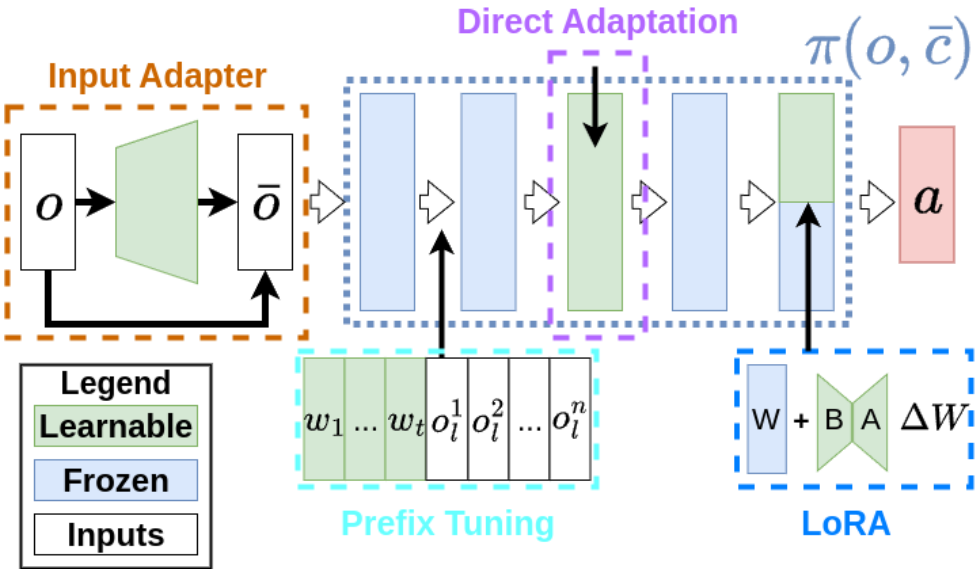


Figure 6.3: A visualization of the various PEFT techniques considered in this paper. We investigate applying PEFT techniques *independently* from each other.

$\text{LN}(f(\bar{\mathbf{o}})) + \bar{\mathbf{o}}$, where $W^{out}, W^{in} \in \mathbb{R}^{h \times h}$, and σ are ReLU functions.

6.2.2 Metamorph Framework

Metamorph is a morphology-aware learning framework that trains a policy over a set of 100 training morphologies.[1] Each morphology c induces an observation sequence $\mathbf{o} = [\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3, ..., \mathbf{o}_{l(c)}]$ *for each* time step. To account for varying $l(c) \in \mathbb{N}^+$ between morphologies, the policy is a transformer as previously discussed. The transformer encoders' hidden representations $\mathbf{h} \in \mathbb{R}^{l(c) \times h}$ with $h \in \mathbb{N}^+$ hidden features per limb. Actions are predicted with a multi-layer perceptron *per limb* as $\mathbf{a}_i = g_\theta(\mathbf{h}_i)$, where $g : \mathcal{H} \to \mathcal{A}$ is a mapping from hidden representations to actions. Here, $\mathbf{a}_i \in \mathbb{R}$ while $\mathbf{h}_i \in \mathbb{R}^h$. Having a token per limb enables a metamorph policy to adapt to varying limb configurations in practice. The policy $\pi_\theta(\mathbf{s}, \mathbf{c}; \theta)$ is optimized over an empirical distribution of morphologies $\mathcal{P}(\hat{C})$ using Proximal Policy Optimization (PPO) [137],

$$L^{CLIP}(\theta) = -\mathbb{E}_{\mathcal{P}(\hat{C})p^c(\tau)}[\min(r_t(c,\theta)\hat{A}_t, \text{clip}(r_t(c,\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)],$$

[1] We explicitly mention training on 100 morphologies because that is done in the original paper. Any number of training morphologies can be used in practice.

Table 6.1: Layer tuning parameters and experiment identifiers

Layer Tuned	**Parameters ϕ**	**Experiment Identifier**
End-to-end	θ^*	E2E
Transformer Layers	$\{T_i; i \in [1, L]\}$	Layer 5
Attention layers	$\{W_i^{\text{attn}}; i \in [1, L]\}$	LoRA
Nonlinear transformers	$\{W_i^{\text{in}}, W_i^{\text{out}}; i \in [1, L]\}$	LoRA
Input Embedding	$\{W^{\text{embed}}, W^{\text{position}}\}$	Embedding
Decoder	$\{W_i^{\text{decoder}}; i \in [1, L^{\text{dec}}]\}$	Decoder

where $r_t(c, \theta) = \frac{\pi_\theta(a_t|s_t,c)}{\pi_{\theta_{old}}(a_t|s_t,c)}$ is the ratio of new to old policy probabilities, $\hat{A}_t$ is the estimated advantage, and ϵ is the clipping hyperparameter.

This framework because it uses transformer-based policies as the morphology-aware policy. Several PEFT techniques considered here are designed specifically for use with transformer models. The framework code is https://github.com/agrimgupta92/metamorphopen-sourced, making it accessible to researchers to reproduce our results and compare other PEFT techniques in potential future work. Several works have also built off this repository to improve the base-architecture design [166, 167].

6.2.3 Parameter Efficient fine-tuning Across Morphologies

PEFT approaches are grouped as either direct, input, or prefix adaptation techniques. *Direct adaptive* PEFT approaches modify some subset of the weights $\phi \subseteq \theta^\star$ or introduce learnable delta weights $\hat{W} = W + \Delta W$. *Input-adaptive* PEFT approaches perform some transformation of the inputs to elicit the optimal performance in the model. Prefix tuning prepends a learnable sequence of tokens to each input sequence. Figure 6.3 visualizes the various types of PEFT algorithms evaluated.

We consider tuning subsets of $\theta^\star$ for direct adaptive PEFT learning. Table C.1 shows the combinations studied and their experiment result identifiers. Several identifiers represent different combinations of direct fine-tuning configurations included in

the experiments. For example, *Input Embedding* includes combinations in which just W^{embed}, W^{position}, and both $\{W^{\text{embed}}, W^{\text{position}}\}$ are tuned online during training, and that *Layer 5* represents directly tuning the all the weights in the final transformer layer.

For attention and nonlinear transformer layers, we used low-lank adapters (LoRA) [57] to learn $\Delta W \in \mathbb{R}^{h^1 \times h^2} = AB$, where $A \in \mathbb{R}^{h^1 \times r}$ and $B \in \mathbb{R}^{r \times h^2}$ are low-rank matrices of rank $r \in \mathbb{N}^+$ to reduce learnable weights for the weight dimensions $h_1 \in \mathbb{N}^+, h_2 \in \mathbb{N}^+$. We initialized LoRA matrix B to small Gaussian noise $b_{ij} \sim N(0; 10^{-4})$. The matrix A is initialized to a zero matrix to presever the zeros-hot performance at the beginning of training. LoRA was included as a fine-tuning method because we want to reduce the total number of parameters used, which LoRA can explicitly do via the rank.

For input-adaptive PEFT approaches, we consider an input adapter layer that modifies the policy observation as $h : \mathbb{R}^{d^c} \to \mathbb{R}^{d^c}$, which modifies the the policy's inputs, $a \sim \pi_{\theta^\star}(h(\mathbf{o}))$. We consider two variations of the function h, where one is a direct nonlinear transform $h(o) = H^{out}\sigma(H^{in}\mathbf{o})$ and a nonlinear transformation with a residual connection $h(\mathbf{o}) = \mathbf{o} + H^{out}\sigma(H^{in}\mathbf{o})$, with learnable weights $\phi = \{H^{in}, H^{out}\}$. We use a hidden layer size of 256 units. The input adapter transforms observations to elicit better performance from a frozen pre-trained model.

Prefix-tuning is a PEFT approach where a set of learnable tokens is pre-pended to the input sequence to elicit desired outputs from the model [82]. These prefixes are a sequence $\phi = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_m]$ of $m \in \mathbb{N}^+$ tokens, where $\mathbf{w}_i \in \mathbb{R}^h$ is a vector. These tokens are then pre-pended to the observations $o^{prefix} = [\phi; \mathbf{o}_1, \mathbf{o}_2, ..., \mathbf{o}_{l(c)}]$ and otherwise processed normally by the transformer layers. Tokens optionally can be pre-pended deeper in the model (e.g., $\mathbf{o}_l^{prefix} = [\phi; T^l(\mathbf{o}^{l-1})]$ for layer $l > 1$) or multiple prefix sets can be used (e.g., $\phi = \{\phi_1, \phi_2, ..., \phi_l\}$ would be learnable prefixes for each layer).

6.3 Experiments

This research aims to evaluate the efficacy of PEFT approaches for online learning on target morphologies. These experiments strive to address the following research questions:

Q1) How effectively do each PEFT learning approach compare with each other and end-to-end fine-tuning?

Q2) What is the relationship between the total number of learnable parameters and the performance when adapting to target morphologies?

Q3) What are the relevant factors for using prefix tuning and LoRA in online reinforcement learning?

These results contribute to understanding the efficacy of these approaches in online learning and can help guide future research developing PEFT algorithms for morphology-aware policy transfer.

These research questions are answered using experiments on a set of tasks previously studied with the Metamorph framework [41]. We use three locomotion tasks that differ in the terrain types shown in Figure 6.1; these include a flat surface, randomized variable terrain, and rectangular obstacle avoidance. The different terrains affect the task difficulty, where the flat terrain is easiest whereas variable terrain and obstacle are more challenging due to obstacles. Each task's reward function emphasizes running as fast as possible to the right. To evaluate the PEFT techniques, we randomly sampled six morphologies from the Metamorph test dataset [41]. We visualize the testing morphologies in Figure 6.2, which include four unique limb configurations and two sets of varying kinematic and dynamic differences. We evaluate PEFT techniques on eighteen environment-morphology combinations.

We pre-trained five models using the Metamorph framework discussed in Section 6.2 with default hyperparameters [41]. Each model was trained on one hundred

training morphologies for ten million time steps for each terrain environment. Each PEFT approach was then applied independently to each of the pre-trained models on the six test morphologies for five million time steps each. Each experiment was repeated on five random seeds for each set of PEFT hyperparameters reported. Each seed used one of the pre-trained models without replacement. We use the same learning hyperparameters for the pre-training phase, except *dropout was not applied* on the transformer embedding. Previous research shows that dropout is helpful for Metamorph pre-training [166], but in preliminary evaluations we found dropout was not helpful for fine-tuning models.

6.3.1 Best Performances Across Methods

In this section, we report results towards answering our first two research questions on the efficacy of different PEFT techniques. We report results in Figure 6.4, which shows the performance of different PEFT techniques. We include results training a policy from scratch to determine whether or not pretraining does help policy transfer. We normalize cumulative rewards after performing parameter-efficient fine-tuning by the pre-trained policy's zero-shot performance, averaging these scores across the six testing morphologies. The x-axis shows the percentage of learnable parameters to the base models original parameter counts. We include the original cumulative reward scores by the best PEFT hyperparameter configuration in Appendix C.3.

Our results reveal a number of notable trends across PEFT approaches. An interesting finding suggests that morphology-pretraining utility is dependent on task complexity. On the flat terrain tasks, learning from scratch is comparable to end-to-end fine-tuning, but performs substantially worse on both variable terrain or obstacle avoidance. Across morphologies, results show that the best input-learnable configurations behave similarly to directly tuning the input Embedding and Decoder, which suggests the two approaches can achieve similar results for the model sizes used in our experiments. We observed substantial performance improvements tuning just the

fifth transformer block, suggesting that if direct model access is possible and a more generous computation budget is available, this layer substantially influences the policy performance. When possible, results suggest that more learning parameters are generally favorable, given the end-to-end fine-tuning results.

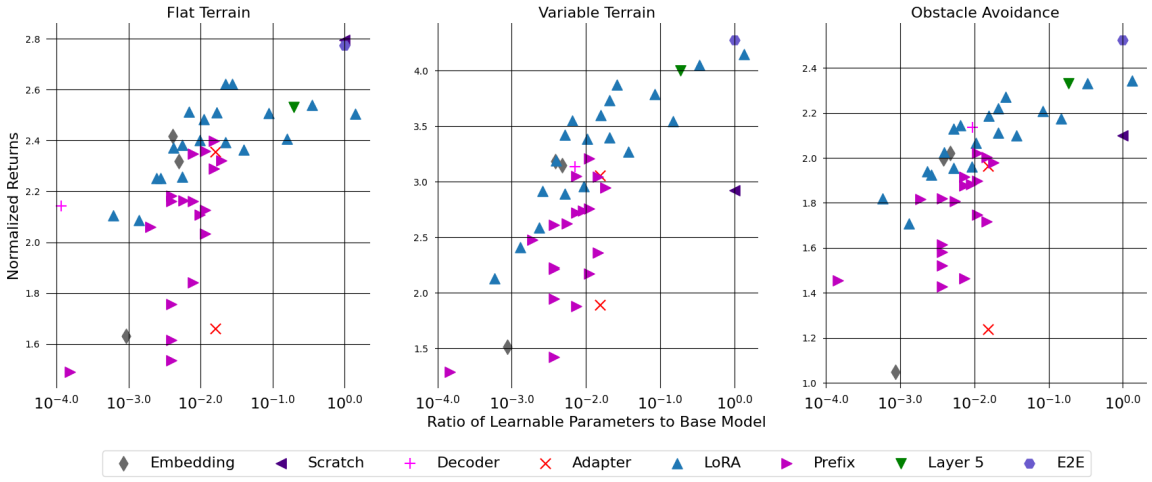


Figure 6.4: Percentage of trainable ratios to total base model parameters vs achieved normalized results. Results suggest total learnable parameters are a contributing factor in final policy performance.

6.3.2 Ablation of LoRA and Prefix Tuning

In this section, we report results comparing different hyperparameter choices for LoRA and Prefix approaches to address our third research question. We include additional results in Appendix C.4. The reported results represent the consistent behaviours observed between the evaluations in each environment. Figure 6.5b shows the results of using LoRA in either the nonlinear transformations (MLP) or attention layer (Attn.) of the fifth transformer layer. The results show that across morphologies, for a single layer's full rank, matrices are necessary. Applying LoRA to the nonlinear transformation is preferable for adaptation to elicit optimal performance, but results suggest that directly tuning a single layer can be better to avoid introducing more learning parameters.

Prefixing tuning results have more nuanced conclusions. We consider three important factors for effective prefix usage: (1) the number of tokens, (2) the injection layer, and (3) comparing token initialization approaches. Each factor has been shown to substantially impact performance [25, 82]. For (3), we propose a second pretraining stage to learn morphology-aware tokens. This second stage repeats the Metamorph training but keeps the base model frozen while learning the tokens.

We generally observe that more learnable parameters are beneficial, such as by increasing the number of tokens used (see Figure 6.5a), which agrees with our other findings previously discussed. In our experiments, a complication with prefix tuning is that introducing untrained tokens can negatively impact policy zero-shot performance. This performance drop can occur because the base model is not trained jointly with the prefix, which initially are noisy observations. This problem is largely overlooked in supervised learning applications because performance is evaluated *after training*.

Unlike supervised learning, performance *during training* is important, especially because its preferred policies have strong initial performance for real-world systems to avoid consequences of poor-performing policies (e.g., damage to the hardware). We conducted experiments adding 50 prefix tokens as input before different transformer blocks to investigate their impact on learning performance. We compared different token initializations, including zero vectors, small Gaussian noise ($N(0; 10^{-4})$), or the pretrained tokens as previously described in this section. We include results when learning from scratch to understand the value of pretraining for sample efficiency. Figure 6.6 shows learning curves.

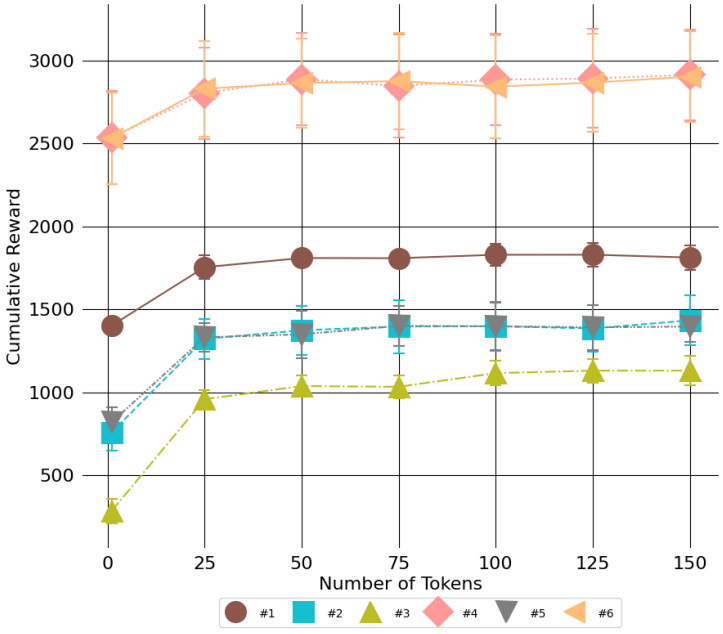
Generally, we observed that the initial zero-shot performance is often negatively affected by zero or random initialization approaches, especially when introducing prefix tokens to the earlier transformer layers. This result suggests that deep layers are less sensitive to the base models' perturbations and better steer feature representations for target morphologies. Interestingly, pre-trained prompting embeddings

significantly improved policy performance during learning compared to other initialization approaches, especially on Morphology number 3, which we found most PEFT approaches struggled to learn. This demonstrates that prefix initialization can mitigate loss in zero-shot performance during fine-tuning in online learning.

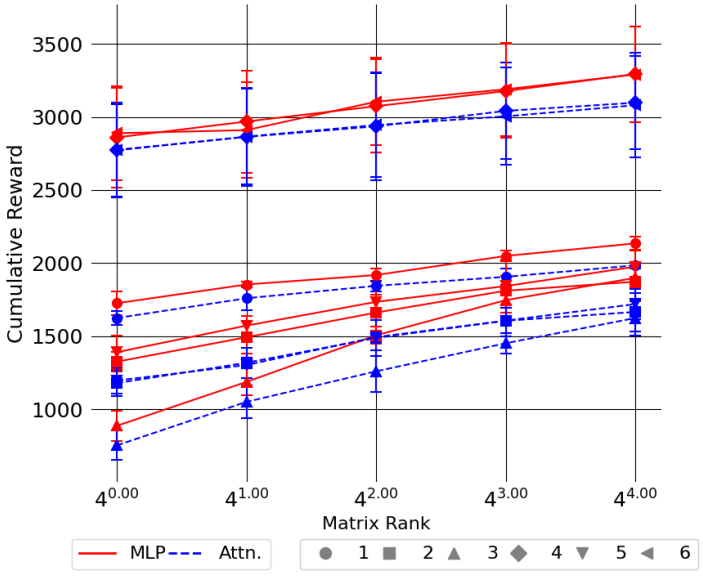
6.4 Conclusions

This chapter discussed a general framework to adapt end-to-end morphology-aware policies to unseen morphologies. Our results show that PEFT techniques can yield substantial improvement in reward with intelligent parameter selection over typical end-to-end fine-tuning. Our ablation experiments show that, particularly for prefix tuning methods, the number of design choices makes substantial impact on performance benefits [119].

Although our results are promising, they have several limitations for use in practical deployment. One concern is that our findings are strictly empirical observations based on collected metrics. We did not produce any profound theoretical results, which is particularly difficult with transformer-based policies. In control systems like robots, it is often helpful to define system properties such as explicit constraints, stability results, or reachable sets of the agent [78].



(a) Number of randomly initialized prefix tokens



(b) LoRA in different layers of fifth transformer block.

Figure 6.5: Ablation studies on prefix tokens and LoRA in variable terrain.

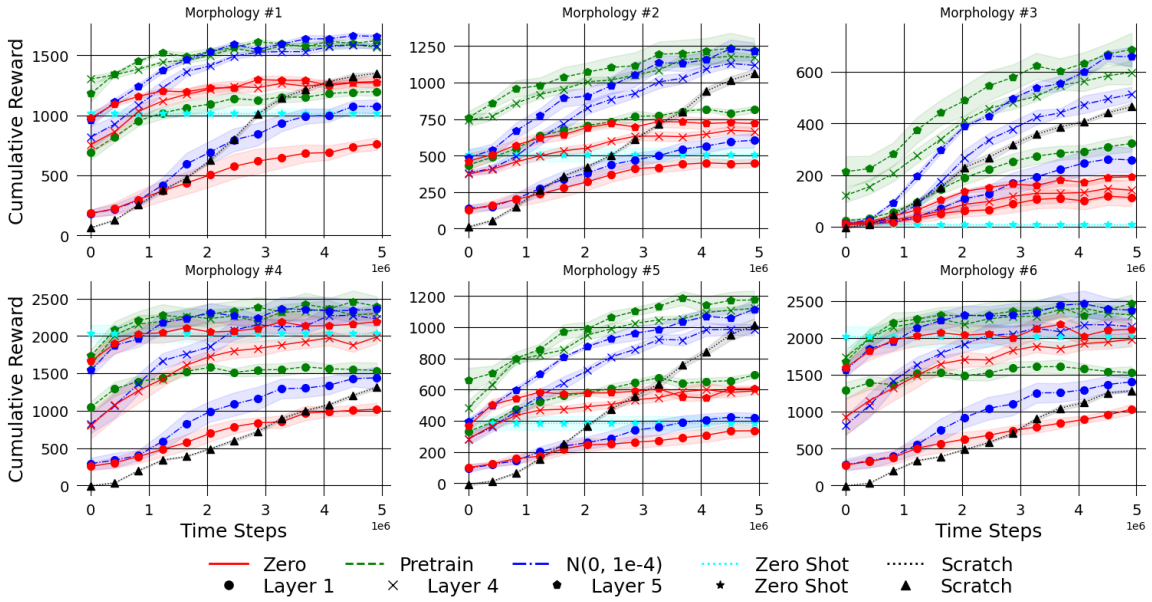


Figure 6.6: Choice of initialization and injection layers of prefix tuning in variable terrain. Initial zero-shot results of E2E learning are plotted to compare affect of prefixes.

Chapter 7

Conclusions

In this dissertation, we proposed several modifications to deep learning models to improve interpretability and generalization across different action-space needs in robotic applications. These models include three robotic domains, including teleoperation, movement primitives, and cross-embodiment learning. Each domain presented unique challenges to improve the deep learning model mappings for learning or generalizing over the action spaces.

7.1 Teleoperation

In Chapter 3, we described our framework for learning action maps for teleoperation settings. We outlined specific properties researchers have proposed for teleoperation, but prior to our work were never formally proven to be enforced. We enforced these properties by proposing neural architecture changes and loss functions to encourage the model to behave as an odd function.

Our theoretical analysis revealed a surprising finding: complex nonlinear models offer no significant advantage over simpler local linear maps for teleoperation tasks. First-order Taylor approximation reveals that nonlinear odd functions behave equivalently to their linear counterparts as discretization steps decrease. This insight suggests that the expressiveness gains from deeper architectures may be unnecessary for teleoperation applications, where local linear maps provide both theoretical guar-

antees and computational efficiency. We use this finding to establish conditions for controllability through small-time local controllability analysis, showing when users can reliably move robots in arbitrary directions. Our controllability results establish a means to identify portions of the robot state space that are small-time locally controllable.

In addition to our controllability results, we showed theoretical results on user action monotonicity, consistency, and reversibility. We showed that model architecture and model first-order approximations lead to monotonic changes in user action at deployment. Consistency can be enforced by bounding the Lipschitz constant of the model. We proved bounds on trajectory reversibility, demonstrating that while perfect reversal is impossible due to discretization errors, bounded soft reversibility can be guaranteed with accuracy improving as sampling rates increase. These three properties help improve the predictability of the interface and can benefit the user experience.

These contributions represent a significant advance in teleoperation research by providing a formal verification that deep learning models can enforce desired teleoperation properties. Our work bridges control theory and machine learning, showing that incorporating mathematical structure through odd function constraints enables both performance and theoretical guarantees. Simpler, more interpretable models may be preferable for assistive robotics, where reliability and predictability are paramount for users with disabilities.

We supported the theoretical analysis of our learning frameworks with a series of user study experiments in Chapter 4. We addressed two primary research questions, including (1) the efficacy of our framework compared to existing teleoperation approaches (both learnable or not) and (2) investigating controllability in the context of mode switching and existing systems. In the former case, we found that our teleoperation-constraint systems improved user experience and performed comparably to non-learning systems. In the latter case, we found mixed results: the benefits

of learned action map mode switching depended on task difficulty.

Our empirical findings strongly validated the theoretical predictions from Chapter 3, particularly demonstrating that odd function behavior is sufficient for effective teleoperation regardless of implementation method. When comparing NOAH architectures, regularization-based approaches, and local linear maps (SCL), we found no statistically significant differences in user performance once models enforced teleoperation properties. This result supports our theoretical equivalence analysis and suggests that the specific technique used to achieve odd function behavior is less important than simply ensuring the property is enforced.

The studies revealed a clear performance hierarchy among different modeling approaches, with our teleoperation property-enforcing methods consistently outperforming conditional autoencoders (CAE) across multiple metrics. SCL models achieved significantly higher success rates, better user satisfaction scores, and lower cognitive workload compared to CAE systems. The poor performance of CAE models was attributed to their violation of basic teleoperation principles. These principles included unwanted movement with zero input and unpredictable behavior when users navigated outside the training distribution, confirming the practical importance of our formalized teleoperation properties.

Perhaps most surprisingly, our controllability experiments reveal a complexity paradox in which providing users with additional control modes degraded performance on certain tasks. Users performed significantly better with single-mode DPC systems compared to three-mode variants, despite reporting similar confidence levels across both conditions. We compared learned action maps to traditional approaches. DPC required significantly fewer mode switches than Cartesian mode switching while achieving comparable success rates. This suggests the benefits of learned mappings emerge primarily through more intuitive control rather than expanded controllability. These findings highlight the importance of balancing system capability with user cognitive load in teleoperation interface design.

An important direction for future research is addressing the data sensitivity and initialization challenges observed in our user studies. Our experiments revealed that learned action maps require careful data collection and validation procedures, with model performance varying significantly based on training data quality and initialization parameters. Future work should investigate more robust training procedures, potentially incorporating techniques from domain adaptation or few-shot learning to reduce the dependency on high-quality demonstration data. Additionally, developing automated methods for validating action map quality before deployment could help ensure consistent user experiences across different tasks and environments.

7.2 Movement Primitives

After studying action representations in teleoperation, we developed a novel movement primitives framework with deep learning systems. Our framework extends previously proposed probabilistic movement primitives, which model motion as a linear-Gaussian model. These models can generalize motions by synthesizing and merging primitives together in a principled way. We incorporated these operations into deep learning models by learning skill latent embeddings to capture motion properties. Even as an imitation learning model, our approach generalized to different input conditioning types. Future work could explore using these movement primitives as modular components, where the controller synthesizes skill embeddings for targeted tasks and modulates them across varying time horizons. Additionally, our models show potential as latent action models in reinforcement learning settings.

A key innovation of our DeepProMPs framework lies in the Bayesian aggregation mechanism, which addresses fundamental limitations of previous neural movement primitive approaches. Unlike mean aggregation methods that treat all conditioning variables equally, our Bayesian approach automatically weights different inputs based on their informativeness and uncertainty. Modeling epistemic uncertainty is essential for handling partial observations and conflicting via-points, allowing the system to

prioritize constraints when generating movements. Our experiments demonstrate this capability across diverse scenarios, from simple via-point conditioning to complex multimodal distributions that emerge naturally from conflicting demonstration data.

The empirical validation across both simulation and real robotic platforms confirms that our theoretical contributions translate to practical improvements. DeepProMPs consistently outperformed baseline methods in reconstruction tasks while maintaining the operational flexibility of classical ProMPs. The framework performed notably better with independent input conditioning compared to concatenation-based approaches. This suggests that the modular encoder design captures meaningful structure in the relationship between different data modalities. Successful deployment on cyclical tasks demonstrates the framework's ability to learn complex temporal patterns. The framework also generalizes beyond the specific demonstrations provided during training.

Our work establishes a foundation for more sophisticated skill composition and transfer learning in robotics. The probabilistic latent space representation enables principled interpolation and extrapolation between learned skills, while the flexible conditioning mechanism supports hierarchical task decomposition. This opens promising avenues for few-shot learning scenarios where new skills can be acquired by combining existing movement primitives with minimal additional demonstrations. The framework's ability to handle diverse input modalities also positions it well for integration with modern vision-language models. This could enable users to control robot movements using natural language commands, making human-robot interaction more intuitive.

7.3 Cross Embodiment Learning

In Chapter 6, we showed it is possible to reduce the learning parameters to fine-tune a policy to target embodiments. We achieved a cumulative reward close to end-to-end fine-tuning with only 10% - 15% of the original model parameters. We further

found that using even fewer parameters without directly modifying the base policy could measurably increase policy performance on average. Our ablation experiments showed challenges in applying these input learnable PEFT techniques to embodied AI: the importance of maintaining initial zero-shot performance during adaptation. When applying prefix-tuning in particular, the weight initialization can severely damage initial pre-trained model performance. Unlike supervised learning applications, where performance is evaluated after training, robotic systems require reliable behavior throughout the learning process to prevent potential hardware damage or safety violations.

Unfortunately, our research has several limitations when considering the results. All experiments were only conducted on locomotion tasks in simulation with dense reward signals. It is unclear how well our results will generalize to other embodiment classes (e.g., manipulators) or reinforcement learning problem variations (e.g., sparse rewards settings). Our findings revealed critical dependencies that limit the practical applicability of PEFT approaches in robotics. Our results showed that the value of pre-training is task-dependent: simple environments showed minimal benefits over learning from scratch, whereas complex terrains showed substantial advantages. This suggests that the computational overhead of maintaining large pre-trained models may not be justified for simpler tasks.

A further limitation is the challenge of interpreting how our policies are making decisions, which can help debug failure cases. In robotic applications, it is often helpful if the system can be *interpreted* in a reliable way particularly as developing safer systems requires understanding the model's decision making. The policies we studied were transformer models, which were trained to directly predict actions given observations. We could make interpretations based on the attention maps, but prior research suggests that conclusions from such analysis can be unreliable in practice. These empirical results, although providing some insights, do not provide the same rigorous support as mathematically grounded results can for real-world deployment.

The lack of theoretical guarantees in our PEFT framework represents a significant limitation for robotic applications. While we achieved promising empirical results, the absence of stability analysis, constraint satisfaction proofs, or reachable set characterizations makes deployment in real-world systems challenging. The black-box nature of transformer-based policies compounds this issue, as understanding why certain morphology adaptations succeed or fail remains opaque. These limitations motivated our subsequent investigation into more interpretable and theoretically grounded approaches in the teleoperation and movement primitives frameworks. In these frameworks, explicit mathematical properties can be proven and verified, providing the reliability and interpretability that practical robotic systems require.

We see several promising directions for improving parameter-efficient fine-tuning of morphology-aware policies. We see several promising directions for improving parameter-efficient fine-tuning of morphology-aware policies. These include incorporating more advanced techniques, such as nonlinear transformations or using morphology information intelligently for policy transfer. Scaling the base model will likely improve transfer performance while reducing total learnable parameters. These directions are promising avenues for future research to extend the results found in this dissertation.

Another direction, motivated by the above limitations of fine-tuning methods, is to build latent action abstractions that generalize over multiple embodiments. These latent abstractions would be able to leverage data collected from multiple robots to generate reusable skills across embodiments with a similar structure. A morphology-aware latent action model could further improve generalizability and benefit from integrating both robotic prior knowledge and embodiment knowledge into the control system.

7.4 Closing Remarks

In this dissertation, we studied techniques to leverage deep learning systems beyond end-to-end settings. We proposed several action abstraction approaches for learning reusable components through imitation learning. These components can be deployed directly as standalone controllers. Alternatively, they can serve as building blocks in larger robot systems. A consistent theme emerges throughout our research: incorporating mathematical structure and inductive biases into neural architectures yields superior performance compared to purely black-box approaches. Our teleoperation systems demonstrate provable properties, our probabilistic movement primitives preserve classical operations, and our morphology transfer methods achieve parameter efficiency. Each of these exemplifies the benefits of structured design.

Our work is a step towards decomposing monolithic learning systems into interpretable, theoretically grounded components that provide both performance and reliability guarantees. Looking forward, these foundations open promising avenues for multi-modal skill conditioning, hierarchical skill composition, and adaptive interfaces that can dynamically adjust to user expertise and task complexity. As robotics evolves toward more general-purpose systems, our principled approaches to human-robot skill transfer will become increasingly essential for ensuring safe, efficient collaboration across diverse real-world applications, with particular potential in assistive robotics and other safety-critical robot learning domains.

Bibliography

[1] Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

[2] Mona Abdel-Keream. Ethical challenges of assistive robotics in the elderly care: Review and reflection. *Robots in care and everyday life*, 121, 2023.

[3] Dmitry Akimov, Vladislav Kurenkov, Alexander Nikulin, Denis Tarasov, and Sergey Kolesnikov. Let offline RL flow: Training conservative agents in the latent space of normalizing flows. In *Offline RL Workshop: Offline RL as a "Launchpad"*, 2022.

[4] Arthur Allshire, Roberto Martín-Martín, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. Laser: Learning a latent action space for efficient reinforcement learning. In *IEEE International Conference on Robotics and Automation*, 2021.

[5] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from Demonstration. *Robotics and autonomous systems*, 57(5), 2009.

[6] Panagiotis K. Artemiadis and Kostas J. Kyriakopoulos. Emg-based control of a robot arm using low-dimensional embeddings. *IEEE Transactions on Robotics*, 26(2), 2010.

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization, 2016.

[8] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, 2017.

[9] Shikhar Bahl, Mustafa Mukadam, Abhinav Gupta, and Deepak Pathak. Neural dynamic policies for end-to-end sensorimotor learning. *Advances in Neural Information Processing Systems*, 2020.

[10] Shikhar Bahl, Abhinav Gupta, and Deepak Pathak. Hierarchical Neural Dynamic Policies. In *Proceedings of Robotics: Science and Systems*, 2021.

[11] Cameron Berg, Vittorio Caggiano, and Vikash Kumar. SAR: generalization of physiological agility and dexterity via synergistic action representation. *Autonomous Robots*, 2024.

[12] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi 0$: A vision-language-action flow model for general robot control. corr, abs/2410.24164, 2024. doi: 10.48550. *arXiv preprint ARXIV.2410.24164*.

[13] Emma Brunskill and Lihong Li. PAC-inspired option discovery in lifelong reinforcement learning. In *International conference on machine learning*. PMLR, 2014.

[14] Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. Myosuite: A contact-rich simulation suite for musculoskeletal motor control. In *Learning for Dynamics and Control Conference*, 2022.

[15] Alexandre Campeau-Lecours, Ulysse Côté-Allard, Dinh-Son Vu, François Routhier, Benoit Gosselin, and Clément Gosselin. Intuitive adaptive orientation control for enhanced human–robot interaction. *IEEE Transactions on Robotics*, 2019.

[16] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, et al. Interpretability of deep learning models: A survey of results. In *2017 IEEE smartworld, ubiquitous intelligence & computing, advanced & trusted computed, scalable computing & communications, cloud & big data computing, Internet of people and smart city innovation (smartworld/SCALCOM/UIC/ATC/CBDcom/IOP/SCI)*. IEEE, 2017.

[17] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for reinforcement learning. In *International conference on machine learning*, 2019.

[18] Tao Chen, Adithyavairavan Murali, and Abhinav Gupta. Hardware conditioned policies for multi-robot transfer learning. In *Advances in Neural Information Processing Systems*, 2018.

[19] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots. In *Proceedings of Robotics: Science and Systems*, 2024.

[20] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2025.

[21] Matei Ciocarlie and Peter Allen. Hand posture subspaces for dexterous robotic grasping. *I. J. Robotic Res.*, 28, 06 2009.

[22] Adrià Colomé and Carme Torras. Dimensionality reduction in learning gaussian mixture models of movement primitives for contextualized action selection and adaptation. *IEEE Robotics and Automation Letters*, 3(4), 2018.

[23] John J Craig. *Introduction to robotics: mechanics and control, 3/E*. Pearson, 2009.

[24] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 1989.

[25] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 2023.

[26] Runyu Ding, Yuzhe Qin, Jiyue Zhu, Chengzhe Jia, Shiqi Yang, Ruihan Yang, Xiaojuan Qi, and Xiaolong Wang. Bunny-visionpro: Real-time bimanual dexterous teleoperation for imitation learning. *arXiv preprint arXiv:2407.03162*, 2024.

[27] A. d'Avella and E. Bizzi. Low dimensionality of supraspinally induced force fields. *National Academy of Sciences*, 1998.

[28] Brian Scott Eberman. *Whole-arm manipulation: kinematics and control*. PhD thesis, Massachusetts Institute of Technology, 1989.

[29] Ann L Edwards, Michael R Dawson, Jacqueline S Hebert, Craig Sherstan, Richard S Sutton, K Ming Chan, and Patrick M Pilarski. Application of real-time machine learning to myoelectric prosthesis control: A case series in adaptive switching. *Prosthetics and orthotics international*, 40(5), 2016.

[30] Julian Eßer, Gabriel B Margolis, Oliver Urbann, Sören Kerner, and Pulkit Agrawal. Action space design in reinforcement learning for robot motor skills. In *Conference on Robot Learning*. PMLR, 2025.

[31] Hongjie Fang, Hao-Shu Fang, Yiming Wang, Jieji Ren, Jingjing Chen, Ruo Zhang, Weiming Wang, and Cewu Lu. Airexo: Low-cost exoskeletons for learning whole-arm manipulation in the wild. In *International Conference on Robotics and Automation*, 2024.

[32] Hiroki Furuta, Yusuke Iwasawa, Yutaka Matsuo, and Shixiang Shane Gu. A system for morphology-task generalization via unified representation and behavior distillation. In *International Conference on Learning Representations*, 2023.

[33] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, 2018.

[34] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[35] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010.

[36] Sebastian Gomez-Gonzalez, Gerhard Neumann, Bernhard Schölkopf, and Jan Peters. Adaptation and robust learning of probabilistic movement primitives. *IEEE Transactions on Robotics*, 2020.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016.

[38] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regulari-

sation of neural networks by enforcing Lipschitz continuity. *Machine Learning*, 2021.

[39] Jérémie Guiochet, Mathilde Machin, and Hélène Waeselynck. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 2017.

[40] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, 2020.

[41] Agrim Gupta, Linxi Fan, Surya Ganguli, and Li Fei-Fei. MetaMorph: Learning universal controllers with transformers. In *International Conference on Learning Representations*, 2022.

[42] David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2022.

[43] Sami Haddadin, Sven Parusel, Lars Johannsmeier, Saskia Golz, Simon Gabl, Florian Walch, Mohamadreza Sabaghian, Christoph Jähne, Lukas Hausperger, and Simon Haddadin. The Franka Emika robot: A reference platform for robotics research and education. *IEEE Robotics & Automation Magazine*, 29 (2), 2022.

[44] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual Markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

[45] Imran Hameed, Xu Chao, David Navarro-Alarcon, and Xingjian Jing. Training dynamic motion primitives using deep reinforcement learning to control a robotic tadpole. In *International Conference on Intelligent Robots and Systems*, 2022.

[46] YiFan Hao, Yang Yang, Junru Song, Wei Peng, Weien Zhou, Tingsong Jiang, and Wen Yao. Heteromorpheus: Universal control based on morphological

heterogeneity modeling. In *International Joint Conference on Neural Networks (IJCNN)*, 2024.

[47] Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (task load index): Results of empirical and theoretical research. In *Human Mental Workload*, volume 52 of *Advances in Psychology*. 1988.

[48] Kaibo He, Chenhui Zuo, Chengtian Ma, and Yanan Sui. DynSyn: dynamical synergistic representation for efficient learning and control in overactuated embodied systems. In *International Conference on Machine Learning*, 2025.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 2015.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[51] Zhanpeng He and Matei Ciocarlie. Discovering synergies for robot manipulation with multi-task reinforcement learning. In *International Conference on Robotics and Automation*, 2022.

[52] Laura V. Herlant, Rachel M. Holladay, and Siddhartha S. Srinivasa. Assistive teleoperation of robot arms via automatic time-optimal mode switching. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2016.

[53] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017.

[54] Sunghoon Hong, Deunsol Yoon, and Kee-Eung Kim. Structure-aware transformer policy for inhomogeneous multi-task reinforcement learning. In *International Conference on Learning Representations*, 2022.

[55] Sarah K Hopko, Riya Khurana, Ranjana K Mehta, and Prabhakar R Pagilla. Effect of cognitive fatigue, operator sex, and robot assistance on task performance metrics, workload, and situation awareness in human-robot collaboration. *IEEE Robotics and Automation Letters*, 6(2), 2021.

[56] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 1989.

[57] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[58] Xiaoyan Hu and Ho-fung Leung. Provably (more) sample-efficient offline RL with options. *Advances in Neural Information Processing Systems*, 36, 2023.

[59] Zhibo Huai, Bo Ding, Huaimin Wang, Mingyang Geng, and Lei Zhang. Towards deep learning on resource-constrained robots: A crowdsourcing approach with model partition. In *IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, 2019.

[60] Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[61] Auke Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning Attractor Land-

scapes for Learning Motor Primitives. *Advances in neural information processing systems*, 15, 2002.

[62] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2), 2013.

[63] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

[64] Jun Jin. Learning geometry from vision for robotic manipulation. *University of Alberta Library*, 2021.

[65] Tor A Johansen and Thor I Fossen. Control allocation—a survey. *Automatica*, 2013.

[66] Ryan Julian, Benjamin Swanson, Gaurav S. Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. In *Conference on Robot Learning*, 2020.

[67] Hong Jun Jeon, Dylan Losey, and Dorsa Sadigh. Shared autonomy with learned latent actions. In *Robotics: Science and Systems*, 2020.

[68] Siddharth Karamcheti, Albert J. Zhai, Dylan P. Losey, and Dorsa Sadigh. Learning visually guided latent actions for assistive teleoperation. In *Proc. Conf Learning for Dynamics and Control*, 2021.

[69] Siddharth Karamcheti, Megha Srivastava, Percy Liang, and Dorsa Sadigh. LILA: Language-informed latent actions. In *Conference on Robot Learning*, 2022.

[70] Kento Kawaharazuka, Jihoon Oh, Jun Yamada, Ingmar Posner, and Yuke Zhu. Vision-language-action models for robotics: A review towards real-world applications. *IEEE Access*, 2025.

[71] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, et al. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024.

[72] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Paul Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. *arXiv preprint*, arXiv:2406.09246, 2024.

[73] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[74] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *International Conference on Learning Representations, ICLR*, 2014.

[75] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. CompILE: Compositional imitation learning and execution. In *International Conference on Machine Learning*, 2019.

[76] Jérémie Kreiss and Jean-François Trégouët. Input redundancy: Definitions, taxonomy, characterizations and application to over-actuated systems. *Systems & Control Letters*, 2021.

[77] Thibaut Kulak, Joao Silvério, and Sylvain Calinon. Fourier movement primitives: an approach for learning rhythmic robot skills from demonstrations. In *Robotics: Science and systems*, 2020.

[78] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[79] Boyu Li, Haoran Li, Yuanheng Zhu, and Dongbin Zhao. MAT: morphological adaptive transformer for universal morphology policy learning. *IEEE Transactions on Cognitive and Developmental Systems*, 16(4), 2024.

[80] Ge Li, Zeqi Jin, Michael Volpp, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. ProDMP: A unified perspective on dynamic and probabilistic movement primitives. *IEEE Robotics and Automation Letters*, 2023.

[81] M. Li, Dylan P. Losey, Jeannette Bohg, and Dorsa Sadigh. Learning user-preferred mappings for intuitive robot control. *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2020.

[82] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*, 2021.

[83] Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[84] Xingyu Lin, John So, Sashwat Mahalingam, Fangchen Liu, and Pieter Abbeel. Spawnnet: Learning generalizable visuomotor skills from pre-trained network. In *International Conference on Robotics and Automation*, 2024.

[85] Zemin Liu, Qingsong Ai, Haojie Liu, Wei Meng, and Quan Liu. Human-like trajectory planning based on postural synergistic kernelized movement primi-

tives for robot-assisted rehabilitation. *IEEE Transactions on Human-Machine Systems*, 54(2), 2024.

[86] Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. TAIL: task-specific adapters for imitation learning with large pretrained models. In *International Conference on Learning Representations*, 2024.

[87] Dylan P. Losey, K. Srinivasan, Ajay Mandlekar, Animesh Garg, and D. Sadigh. Controlling assistive robots with learned latent actions. *IEEE Int. Conf. Robotics and Automation*, 2020.

[88] Dylan P Losey, Hong Jun Jeon, Mengxi Li, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, Jeannette Bohg, and Dorsa Sadigh. Learning latent actions to control assistive robots. *Autonomous robots*, 46(1), 2022.

[89] Kai Lu, Kim Tien Ly, William Hebberd, Kaichen Zhou, Ioannis Havoutis, and Andrew Markham. Learning generalizable manipulation policy with adapter-based parameter fine-tuning. In *International Conference on Intelligent Robots and Systems*, 2024.

[90] Marlos C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2017.

[91] Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption discovery through the deep successor representation. In *International Conference on Learning Representations*. International Conference on Learning Representations, ICLR, 2018.

[92] Timo Malm, Juhani Viitaniemi, Jyrki Latokartano, Salla Lind, Outi Venho-

Ahonen, and Jari Schabel. Safety of interactive robotics—learning from accidents. *International Journal of Social Robotics*, 2010.

[93] Giulia Matrone, Christian Cipriani, Maria Chiara Carrozza, and Giovanni Magenes. Real-time myoelectric control of a multi-fingered hand prosthesis using principal components analysis. *Journal of neuroengineering and rehabilitation*, 9, 06 2012.

[94] Shaunak A. Mehta, Sagar Parekh, and Dylan P. Losey. Learning latent actions without human demonstrations. In *International Conference on Robotics and Automation*, 2022.

[95] Richard K. Miller and Anthony N. Michel. *Ordinary Differential Equations*. Academic Press, 2014. ISBN 9781483259109.

[96] Suvir Mirchandani, Suneel Belkhale, Joey Hejna, Evelyn Choi, Md Sazzad Islam, and Dorsa Sadigh. So you think you can scale up autonomous robot data collection? In *Conference on Robot Learning*. PMLR, 2025.

[97] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3), 2013.

[98] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, 2018.

[99] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3M: a universal visual representation for robot manipulation. In *Conference on Robot Learning*, 2022.

[100] Sabrina M. Neuman, Brian Plancher, Bardienus P. Duisterhof, Srivatsan Krishnan, Colby Banbury, Mark Mazumder, Shvetank Prakash, Jason Jabbour,

Aleksandra Faust, Guido C.H.E. de Croon, and Vijay Janapa Reddi. Tiny robot learning: Challenges and directions for machine learning in resource-constrained robots. In *International Conference on Artificial Intelligence Circuits and Systems*, 2022.

[101] Benjamin A. Newman, Reuben M. Aronson, Siddhartha S. Srinivasa, Kris Kitani, and Henny Admoni. Harmonic: A multimodal dataset of assistive human–robot collaboration. *The International Journal of Robotics Research*, 2022.

[102] Michael Noseworthy, Rohan Paul, Subhro Roy, Daehyung Park, and Nicholas Roy. Task-conditioned variational autoencoders for learning movement primitives. In *Conference on Robot Learning*, 2020.

[103] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems*, 2024.

[104] Aggeliki Odest and Odest Jenkins. 2d subspaces for user-driven robot grasping. *Robotics, Science and Systems Conference: Workshop on Robot Manipulation*, 2007.

[105] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open X-Embodiment: Robotic learning datasets and rt-x models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[106] Xinlei Pan, Tingnan Zhang, Brian Ichter, Aleksandra Faust, Jie Tan, and Sehoon Ha. Zero-shot imitation learning from demonstrations for legged robot

visual navigation. In *International Conference on Robotics and Automation*, 2020.

[107] A. Paraschos, C. Daniel, J. Peters, and G Neumann. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems*, 2013.

[108] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, (3), 2018.

[109] J Hyeon Park, Wonhyuk Choi, Sunpyo Hong, Hoseong Seo, Joonmo Ahn, Changsu Ha, Heungwoo Han, and Junghyun Kwon. Hierarchical action chunking transformer: Learning temporal multimodality from demonstrations with fast imitation behavior. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024.

[110] Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A. Efros. Learning to control self-assembling morphologies: A study of generalization via modularity. In *Advances in Neural Information Processing Systems*, 2019.

[111] Xue Bin Peng and Michiel Van De Panne. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2017.

[112] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In *Advances in Neural Information Processing Systems*, 2019.

[113] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*. PMLR, 2021.

[114] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

[115] Laura Petrich, Jun Jin, Masood Dehghan, and Martin Jagersand. A quantitative analysis of activities of daily living: Insights into improving functional independence with assistive robotics. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[116] Laura Petrich, Jun Jin, Masood Dehghan, and Martin Jagersand. A quantitative analysis of activities of daily living: Insights into improving functional independence with assistive robotics. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[117] Vitchyr Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *Proceedings of International Conference on Machine Learning*, 2020.

[118] Michael Przystupa, Kerrick Johnstonbaugh, Zichen Zhang, et al. Learning state conditioned linear mappings for low-dimensional control of robotic manipulators. In *IEEE International Conference on Robotics and Automation*, 2023.

[119] Michael Przystupa, Hongyao Tang, Glen Berseth, Mariano Phielipp, Santiago Miret, Martin Jägersand, and Matthew E. Taylor. Efficient morphology-aware policy transfer to new embodiments, 2025.

[120] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021.

[121] Ilija Radosavovic, Tete Xiao, Stephen James, Pieter Abbeel, Jitendra Malik, and Trevor Darrell. Real-world robot learning with masked visual pre-training. In *Conference on Robot Learning*, 2022.

[122] Akilesh Rajavenkatanarayanan, Varun Kanal, Konstantinos Tsiakas, James Brady, Diane Calderon, Glenn Wylie, and Fillia Makedon. Towards a robot-based multimodal framework to assess the impact of fatigue on user behavior and performance: a pilot study. In *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, 2019.

[123] Rahul Ramesh, Manan Tomar, and Balaraman Ravindran. Successor options: an option discovery framework for reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019.

[124] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3, 2020.

[125] Arnaud Robert, Ciara Pike-Burke, and Aldo A Faisal. Sample complexity of goal-conditioned hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023.

[126] Sandra Robla-Gómez, Victor M Becerra, José Ramón Llata, Esther Gonzalez-Sarabia, Carlos Torre-Ferrero, and Juan Perez-Oria. Working together: A review on safe human-robot collaboration in industrial environments. *IEEE Access*, 2017.

[127] Francois Routhier and Philippe S. Archambault. Usability of a joystick-controlled six degree-of-freedom robotic manipulator. In *RESNA ANNUAL Conference*, 2010.

[128] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, (6088), 1986.

[129] Marco Santello, Martha Flanders, and John Soechting. Postural hand synergies for tool use. *The Journal of Neuroscience*, 18, 12 1998.

[130] Matteo Saveriano, Fares J Abu-Dakka, Aljaž Kramberger, and Luka Peternel. Dynamic movement primitives in robotics: A tutorial survey. *The International Journal of Robotics Research*, 2023.

[131] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 2009.

[132] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6), 1999.

[133] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*. 2006.

[134] Charles B. Schaff, David Yunis, Ayan Chakrabarti, and Matthew R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *International Conference on Robotics and Automation*, 2019.

[135] Maurice GE Schneiders, MJG Van De Molengraft, and Maarten Steinbuch. Benefits of over-actuation in motion systems. In *Proceedings of the 2004 American control conference*. IEEE, 2004.

[136] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*. PMLR, 2015.

[137] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[138] Muhammet Yunus Seker, Mert Imre, Justus H Piater, and Emre Ugur. Conditional neural movement primitives. In *Robotics: Science and Systems*, 2019.

[139] Carmelo Sferrazza, Dun-Ming Huang, Fangchen Liu, Jongmin Lee, and Pieter Abbeel. Body transformer: Leveraging robot embodiment for policy learning. In *Workshop on Embodiment-Aware Robot Learning*, 2024.

[140] Nur Muhammad Mahi Shafiullah, Anant Rai, Haritheja Etukuru, Yiqian Liu, Ishan Misra, Soumith Chintala, and Lerrel Pinto. On bringing robots home. *arXiv preprint arXiv:2311.16098*, 2023.

[141] Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, 2020.

[142] Mohit Sharma, Arjun Sharma, Nicholas Rhinehart, and Kris M Kitani. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. In *International Conference on Learning Representations*, 2018.

[143] Mohit Sharma, Claudio Fantacci, Yuxiang Zhou, Skanda Koppula, Nicolas Heess, Jon Scholz, and Yusuf Aytar. Lossless adaptation of pretrained vision models for robotic manipulation. In *International Conference on Learning Representations*, 2023.

[144] Abhik Singla, Shounak Bhattacharya, Dhaivat Dholakiya, et al. Realizing learned quadruped locomotion behaviors through kinematic motion primitives. In *International Conference on Robotics and Automation*, 2019.

[145] M. Shelton Smith, Kenneth A. Wallston, and Craig A. Smith. The development

and validation of the perceived health competence scale. *Health Education Research*, 1995. ISSN 0268-1153. doi: 10.1093/her/10.1.51.

[146] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

[147] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*. Springer, 2002.

[148] Yunsick Sung, Jeonghoon Kwak, and Jong Hyuk Park. Graph-based motor primitive generation framework. *Human-centric Computing and Information Sciences*, 2015.

[149] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, The MIT Press, Cambridge, MA, USA, second edition, 2018.

[150] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

[151] H.A. Tijsma, F. Liefhebber, and J.L. Herder. Evaluation of new user interface features for the manus robot arm. In *9th International Conference on Rehabilitation Robotics*, 2005.

[152] Saket Tiwari and Philip S Thomas. Natural option critic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[153] Emanuel Todorov and Michael Jordan. A minimal intervention principle for coordinated movement. In *Advances in Neural Information Processing Systems*, 2002.

[154] Samuele Tosatto, Georgia Chalvatzaki, and Jan Peters. Contextual latent-movements off-policy optimization for robotic manipulation skills. In *IEEE international conference on robotics and automation*, 2021.

[155] William T Townsend and J Kenneth Salisbury. Mechanical design for whole-arm manipulation. In *Robots and Biological Systems: Towards a New Bionics*, 1993.

[156] Brandon Trabucco, Mariano Phielipp, and Glen Berseth. AnyMorph: Learning transferable polices by inferring agent morphology. In *International Conference on Machine Learning*, 2022.

[157] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (Neurips)*, 2017.

[158] Chris Vermillion, Jing Sun, and Ken Butts. Model predictive control allocation for overactuated systems-stability and performance. In *2007 46th IEEE Conference on Decision and Control*. IEEE, 2007.

[159] Chris Vermillion, Jing Sun, and Ken Butts. Model predictive control allocation for overactuated systems - stability and performance. In *IEEE Conference on Decision and Control*, 2007.

[160] Michael Volpp, Fabian Flürenbrock, Lukas Grossberger, Christian Daniel, and Gerhard Neumann. Bayesian Context Aggregation for Neural Processes. In *International Conference on Learning Representations*, 2020.

[161] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[162] Yating Wang, Haoyi Zhu, Mingyu Liu, Jiange Yang, Hao-Shu Fang, and Tong He. Vq-vla: Improving vision-language-action models via scaling vector-quantized action tokenizers. 2025.

[163] Nick Whiteley, Annie Gray, and Patrick Rubin-Delanchy. Statistical exploration of the manifold hypothesis. *Journal of the Royal Statistical Society: Series B*, 2025.

[164] GC Williams, ZR Freedman, EL Deci, and J Leone. Perceived competence scales. *Diabetes Care*, 1998.

[165] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136), 2017.

[166] Zheng Xiong, Jacob Beck, and Shimon Whiteson. Universal morphology control via contextual modulation. In *International Conference on Machine Learning*, 2023.

[167] Zheng Xiong, Risto Vuorio, Jacob Beck, Matthieu Zimmer, Kun Shao, and Shimon Whiteson. Distilling morphology-conditioned hypernetworks for efficient universal morphology control. In *International Conference on Machine Learning*, 2024.

[168] Honghu Xue, Rebecca Herzog, Till M. Berger, Tobias Bäumer, Anne Weissbach, and Elmar Rueckert. Using probabilistic movement primitives in analyzing human motion differences under transcranial current stimulation. *Frontiers in Robotics and AI*, 2021.

[169] Ye Yuan, Yuda Song, Zhengyi Luo, Wen Sun, and Kris M. Kitani. Transform2act: Learning a transform-and-control policy for efficient agent design. In *International Conference on Learning Representations*, 2022.

[170] Shangtong Zhang and Shimon Whiteson. Dac: The double actor-critic architecture for learning options. *Advances in Neural Information Processing Systems*, 2019.

[171] Yi Zhang, Chao Zeng, Jian Zhang, and Chenguang Yang. Wavelet movement primitives: A unified framework for learning discrete and rhythmic movements. *IEEE Robotics and Automation Letters*, 2025.

[172] Chunlin Zhou, Boxing Wang, Qiuguo Zhu, and Jun Wu. An online gait generator for quadruped walking using motor primitives. *International Journal of Advanced Robotic Systems*, 2016.

[173] Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, 2020.

[174] You Zhou, Jianfeng Gao, and Tamim Asfour. Learning via-point movement primitives with inter-and extrapolation capabilities. In *International Conference on Intelligent Robots and Systems*, 2019.

Appendix A: Action Maps for Teleoperation Systems

This section includes the full proofs of theoretical results reported in the thesis. Additional result include corroalry resultson reversibility and controllability. These results were not included in the main thesis as they are derivative proofs under mild assmption changes to the main thesis results.

A.1 Limitations of Nonlinear Odd Functions

Lemma 1 *For two states $x(t)$ and $\hat{x}(t)$, the action $a(t)$ under a Lipschitz continuous, odd, action map f and linearization $\hat{f} = \nabla_a f(\hat{x}(t), a)|_{a=0} a(t)$ have $\|f(x(t), a(t)) - \hat{f}(\hat{x}(t), a(t)\| \leq M + LC$, where $M = \max_a \|f(x(t), a(t))\|$, L is the Lipschitz constant, and $C = \max\{\|a\| : a \in A\}$.*

Proof.

$$
\begin{aligned}
&\|f(x(t), a(t)) - \hat{f}(\hat{x}(t), a(t))\| \\
\leq &\|f(x(t), a(t)\| + \|\hat{f}(\hat{x}(t), a(t))\| \\
= &\|f(x(t), a(t)|| + \|\nabla_a f(\hat{x}(t), a)|_{a=0} a(t)\| \\
\leq &\|f(x(t), a(t)\| + \|\nabla_a f(\hat{x}(t), a)|_{a=0}\| \|a(t)\| \\
\leq &\|f(x(t), a(t)\| + L\|a(t)\| \leq M + L\|a(t)\| \\
\leq &M + LC.
\end{aligned}
$$

■

The intuition of this lemma is that in a robot's state space for a Lipschitz action map with bounded user inputs, there exists some maximal distance of robot velocities between two arbitrary robot states. With this result, we can then prove that as the discretization step $\tau \to 0$, these errors disappear.

Theorem 1 *For an odd action map f that has Lipschitz constant L on a bounded action space, following Euler integration, the distance between trajectory $x(t)$ and $\hat{x}(t)$, where $\hat{X}$ follows linearization $\hat{f}$ of f, we have $\|x(t) - \hat{x}(t)\| \leq \tau(T-1)(M + LC)$, where τ is the discretization step and T is the number of discrete steps taken.*

Proof. Let $f_i = f(x_i, a_i)$, $\hat{f}_i = \hat{f}(\hat{x}_i, a_i)$, $E_i = \|x_i - \hat{x}_i\|$, and $\Delta f_i = \|f_i - \hat{f}_i\|$. It is possible to see that

$$
\begin{aligned}
&\|x_T - \hat{x}_T\| = \|x_{T-1} + \nu f_{T-1} - \hat{x}_{T-1} - \nu \hat{f}_{T-1}\| \\
\leq &\|x_{T-1} - \hat{x}_{T-1}\| + \nu \|f_{T-1} - \hat{f}_{T-1}\| \\
= &E_{T-1} + \nu \Delta f_{T-1} \\
\leq &E_{T-2} + \nu \Delta f_{T-2} + \nu \Delta f_{T-1} \\
\leq &\ldots \leq E_0 + \nu \sum_{i=1}^{T-1} \Delta f_i && \text{(Iterating the inequality above)} \\
\leq &\nu(T-1) \max_{i \in [1, T-1]} \Delta f_i && (E_0 = 0 \text{ since } x(0) = \hat{x}(0)) \\
\leq &\nu(T-1)(M + LC).
\end{aligned}
$$

■

This theorem suggests, particularly for shorter trajectories, users likely would not notice substantial changes in motion between the model's local linear approximation and original prediction.

A.2 Enforcing Action Montonicity

Theorem 2 *A local linear action map $W(\mathbf{x})$ is monotonic in user inputs for any pair of $\alpha_1, \alpha_2 \in \mathbb{R}^+$ with $\alpha_1 < \alpha_2$.*

Proof. Let $\alpha_1, \alpha_2 \in \mathbb{R}^+$ such that $\alpha_1 < \alpha_2$. For action vectors $\mathbf{a}_1 = \alpha_1 \mathbf{a}$ and $\mathbf{a}_2 = \alpha_2 \mathbf{a}$, we have:

$$||W(\mathbf{x})\mathbf{a}_1|| = ||\alpha_1 W(\mathbf{x})\mathbf{a}|| = \alpha_1 ||W(\mathbf{x})\mathbf{a}|| \tag{A.1}$$

$$||W(\mathbf{x})\mathbf{a}_2|| = ||\alpha_2 W(\mathbf{x})\mathbf{a}|| = \alpha_2 ||W(\mathbf{x})\mathbf{a}|| \tag{A.2}$$

Since $\alpha_1 < \alpha_2$ and both are positive:

$$||W(\mathbf{x})\mathbf{a}_1|| = \alpha_1 ||W(\mathbf{x})\mathbf{a}|| < \alpha_2 ||W(\mathbf{x})\mathbf{a}|| = ||W(\mathbf{x})\mathbf{a}_2|| \tag{A.3}$$

■

For some $\alpha \in \mathbb{R}^+$ such that $\hat{\mathbf{a}} = \alpha \mathbf{a}$, we have $W(\mathbf{x})\hat{\mathbf{a}} = \alpha W(\mathbf{x})\mathbf{a}$, directly supporting monotonicity. In the general case, as long as activation functions are monotonic and no bias terms are used, this condition extends to deeper architectures.

Theorem 3 *For a NOAH architecture with monotonic activation functions σ, the complete model maintains monotonicity in user inputs.*

Proof. In a NOAH architecture, the output is given by $\sigma(W(\mathbf{x})\mathbf{a})$, where $W(\mathbf{x})$ is a local linear layer and σ is the activation function.

For $\alpha_1, \alpha_2 \in \mathbb{R}^+$ with $\alpha_1 < \alpha_2$:

- From Theorem 1: $||W(\mathbf{x})(\alpha_1 \mathbf{a})|| \leq ||W(\mathbf{x})(\alpha_2 \mathbf{a})||$
- Since σ is monotonic in norm: $||\sigma(W(\mathbf{x})(\alpha_1 \mathbf{a}))|| \leq ||\sigma(W(\mathbf{x})(\alpha_2 \mathbf{a}))||$

Therefore, the composition of monotonic functions preserves monotonicity, completing the proof.

■

For our regularization objective, we impose a strong structural constraint that leads to favorable monotonicity properties:

Theorem 4 *Suppose we optimize the function $f_\theta(\mathbf{x}, \mathbf{a})$ such that $f(\mathbf{x}, -\mathbf{a}) + f(\mathbf{x}, \mathbf{a}) = 0$. Then f is an odd function in $\mathbf{a}$ and exhibits monotonic behavior in the first-order approximation.*

Proof. The constraint $f(\mathbf{x}, -\mathbf{a}) + f(\mathbf{x}, \mathbf{a}) = 0$ implies $f(\mathbf{x}, -\mathbf{a}) = -f(\mathbf{x}, \mathbf{a})$, so that f odd in $\mathbf{a}$. For odd functions, $f(\mathbf{x}, 0) = 0$ since $f(\mathbf{x}, 0) = -f(\mathbf{x}, -0) = 0$.

The first-order Taylor approximation around $\mathbf{a} = 0$ gives:

$$f(\mathbf{x}, \mathbf{a}) \approx f(\mathbf{x}, 0) + \nabla_{\mathbf{a}} f(\mathbf{x}, 0) \cdot \mathbf{a} = \nabla_{\mathbf{a}} f(\mathbf{x}, 0) \cdot \mathbf{a} \tag{A.4}$$

For this linear approximation $L(\mathbf{a}) = \nabla_{\mathbf{a}} f(\mathbf{x}, 0) \cdot \mathbf{a}$, we have:

$$||L(\alpha \mathbf{a})|| = |\alpha| \cdot ||\nabla_{\mathbf{a}} f(\mathbf{x}, 0) \cdot \mathbf{a}|| \tag{A.5}$$

Therefore, for $0 < \alpha_1 < \alpha_2$:

$$||L(\alpha_1 \mathbf{a})|| = \alpha_1 ||\nabla_{\mathbf{a}} f(\mathbf{x}, 0) \cdot \mathbf{a}|| < \alpha_2 ||\nabla_{\mathbf{a}} f(\mathbf{x}, 0) \cdot \mathbf{a}|| = ||L(\alpha_2 \mathbf{a})|| \tag{A.6}$$

This establishes monotonicity in the first-order approximation.

■

A.3 Proofs for Controllability

This section includes the proofs of theoretical results reported in the main paper. This includes additional results to prove the reported theorems, and additional corollary results.

A.3.1 Teleoperation Controllability

Here, we include the proof our results on identifying state's that are small time-scale locally controllable in the main paper.

Theorem 5 *For an affine control system* $\dot{x} = \sum_{i=0}^{n} \mathbf{h}_i(\mathbf{x}) a_i$, *where* $\mathbf{h}_i = W_i \phi(\mathbf{x})$, *all states* $\mathbf{x}$ *are small-time controllable if for all* $i \in [1, ..., n]$, *we have that* $\phi(\mathbf{x})$ *is not in the null space* W_j *and there does not exist coefficients* $\alpha_j \in \mathbb{R}$ *such that* $\sum_{i \in [1,...,n]/j} \alpha_j W_j = W_1$.

Proof. Suppose not, that is suppose we have W_i such that such that $\phi(\mathbf{x})$ is in the null space. In this case, we have $H(\mathbf{x}) = [W_1\phi(\mathbf{x}); W_2\phi(\mathbf{x}), ..., W_n\phi(\mathbf{x})]$, and that each column where $\phi(\mathbf{x})$ in the null space of W_i we have that $H(\mathbf{x})$ has column $W_i\phi(\mathbf{x}) = \mathbf{0}$. In this case, $H(\mathbf{x})$ does not have full rank because those columns are the $\mathbf{0}$ vector are linearly dependent, which is a contradiction.

Likewise, suppose exists α such that $\sum_{i\in\{1,...,n\}\setminus\{j\}} \alpha_i W_i = W_j$, If this is the case then we have:

$$\sum_{i\in\{1,...,n\}\setminus\{j\}} \alpha_i W_i = W_j \tag{A.7}$$

$$\sum_{i\in\{1,...,n\}\setminus\{j\}} \alpha_i W_i \mathbf{x} = W_j x \tag{A.8}$$

$$\sum_{i\in\{1,...,n\}\setminus\{j\}} (\alpha_i W_i \mathbf{x}) = W_j x \tag{A.9}$$

$$\sum_{i\in\{1,...,n\}\setminus\{j\}} \alpha_i \mathbf{h}_i = \mathbf{h}_j \tag{A.10}$$

which means that basis $\mathbf{h}_j$ is a linear combination of the other basis $\mathbf{h}_i, i \in \{1, ..., n\}/\{j\}$. By definition, means that $H(\mathbf{x})$ is not linear independent. Assuming these conditions are true, the matrix $H(\mathbf{x})$ will be full rank, then by the conditions of small local time controllability in Chapter 15 Equation 15.110 of LaValle [78], the system is small-time controllable in every such state.

■

This result does not necessarily mean that the given state $\mathbf{x}$ is not STLC, which can still be possible in practice due to other conditions for an affine control system to be STLC. Under the additional assumption that the transformation of $\mathbf{x}$ is the identity function, $\phi(\mathbf{x}) = \mathbf{x}$, we have the following corollary when fewer DOFs of control are available $d < n$. An important concept for the below proof is the definition of a lie bracket,

$$[\mathbf{h}_i(\mathbf{x}), \mathbf{h}_j(\mathbf{x})] = \frac{\delta \mathbf{h}_i}{\delta \mathbf{x}} \mathbf{h}_j(\mathbf{x}) - \frac{\delta h_j}{\delta \mathbf{x}} \mathbf{h}_i(\mathbf{x}).$$

For the following corollary, we will utilize this definition as part of the proof. We recommend LaValle [78] Chapter 15 for full details on the mathematics of Lie algebras.

Corollary 5.1 *For an affine control system $H(\mathbf{x}) \in \mathbb{R}^{n \times d}$, if a set of lie brackets exists to form a full rank basis, then all states excluding $\mathbf{x} = 0$ will be STLC.*

Proof. We note that for any $\mathbf{h}_i$ that the Jacobian w.r.t to $\mathbf{x}$ will always be something to this affect: $\frac{\delta \mathbf{h}_i}{\delta \mathbf{x}} = W_i$, so for a first order lie bracket we have that

$$\begin{aligned}
[\mathbf{h}_i(\mathbf{x}), \mathbf{h}_j(\mathbf{x})] &= \frac{\delta \mathbf{h}_j}{\delta \mathbf{x}} \mathbf{h}_i - \frac{\delta \mathbf{h}_i}{\delta \mathbf{x}} \mathbf{h}_j \\
&= W_j W_i \mathbf{x} - W_i W_j \mathbf{x} \\
&= [W_j W_i - W_i W_j] \mathbf{x} \\
&= \tilde{W_{ij}} \mathbf{x},
\end{aligned}$$

where we define $\tilde{W}_{ij} = W_j W_i - W_i W_j$. This structure will arise for every lie-bracket calcualted in the model, even for higher order brackets. Given this recursive structure, if a Lie algebra $\tilde{H}$ of rank n is found, it will have the form

$$\tilde{H}(\mathbf{x}) = [\tilde{\mathbf{h}_1}; \tilde{\mathbf{h}_2}; ...; \tilde{\mathbf{h}_n}] \tag{A.11}$$

$$= [\tilde{W}_1 \mathbf{x}; \tilde{W}_2 \mathbf{x}; ...; \tilde{W}_n \mathbf{x}] \tag{A.12}$$

And Theorem 1 is applicable as before.

■

An important consequence of the above corollary is even in low DOF settings, it's possible that learning linear neural networks as affine control systems can still be STLC. The practical trade-off is navigating the full-state space requires transitioning to near-by states to elicit desired actions, which can be difficult in practice for human operators.

A.3.2 Trajectory Reversibility

In this subsection, we provide proofs for our results on reversibility over trajectories of motions. We include results in continuous time that show it's theoretically possible to perfectly reverse a user's action trajectory despite state conditioning. In our results,

we will assume that $f(\mathbf{x}, \mathbf{a})$ is an odd function $f(\mathbf{x}, -\mathbf{a}) = -f(\mathbf{x}, \mathbf{a})$ and Lipschitz continuous for constant $L \in \mathbb{R}^+$.

Theorem 6 (Trajectory Reversibility) *In the interval* $0 \leq t \leq 1$ *following* f *in Equation 3.1, and assuming that* $\mathbf{a}(t+1) = -\mathbf{a}(t-1)$*, we have* $\mathbf{x}(2-t) = \mathbf{x}(t)$.

Proof. Suppose $d(t) := \mathbf{x}(t+1) - \mathbf{x}(1-t)$, such that $d(0) = \mathbf{x}(1) - \mathbf{x}(1) = 0$. Let $t_0 = \inf_{t \in [0,1]} d(t) \neq 0$ be the first instance $d(t)$ deviates from 0, then $t_0 > 0$ because $d(0) = 0$, and that $d(t) = 0$ for all $t \leq t_0$. Take the derivative w.r.t. t:

$$\begin{aligned}\frac{\partial d(t)}{\partial t} &= \frac{\partial \mathbf{x}(t+1)}{\partial t} + \frac{\partial \mathbf{x}(t-1)}{\partial t} \\ &= f(\mathbf{x}(t+1), \mathbf{a}(t+1)) + f(\mathbf{x}(1-t), \mathbf{a}(1-t)) \\ &= f(\mathbf{x}(1-t), \mathbf{a}(1-t)) - f(\mathbf{x}(t+1), \mathbf{a}(1-t))\end{aligned}$$

where $f(\mathbf{x}(t+1), \mathbf{a}(1+t)) = -f(\mathbf{x}(t+1), \mathbf{a}(1-t))$ because we have $\mathbf{a}(t+1) = -\mathbf{a}(1-t)$. This derivative is zero for $t = t_0$, because $d(t_0) = 0$, or that $d(t_0) = 0$ and $\dot{d}(t_0) = 0$. Moreover, for $d(t) = 0$, we have $\partial d(t)/\partial t = f(\mathbf{x}(t+1), \mathbf{a}(t+1)) + f(\mathbf{x}(1-t), \mathbf{a}(1-t)) = f(\mathbf{x}(t+1), \mathbf{a}(t+1)) - f(\mathbf{x}(t+1), \mathbf{a}(t+1)) = 0 = \dot{0}$.

By the Picard-Lindelof theorem [95] there exists $t > t_0$ such that $d(t) = 0$ is the unique solution of the differential equation for $t > t > t_0$. ■

We now consider results bounding error due to discretization errors in several cases, begining first with a lemma to bound the error between subsequent steps in a trajectory. As a reminder, we denote points along the trajectory as $\mathbf{x}_T(k) = \mathbf{x}(0) + \nu \sum_{i=0}^{k-1} f(\mathbf{x}_T(i), \mathbf{a}(i))$.

Lemma 2 *For two subsequent points* $\mathbf{x}_T(k)$ *and* $\mathbf{x}_T(k+1)$ *following a function* f *with Euler's method, we have that* $||f(\mathbf{x}_T(k), \mathbf{a}) - f(\mathbf{x}_T(k+1), \mathbf{a})|| \leq ML\nu$ *where* $M = \max_{\mathbf{x} \in \mathcal{X}, \mathbf{a} \in \mathcal{A}} ||f(\mathbf{x}, \mathbf{a})||$

Proof.

$$\begin{aligned} ||f(\mathbf{x}_T(k), \mathbf{a}) - f(\mathbf{x}_T(k+1, \mathbf{a})|| &\leq L||\mathbf{x}_T(k) - \mathbf{x}_T(k+1)|| \\ &= L||\nu f(\mathbf{x}(k), \mathbf{a})|| \\ &\leq ML\nu \end{aligned}$$

■

Theorem 7 (Trajectory Soft Reversibility) *Let points along our trajectory be* $\mathbf{x}_T(k) = \mathbf{x}(0) + \nu \sum_{i=0}^{k-1} f(\mathbf{x}_T(i), \mathbf{a}(i))$ *with* $\nu \in \mathbb{R}^+$ *for a fixed duration* $T \in \mathbb{N}^+$. *If* $f(\mathbf{x}(t), \mathbf{a}(t))$ *is a odd, bounded, and Lipschitz continuous function* $\|f(\mathbf{x}(t), \mathbf{a}(t)) - f(\mathbf{x}'(t), \mathbf{a}(t))\| \leq L\|\mathbf{x}(t) - \mathbf{x}'(t)\|$, *and a user performs an action sequence such that* $\mathbf{a}_T(T+k) = -\mathbf{a}_T(T-k)$ *at each step during control, with* $\mathbf{a}_T(i) = \mathbf{a}(i/T)$, *Then we have,*

$$\|\mathbf{x}_T(0) - \mathbf{x}_T(2T)\| \leq \nu M \left(\exp\left(TL\nu\right) - 1\right). \tag{A.13}$$

where $M = \max_{\mathbf{x}\in\mathcal{X}, \mathbf{a}\in\mathcal{A}} \|f(\mathbf{x}, \mathbf{a})\|$.

Proof. Our proof is similar to error bound in Euler's method but exploits knowledge about the true function. Actions are such that up to T we have $\mathbf{a}_T(i) = \mathbf{a}(i/T) = \mathbf{a}$ for a fixed action, and after T we have that $\mathbf{a}_T(T+k) = -\mathbf{a}_T(T-k)$. We denote state error $E_t = ||\mathbf{x}_T(T+t) - \mathbf{x}_T(T-t)||$ and velocity error $\Delta f_t = ||f(\mathbf{x}_T(t+1), \mathbf{a}(t)) - f(\mathbf{x}_T(t), \mathbf{a}(t))||$. Using these, we then derive the error $E_i = ||\mathbf{x}_T(T-i) - \mathbf{x}_T(T+i)||$

for $T \in \mathcal{N}^+$.

$$
\begin{aligned}
E_T &= ||\mathbf{x}_T(0) - \mathbf{x}_T(2T)|| \\
&\leq ||\mathbf{x}_T(1) - \nu f(\mathbf{x}_T(1), \mathbf{a}(1)) - \mathbf{x}_T(2T-1) \\
&\quad + \nu f(\mathbf{x}_T(2T-1), a_T(1))|| + \nu \Delta f_0 \\
&\leq E_{T-1} + \nu ||f(\mathbf{x}_T(2T-1), a_T(1)) - f(\mathbf{x}_T(1), a_T(1))|| \\
&+ \nu \Delta f_0 \\
&\leq E_{T-1} + L\nu ||\mathbf{x}_T(2T-1) - \mathbf{x}_T(1)|| + \nu \Delta f_0 \\
&= E_{T-1}(1 + L\nu) + \nu \Delta f_0 \\
&\leq E_{T-1}(1 + L\nu) + ML\nu^2
\end{aligned}
$$

Where we use the previous lemma to bound subsequent steps. If we define $c = 1 + L\nu$ and $G = MLv^2$, then recursively applying the definition of E_t leads to the following:

$$
\begin{aligned}
cE_{T-1} + G &\leq c(cE_{T-2} + G) + G \\
&= c^2 E_{T-2} + (cG + G) \\
&\leq c^2[cE_{T-3} + G] + (cG + G) \\
&= c^3 E_{T-3} + c^2 G + cG + G \\
&\leq ... \\
E_T &\leq (c)^{T-1} E_0 + G \sum_{i=0}^{T-1} c^i,
\end{aligned}
$$

and replacing definitions of c and G:

$$
E_T \leq (1 + L\nu)^{T-1} E_0 + ML\nu^2 \sum_{i=0}^{T-1} (1 + L\nu)^i.
$$

Note that $E_0 = 0$ because the optimal inverse solution and approximation start at same $\mathbf{x}_T(T)$ position. Applying the geometric series on the second term we end up

with:

$$\begin{aligned} E_T &\leq ML\nu^2(\frac{(1+L\nu)^T-1}{L\nu}) \\ &= M\nu[(1+L\nu)^T-1] \\ &\leq \nu M[\exp{(TL\nu)}-1] \end{aligned}$$

■

As discussed in this paper, this bound shows that as $\nu \to 0$ the error converges to 0. The previous proof assumes that the learned function f is exactly reversible. In certain cases, it could be the case that the action map is only approximately odd $f(\mathbf{x}, -\mathbf{a}) \approx -f\mathbf{x}, \mathbf{a}) + \epsilon$ for some error $||\epsilon|| < E$, with maximum error $E \in \mathcal{R}^+$. To account for this, we derive a reversibility bound assuming this residual quantity. We do not include this result in our paper as driftless affine control systems are exactly reversible by design. The use of this corollary is for action maps, such as multi-layer perceptrons, that can approximate odd function behavior which is crucial for reversibility [118].

Corollary 7.1 (Residual Inversing Actions) *Suppose that $f(\mathbf{x}, \mathbf{a})$ is such that $f(\mathbf{x}, -\mathbf{a}) = -f(\mathbf{x}, \mathbf{a}) + \epsilon$ is some bounded residual error $||\epsilon|| \leq E$ with Lipschitz constant $L \in \mathbb{R}^+$.*

$$||\mathbf{x}_T(0) - \mathbf{x}_T(2T)|| \leq (\nu M + \frac{E}{L})\exp{(TL\nu)} - 1] \tag{A.14}$$

Proof. The proof follows the same steps as the previous proof, but we have that: $\Delta f_t = ||f(\mathbf{x}_T(t+1), \mathbf{a}(t) - f(\mathbf{x}_T(t), \mathbf{a}(t)|| + ||\epsilon(t)||$. where $\epsilon(t)$ is the error for taking $-\mathbf{a}(t)$. In the previous proof then would use the following relationship:

$$\nu\Delta f_t \leq ML\nu^2 + \nu E$$

Applying the above bound after telescoping we would end up with the following:

$$\begin{aligned} E_T &\leq (ML\nu^2 + E\nu)(\frac{(1+L\nu)^T - 1}{L\nu}) \\ &= (M\nu + \frac{E}{L})[(1+L\nu)^T - 1] \\ &\leq (M\nu + \frac{E}{L})[\exp(TL\nu) - 1] \end{aligned}$$

■

At least in our analysis, this provides insight that a typical end-to-end encoder is inappropriate for teleoperation settings. By designing the action map to be an odd function, we mitigate the needs of additional data collection to achieve reversibility without any errors in the odd function behavior.

Appendix B: Teleoperation User Studies

This appendix chapter provides additional information for the user study results reported in Chapter 4.

B.1 Controllability User Study Additional Results

B.1.1 Model Training Details

Across all experiments we use the same hyperparameters for training deep principle component and state-conditioned linear action maps. Neural networks (all decoders, and encoders where relevant) are trained with two hidden layers with 256 units each, and tanh activation functions. Models are optimized with the Adam optimizer with a 10^{-4} learning rate and otherwise default optimizer hyperparameters. We constraint network Lipschitz constant to $L = 1.0$ using Algorithm 2 from Gouk et al. [38]. For all datasets we use mini-batches of 256. For user studies we train DPC for 2000 epochs, training DPC with $\sigma = 0.01$. For simulation experiments we train models for 5000 epochs. We used the respective total tuples (aggregated across trajectories) for each task: `Barrier` 6580 training tuples, `Shelf` 4058 training tuples, and `Pour` 46868 training tuples.

As part of our analysis, we conducted an extensive number of user studies to verify the efficacy of DPC action maps with multiple control modes. Our results included questionnaires on perceived competence (Table B.1) and user experience with Liker scale questions (Table B.2). We provide tables with full results and statistical analysis

per task.

For the `Barrier` task, we report perceived competence results in Table B.3 and user trial statistics are reported in Table B.4. We did not collect the NASA-TLX or Likert questionnaire responses for the `Barrier` task. We also include results providing participants 2 control modes in these results.

For the shelf task we had three separate cohorts of participants comparing DPC to either a single-mode variant, Cartesian mode-switching, or a global PCA model. We report the Likert scale questionnaire results comparing against the one-mode baseline in Table B.8, Cartesian mode-switching in Table B.9, and PCA in Table B.10. Perceived competence results comparing the one-mode baseline are in Table B.5, Cartesian mode-switching in Table B.7, and PCA in Table B.6. We include the Nasa TLX summary statistics in Table B.11. User Trial statistics are shown for one-mode baselines in Table B.8, Cartesian baseline in Table B.14.

For the pouring task, we had two cohorts comparing DPC against Cartesian mode-switching and a single global PCA action map. Perceived Competence results compared against PCA are in Table B.15 and Cartesian are in Table B.20. Likert questionnaire results are in Table B.16 for PCA baseline and Table B.20. Nasa TLX results are shown in Table B.17. User trial results are included for Cartesian mode-switching in Table B.19 and PCA baseline in Table B.18.

Table B.1: Perceived Competence Questions

Survey Question	**Short Label**
I feel confident in my ability to use the teleoperation mapping function.	User Confidence
I am capable of controlling the robot to complete the task.	Task Completion
I am able to move the robot to desired locations.	Robot Navigation
I am independently able to control the robot.	User Autonomy

Table B.2: Likert Scale Questionaire for user studies. We include the corresponding label used in other results as part of this table.

Survey Question	**Short Label**
The robot went where I wanted it to go most of the time.	Accuracy
I found the robot's behavior was predictable.	Predictability
How easy was it to undo mistakes with the system?	Recoverability
I was able to adapt to the robot's behavior during control.	Adaptability
How in control of the system did you feel?	Control
How intuitive was the robot to control?	Intuitiveness
How reliable did you find the behavior of the robot while using it?	Reliability
The joystick mapping made controlling the robot easier.	Mapping
How precise were the commands with the teleoperation mapping?	Precision
How natural did you find controlling the robot?	Naturalness

Table B.3: Summary Statistics of Perceived Competence Metrics in `Barrier`. Superscripts indicate statistical significance ($p < 0.05$), where 1$\widehat{=}$Mode 1, 2$\widehat{=}$Mode 2, 3$\widehat{=}$Mode 3, and A if there was significance compared to all other groups. We used a Mann-Whitney test, to test for significance, since none were normally distributed.

Group	Metric	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
1 Mode	User Confidence^{A}	4.727	5.000	1.728	1.000	7.000	3.000	6.000	33
	Task Completion^{2}	5.061	6.000	1.890	1.000	7.000	3.000	7.000	33
	Robot Navigation^{2}	4.545	5.000	1.742	1.000	7.000	3.000	6.000	33
	User Autonomy^{2}	4.909	5.000	1.764	1.000	7.000	4.000	6.000	33
	Perceived Competence^{A}	4.811	5.250	1.617	1.250	7.000	3.000	6.250	33
2 Mode	User Confidence^{1}	3.515	3.000	1.617	1.000	6.000	3.000	5.000	33
	Task Completion^{1}	3.788	4.000	1.647	1.000	6.000	3.000	5.000	33
	Robot Navigation^{1}	3.576	4.000	1.478	1.000	6.000	2.000	5.000	33
	User Autonomy^{1}	4.000	4.000	1.633	1.000	7.000	3.000	5.000	33
	Perceived Competence^{1}	3.720	3.750	1.425	1.000	6.000	2.750	5.000	33
DPC	User Confidence^{1}	3.515	4.000	1.654	1.000	7.000	2.000	5.000	33
	Task Completion	4.364	5.000	1.967	1.000	7.000	3.000	6.000	33
	Robot Navigation	3.970	4.000	1.678	1.000	6.000	3.000	5.000	33
	User Autonomy	4.333	5.000	1.837	1.000	7.000	3.000	6.000	33
	Perceived Competence^{1}	4.045	4.500	1.632	1.000	6.750	3.000	5.250	33

Table B.4: Statistics for 1 Mode, 2 Mode and DPC for `Barrier`. Superscripts indicate statistical significance ($p < 0.05$), where 1≙Mode 1, 2≙Mode 2, 3≙Mode 3, and A if there was significance compared to all other groups. We used a Mann-Whitney test, to test for significance, since none were normally distributed.

Metric	1 Mode	2 Mode	DPC
Time (Mean ± Std)	108.22 ± 55.58^{A}	153.21 ± 41.58^{A}	141.39 ± 47.07^{A}
Mode Switches (Mean ± Std)	-^{A}	11.81 ± 6.78^{1}	11.28 ± 8.15^{1}
Resets Involuntary (Mean ± Std)	0.08 ± 0.27^{2}	0.21 ± 0.48^{1}	0.13 ± 0.40
Resets Voluntary (Mean ± Std)	0.33 ± 0.61^{2}	0.52 ± 0.66^{1}	0.40 ± 0.65
Resets Total (Mean ± Std)	0.41 ± 0.67^{2}	0.73 ± 0.87^{1}	0.54 ± 0.77
Mode 1 Time Spent (Mean ± Std)	108.22 ± 55.58^{3}	89.44 ± 30.72^{3}	79.50 ± 33.85^{A}
Mode 2 Time Spent (Mean ± Std)	-^{A}	63.77 ± 34.21^{A}	33.37 ± 24.83^{A}
Mode 3 Time Spent (Mean ± Std)	-^{3}	-^{3}	28.52 ± 25.44^{A}
Success Rate	0.72^{A}	0.39^{A}	0.54^{A}
Joystick Smoothness(Mean ± Std)	-42.03 ± 12.72^{A}	-53.67 ± 12.35^{1}	-52.39 ± 11.92^{1}
Robot Smoothness (Mean ± Std)	-10.89 ± 3.63^{A}	-15.84 ± 4.41^{1}	-15.07 ± 4.49^{1}

Table B.5: Summary Statistics of Perceived Competence Metrics in `Shelf` compared to 1 mode baseline. We used a Mann-Whitney test, to test for significance, since none were normally distributed, but found no statistically significant results.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
User Confidence	1 Mode	5.800	6.000	1.536	2.000	7.000	6.000	7.000	10
	DPC	5.700	6.000	1.269	3.000	7.000	5.250	6.750	10
Task Completion	1 Mode	5.300	5.500	1.616	1.000	7.000	5.000	6.000	10
	DPC	6.300	6.500	0.781	5.000	7.000	6.000	7.000	10
Robot Navigation	1 Mode	5.100	6.000	1.513	2.000	7.000	4.250	6.000	10
	DPC	5.400	5.500	1.020	3.000	7.000	5.000	6.000	10
User Autonomy	1 Mode	5.600	6.000	1.428	2.000	7.000	5.000	6.750	10
	DPC	5.600	6.000	0.800	4.000	7.000	5.000	6.000	10
Perceived Competence	1 Mode	5.450	5.875	1.396	1.750	6.750	5.062	6.250	10
	DPC	5.750	5.750	0.689	4.750	7.000	5.312	6.188	10

Table B.6: Summary Statistics of Perceived Competence Metrics in `Shelf` compared to PCA baseline. We used a Mann-Whitney test, to test for significance, since none were normally distributed, but found no statistically significant results.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
User Confidence	PCA	4.400	4.500	1.800	2.000	7.000	3.000	5.750	10
	DPC	4.800	5.000	1.470	2.000	7.000	4.000	6.000	10
Task Completion	PCA	4.000	3.500	1.732	2.000	7.000	3.000	4.750	10
	DPC	4.800	5.500	1.833	2.000	7.000	3.250	6.000	10
Robot Navigation	PCA	3.900	3.000	1.758	2.000	7.000	3.000	4.750	10
	DPC	4.900	5.500	1.578	2.000	7.000	3.500	6.000	10
User Autonomy	PCA	3.900	3.000	1.868	2.000	7.000	2.250	5.000	10
	DPC	4.900	5.500	1.578	2.000	7.000	3.500	6.000	10
Perceived Competence	PCA	4.050	3.875	1.680	2.000	7.000	2.625	4.812	10
	DPC	4.850	5.250	1.570	2.250	7.000	3.438	6.000	10

Table B.7: Summary Statistics of Perceived Competence Metrics in `Shelf` compared to Cartesian baseline. We ran a Mann-Whitney hypothesis test. We include a W superscript for statistically significant results ($p < 0.05$).

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
User Confidence^{W}	Cartesian	6.400	7.000	0.800	5.000	7.000	6.000	7.000	10
	DPC	4.700	5.000	1.552	2.000	7.000	3.250	6.000	10
Task Completion^{W}	Cartesian	6.800	7.000	0.600	5.000	7.000	7.000	7.000	10
	DPC	4.800	5.000	1.536	2.000	7.000	3.500	6.000	10
Robot Navigation^{W}	Cartesian	6.700	7.000	0.640	5.000	7.000	7.000	7.000	10
	DPC	4.800	5.000	1.249	3.000	7.000	4.000	5.750	10
User Autonomy^{W}	Cartesian	6.500	7.000	0.806	5.000	7.000	6.250	7.000	10
	DPC	4.800	5.000	1.600	2.000	7.000	3.500	5.750	10
Perceived Competence^{W}	Cartesian	6.600	7.000	0.654	5.000	7.000	6.562	7.000	10
	DPC	4.775	5.000	1.389	2.250	7.000	3.812	5.688	10

Table B.8: `Shelf` Summary Likert Question Results for 1 Mode baseline. These results did not indicate any statistical significance, using either the Brunner-Munzel test or Mood's median test.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
Accuracy	1 Mode	5.000	5.000	1.183	2.000	6.000	5.000	6.000	10
	DPC	5.200	6.000	1.077	3.000	6.000	4.250	6.000	10
Predictability	1 Mode	4.600	5.000	1.356	2.000	6.000	5.000	5.000	10
	DPC	5.400	5.000	1.114	3.000	7.000	5.000	6.000	10
Recoverability	1 Mode	4.700	5.000	1.552	1.000	7.000	4.000	5.750	10
	DPC	4.500	5.000	1.688	2.000	7.000	3.250	5.000	10
Adaptability	1 Mode	5.000	5.000	1.265	3.000	7.000	4.250	6.000	10
	DPC	5.500	6.000	0.922	4.000	7.000	5.000	6.000	10
Control	1 Mode	4.600	5.500	1.685	1.000	6.000	3.250	6.000	10
	DPC	5.100	5.500	1.375	2.000	7.000	4.250	6.000	10
Intuitiveness	1 Mode	5.000	5.500	1.673	1.000	7.000	5.000	6.000	10
	DPC	5.100	5.500	1.375	2.000	7.000	4.250	6.000	10
Reliability	1 Mode	4.800	5.000	1.661	1.000	7.000	4.250	6.000	10
	DPC	5.100	5.500	1.044	3.000	6.000	4.250	6.000	10
Mapping	1 Mode	5.700	6.000	1.345	2.000	7.000	6.000	6.000	10
	DPC	5.600	6.000	1.428	3.000	7.000	5.250	6.750	10
Precision	1 Mode	5.000	5.500	1.483	2.000	7.000	4.250	6.000	10
	DPC	5.800	6.000	0.748	4.000	7.000	6.000	6.000	10
Naturalness	1 Mode	4.600	5.500	2.010	1.000	7.000	2.750	6.000	10
	DPC	4.800	5.500	1.327	3.000	6.000	3.250	6.000	10

Table B.9: `Shelf` Likert Scale Summary statistics Cartesian Control Baseline. Superscripts indicate statistical significance ($p < 0.05$) between the two groups. Here B indicates significance with respect to Brunnel-Munzel's test and M with respect to Mood's median test.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
Accuracy^{B}	Cartesian	6.300	6.500	0.900	4.000	7.000	6.000	7.000	10
	DPC	4.100	4.000	1.578	2.000	7.000	3.000	5.000	10
Predictability^{B,M}	Cartesian	6.400	7.000	1.020	4.000	7.000	6.250	7.000	10
	DPC	3.700	3.500	1.552	1.000	6.000	3.000	4.750	10
Recoverability^{B}	Cartesian	5.800	6.000	1.249	3.000	7.000	5.000	7.000	10
	DPC	3.800	4.000	1.327	2.000	6.000	3.000	4.000	10
Adaptability^{B,M}	Cartesian	6.400	7.000	0.917	4.000	7.000	6.000	7.000	10
	DPC	4.200	4.500	1.327	2.000	6.000	3.000	5.000	10
Control^{B}	Cartesian	6.400	7.000	0.917	4.000	7.000	6.000	7.000	10
	DPC	4.300	4.500	1.616	1.000	7.000	3.250	5.000	10
Intuitiveness^{B,M}	Cartesian	6.300	7.000	1.005	4.000	7.000	6.000	7.000	10
	DPC	4.000	4.000	1.549	2.000	7.000	3.000	4.750	10
Reliability^{B,M}	Cartesian	6.000	6.000	1.183	3.000	7.000	6.000	7.000	10
	DPC	4.300	4.500	1.187	3.000	6.000	3.000	5.000	10
Mapping^{B}	Cartesian	5.500	6.000	1.803	1.000	7.000	5.000	7.000	10
	DPC	4.100	4.500	1.513	1.000	6.000	3.000	5.000	10
Precision^{B}	Cartesian	5.900	6.000	1.136	4.000	7.000	5.000	7.000	10
	DPC	4.400	4.500	1.114	3.000	6.000	3.250	5.000	10
Naturalness^{B}	Cartesian	5.600	6.000	1.428	2.000	7.000	5.000	6.750	10
	DPC	3.600	4.000	2.107	1.000	7.000	1.250	5.000	10

Table B.10: `Shelf` Summary Statistics of Likert Questions for PCA Baseline. Here the results did not indicate any statistical significance ($p < 0.05$), using either the Brunner-Munzel test or Mood's median test.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
Accuracy	PCA	4.000	3.500	1.483	2.000	7.000	3.000	4.750	10
	DPC	4.600	5.000	1.428	2.000	6.000	4.250	5.750	10
Predictability	PCA	4.300	4.000	1.616	2.000	7.000	3.000	5.000	10
	DPC	4.600	5.000	1.200	2.000	6.000	4.250	5.000	10
Recoverability	PCA	4.000	3.500	1.732	2.000	7.000	3.000	4.750	10
	DPC	4.400	5.000	1.356	2.000	6.000	4.000	5.000	10
Adaptability	PCA	4.700	4.000	1.418	3.000	7.000	4.000	5.750	10
	DPC	5.300	5.500	1.418	3.000	7.000	4.000	6.750	10
Control	PCA	4.300	4.000	1.552	2.000	7.000	3.000	5.750	10
	DPC	4.600	4.500	1.562	2.000	7.000	3.250	6.000	10
Intuitiveness	PCA	3.800	4.500	1.833	1.000	6.000	2.250	5.000	10
	DPC	4.400	5.000	1.855	1.000	6.000	2.750	6.000	10
Reliability	PCA	4.200	4.000	1.400	2.000	7.000	3.250	4.750	10
	DPC	4.500	5.000	1.360	2.000	6.000	4.250	5.000	10
Mapping	PCA	4.000	3.500	1.844	2.000	7.000	2.250	5.000	10
	DPC	4.300	5.000	1.900	1.000	7.000	2.500	5.750	10
Precision	PCA	4.000	3.500	1.612	2.000	7.000	3.000	5.000	10
	DPC	4.800	5.000	1.600	2.000	7.000	4.250	6.000	10
Naturalness	PCA	3.600	3.500	1.625	1.000	6.000	2.250	4.750	10
	DPC	4.200	5.000	1.600	1.000	6.000	3.250	5.000	10

Table B.11: `Shelf` Task Nasa TLX summary statistics. We tested for statistical significance using the Mann-Whitney test for PCA vs DPC and t-test for Cartesian vs DPC such as 1 Mode vs DPC. However, no statistical significance could be determined.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
Nasa TLX	DPC	53.533	56.333	11.578	23.333	64.000	51.833	61.333	10
	PCA	51.800	53.667	10.332	34.000	70.000	44.833	58.500	10
Nasa TLX	DPC	64.333	65.000	8.285	52.667	75.333	56.333	71.333	10
	Cartesian	51.667	46.667	18.226	14.000	84.000	43.833	64.500	10
Nasa TLX	1 Mode	60.667	60.667	8.121	50.667	73.333	52.833	66.667	10
	DPC	57.400	59.667	11.409	30.667	74.000	56.167	63.333	10

Table B.12: 1 Mode baseline on *Shelf* task: Comparison of DPC and 1 Mode Systems. Superscript W indicates statistical significance between the two groups. We used Mann-Whitney test to test for statistical significance ($p < 0.05$), since the results were not normal distributed.

Metric	DPC	1 Mode
Time (Mean ± Std)	140.34 ± 87.69	170.51 ± 95.65
Mode Switches (Mean ± Std)^{W}	10.23 ± 9.78	0.00 ± 0.00
Resets Invol. (Mean ± Std)	0.30 ± 0.53	0.23 ± 0.43
Resets Vol. (Mean ± Std)	0.13 ± 0.35	0.30 ± 0.53
Resets Total (Mean ± Std)	0.43 ± 0.68	0.53 ± 0.63
Mode 1 Time (Mean ± Std)^{W}	81.92 ± 43.89	170.51 ± 95.65
Mode 2 Time (Mean ± Std)^{W}	32.81 ± 30.42	-
Mode 3 Time (Mean ± Std)^{W}	25.61 ± 37.68	-
Success Rate	0.83	0.87
Joystick Smoothness (Mean ± Std)	-51.49 ± 17.17	-53.64 ± 16.53
Robot Smoothness (Mean ± Std)	-13.69 ± 5.77	-12.56 ± 4.62

Table B.13: `Shelf`: Comparison of DPC and Cartesian Systems. Superscript W indicates statistical significance between the two groups. We used Mann-Whitney test to test for statistical significance ($p < 0.05$), since the results were not normal distributed.

Metric	DPC	Cartesian
Time (Mean ± Std)^{W}	150.64 ± 82.89	102.47 ± 41.37
Mode Switches (Mean ± Std)	17.73 ± 11.89	12.07 ± 4.89
Resets Invol. (Mean ± Std)^{W}	0.20 ± 0.55	0.00 ± 0.00
Resets Vol. (Mean ± Std)	0.27 ± 0.64	0.03 ± 0.18
Resets Total (Mean ± Std)^{W}	0.47 ± 0.78	0.03 ± 0.18
Mode 1 Time (Mean ± Std)^{W}	76.40 ± 36.29	47.79 ± 22.78
Mode 2 Time (Mean ± Std)	40.41 ± 33.40	32.98 ± 15.70
Mode 3 Time (Mean ± Std)	33.83 ± 30.53	21.70 ± 8.02
Success Rate^{W}	0.87	1.00
Joystick Smoothness (Mean ± Std)	-55.92 ± 19.26	-55.16 ± 12.06
Robot Smoothness (Mean ± Std)^{W}	-14.68 ± 5.81	-9.61 ± 2.59

Table B.14: `Shelf`: Comparison of DPC and PCA Systems. Super script W indicates statistical significance between the two groups. We used Mann-Whitney test to test for statistical significance ($p < 0.05$), since the results were not normal distributed.

Metric	DPC	PCA
Time (Mean ± Std)^{W}	147.34 ± 87.71	198.40 ± 99.30
Mode Switches (Mean ± Std)^{W}	14.07 ± 13.47	25.87 ± 16.60
Resets Invol. (Mean ± Std)	0.20 ± 0.48	0.27 ± 0.52
Resets Vol. (Mean ± Std)	0.43 ± 0.73	0.30 ± 0.65
Resets Total (Mean ± Std)	0.63 ± 1.07	0.57 ± 0.82
Mode 1 Time (Mean ± Std)	84.98 ± 52.14	84.13 ± 45.67
Mode 2 Time (Mean ± Std)^{W}	33.16 ± 29.70	57.17 ± 38.36
Mode 3 Time (Mean ± Std)^{W}	29.20 ± 29.28	62.88 ± 32.39
Success Rate^{W}	0.83	0.50
Joystick Smoothness (Mean ± Std)^{W}	-51.11 ± 18.81	-63.20 ± 17.05
Robot Smoothness (Mean ± Std)^{W}	-14.42 ± 5.18	-17.58 ± 4.60

Table B.15: `Pouring` Task Perceived Competence results with PCA. Superscript W indicates statistical significance with respect to Mann-Whitney test and t with respect to t-test.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
User Confidence^{W}	DPC	4.800	5.000	1.077	3.000	7.000	4.000	5.000	10
	PCA	2.200	2.000	1.249	1.000	5.000	1.000	3.000	10
Task Completion^{W}	DPC	5.600	6.000	1.281	3.000	7.000	5.000	6.750	10
	PCA	2.200	2.000	0.980	1.000	4.000	2.000	2.000	10
Robot Navigation^{t}	DPC	4.900	5.000	1.136	3.000	7.000	4.000	5.750	10
	PCA	2.700	3.000	1.345	1.000	5.000	1.250	3.750	10
User Autonomy^{W}	DPC	6.100	6.000	0.831	5.000	7.000	5.250	7.000	10
	PCA	3.200	3.000	1.077	1.000	5.000	3.000	4.000	10
Perceived Competence^{t}	DPC	5.350	5.250	0.910	4.000	7.000	4.562	5.938	10
	PCA	2.575	2.625	0.807	1.500	4.000	1.875	3.125	10

Table B.16: Likert Scale Question statistics `Pouring`. Superscripts indicate statistical significance ($p < 0.05$) between the two groups. Here B indicates significance with respect to Brunnel-Munzel's test and M with respect to Mood's median test.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
Accuracy^{B,M}	PCA	2.300	2.000	0.781	1.000	4.000	2.000	2.750	10
	DPC	5.100	5.000	1.044	4.000	7.000	4.000	6.000	10
Predictability^{B,M}	PCA	2.600	3.000	1.020	1.000	4.000	2.000	3.000	10
	DPC	4.600	4.500	1.200	3.000	7.000	4.000	5.000	10
Recoverability	PCA	4.400	4.500	1.855	1.000	7.000	3.250	6.000	10
	DPC	5.600	5.500	1.497	2.000	7.000	5.000	7.000	10
Adaptability^{B,M}	PCA	2.900	3.500	1.446	1.000	5.000	1.250	4.000	10
	DPC	5.600	6.000	0.800	4.000	7.000	5.000	6.000	10
Control^{B,M}	PCA	2.200	2.000	1.166	1.000	5.000	1.250	2.750	10
	DPC	4.600	4.000	1.281	3.000	7.000	4.000	5.750	10
Intuitiveness^{B,M}	PCA	2.000	2.000	0.775	1.000	3.000	1.250	2.750	10
	DPC	4.600	5.000	0.800	3.000	6.000	4.000	5.000	10
Reliability^{B,M}	PCA	3.100	3.000	1.640	1.000	6.000	1.500	4.000	10
	DPC	5.400	5.500	0.917	4.000	7.000	5.000	6.000	10
Mapping^{B}	PCA	3.000	2.000	2.000	1.000	7.000	1.250	4.750	10
	DPC	5.800	6.000	1.166	4.000	7.000	5.000	7.000	10
Precision	PCA	5.300	5.000	1.100	4.000	7.000	4.250	6.000	10
	DPC	5.900	6.000	1.221	3.000	7.000	5.250	7.000	10
Naturalness^{B,M}	PCA	2.100	2.000	0.831	1.000	3.000	1.250	3.000	10
	DPC	4.900	5.000	1.221	3.000	7.000	4.250	5.750	10

Table B.17: Summary Statistics of Nasa TLX `Pouring`. Testing for statistical significance ($p < 0.05$), using the the t-test did not yield positive results in any of the two groups.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
Nasa TLX	DPC	47.200	49.333	9.941	25.333	59.333	41.167	54.667	10
	PCA	59.333	62.333	17.119	32.000	84.000	48.500	70.667	10
Nasa TLX	DPC	57.833	57.333	10.943	39.333	78.667	51.667	61.833	8
	Cartesian	53.926	56.667	11.744	35.333	76.667	44.667	58.667	9

Table B.18: Performance Metrics `Pouring` – Baseline: PCA. Superscript W indicates statistical significance between the two groups. We used Mann-Whitney test to test for statistical significance ($p < 0.05$), since the results were not normal distributed.

Metric	**PCA**	**DPC**
Time (Mean ± Std)^W	273.62 ± 38.25	185.19 ± 72.17
Mode Switches^W	37.74 ± 11.80	15.63 ± 15.09
Resets Involuntary^W	0.63 ± 1.01	0.15 ± 0.60
Resets Voluntary^W	0.37 ± 0.56	0.07 ± 0.27
Resets Total^W	1.00 ± 1.11	0.22 ± 0.64
Mode 1 Time Spent	145.34 ± 42.55	148.52 ± 55.74
Mode 2 Time Spent^W	88.07 ± 23.66	44.55 ± 34.59
Mode 3 Time Spent^W	56.27 ± 20.75	21.13 ± 27.33
Joystick Smoothness^W	-85.03 ± 15.46	-72.82 ± 23.37
Robot Smoothness^W	-36.87 ± 7.81	-22.28 ± 6.81
Success Rate^W	0.333	0.852

Table B.19: Performance Metrics – Baseline: Cartesian `Pouring`. Superscript W indicates statistical significance between the two groups. We used Mann-Whitney test to test for statistical significance ($p < 0.05$), since the results were not normal distributed.

Metric	**Cartesian**	**DPC**
Time (Mean ± Std)^W	250.77 ± 54.47	184.59 ± 88.44
Mode Switches^W	36.67 ± 11.60	11.89 ± 11.00
Resets Involuntary^W	0.52 ± 0.70	0.11 ± 0.32
Resets Voluntary	0.04 ± 0.19	0.15 ± 0.46
Resets Total	0.56 ± 0.75	0.26 ± 0.66
Mode 1 Time Spent^W	77.25 ± 24.13	159.92 ± 68.39
Mode 2 Time Spent^W	126.11 ± 29.45	39.34 ± 31.57
Mode 3 Time Spent^W	57.00 ± 33.22	16.14 ± 21.61
Joystick smoothness^W	-103.76 ± 28.08	-73.26 ± 26.80
Robot Smoothness^W	-32.02 ± 7.58	-21.79 ± 9.21
Success Rate	0.519	0.667

Table B.20: Likert Scale Question statistics `Pouring` with Cartesian Baseline. Superscripts indicate statistical significance ($p < 0.05$) between the two groups. Here B indicates significance with respect to Brunnel-Munzel's test and M with respect to Mood's median test.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
Accuracy	Cartesian	5.778	6.000	1.618	2.000	7.000	6.000	7.000	9
	DPC	4.500	4.500	1.414	3.000	7.000	3.000	5.250	8
Predictability^{B}	Cartesian	5.889	6.000	1.523	2.000	7.000	6.000	7.000	9
	DPC	4.000	3.500	1.581	2.000	7.000	3.000	4.500	8
Recoverability	Cartesian	5.222	5.000	1.315	3.000	7.000	4.000	6.000	9
	DPC	5.250	6.000	1.299	2.000	6.000	5.000	6.000	8
Adaptability	Cartesian	5.667	6.000	1.247	3.000	7.000	6.000	6.000	9
	DPC	4.875	5.500	1.615	2.000	7.000	3.750	6.000	8
Control^{B}	Cartesian	5.889	6.000	1.286	3.000	7.000	5.000	7.000	9
	DPC	4.250	3.500	1.392	3.000	6.000	3.000	6.000	8
Intuitiveness	Cartesian	4.667	5.000	1.563	2.000	7.000	3.000	6.000	9
	DPC	4.375	4.000	1.409	2.000	6.000	3.750	6.000	8
Reliability^{B,M}	Cartesian	6.222	7.000	1.227	3.000	7.000	6.000	7.000	9
	DPC	5.250	5.000	0.661	4.000	6.000	5.000	6.000	8
Mapping	Cartesian	5.778	6.000	1.030	4.000	7.000	5.000	7.000	9
	DPC	4.875	4.500	1.269	3.000	7.000	4.000	6.000	8
Precision^{B}	Cartesian	6.444	7.000	0.685	5.000	7.000	6.000	7.000	9
	DPC	4.875	6.000	1.763	2.000	7.000	3.000	6.000	8
Naturalness	Cartesian	4.556	4.000	1.641	2.000	7.000	3.000	6.000	9
	DPC	5.375	6.000	1.576	2.000	7.000	4.750	6.250	8

Table B.21: `Pouring` Task Perceived Competence results with Cartesian. Superscript W indicates statistical significance with respect to Mann-Whitney test and t with respect to t-test.

Metric	Group	Mean	Median	StdDev	Min	Max	Q1	Q3	Count
User Confidence	Cartesian	5.667	6.000	1.054	4.000	7.000	5.000	6.000	9
	3 Mode	5.000	5.000	1.000	3.000	6.000	4.750	6.000	8
Task Completion	Cartesian	5.889	6.000	0.875	5.000	7.000	5.000	7.000	9
	3 Mode	5.500	6.000	1.118	3.000	7.000	5.000	6.000	8
Robot Navigation^{W}	Cartesian	6.111	6.000	0.994	4.000	7.000	6.000	7.000	9
	3 Mode	4.750	5.000	1.299	2.000	6.000	4.000	6.000	8
User Autonomy	Cartesian	6.333	7.000	0.816	5.000	7.000	6.000	7.000	9
	3 Mode	5.750	6.000	1.199	3.000	7.000	5.750	6.250	8
Perceived Competence	Cartesian	6.000	6.000	0.791	5.000	7.000	5.250	6.750	9
	3 Mode	5.250	5.500	0.952	3.250	6.250	5.000	6.000	8

Appendix C: Morphology Aware Transfer

C.1 Direct Finetuning Configurations

In our experiments, we consider various finetuning scenarios in our evaluations. For direct finetuning methods, we include combinations of subsets we finetune online in Table C.1. Our evaluations included subsets of the direct tuning configurations of weight combinations. For example, *Input Embedding* includes combinations in which just W^{embed}, W^{position} and both $\{W^{\text{emebed}}, W^{\text{position}}\}$ are tuned online during training.

C.2 LoRA Initialization Details

When using LoRA in our experiments, we initialize B to small Gaussian noise $b_{ij} \sim N(0; 10^{-4})$ and A to a zero matrix which eliminates LoRA adapters affect on the zeros-hot performance at the beginning of training. LoRA was included as a finetuning

Table C.1: Layer tuning parameters and experiment identifiers

Layer Tuned	**Parameters ϕ**	**Exp. Identifier**
End-to-end	θ^*	E2E
Transformer layers	$\{T_i; i \in [1, L]\}$	Layer 5
Attention layers	$\{W_i^{\text{attn}}; i \in [1, L]\}$	Lora
Nonlinear transformation	$\{W_i^{\text{in}}, W_i^{\text{out}}; i \in [1, L]\}$	Lora
Input Embedding	$\{W^{\text{embed}}, W^{\text{position}}\}$	Embedding
Decoder	$\{W_i^{\text{decoder}}; i \in [1, L^{\text{dec}}]\}$	Decoder

method because we want to reduce the total number of parameters used which LoRA can explicitly do via the rank.

C.3 Morphology-Aware Policy Performance

This section reports results for the best-performing PEFT algorithms for each significant grouping of methods we consider. Table C.2 show flat terrain results, Table C.3 shows variable terrain results, and Table C.4 shows results for obstacle avoidance. These results report statistical significance when comparing results to zero-shot pre-training performance and training policies from scratch. Surprisingly, training from scratch worked surprisingly well in flat terrain. Still, most PEFT techniques perform better after five million samples than training from scratch on more complex tasks.

C.4 Prefix Tuning Additional Results

In this section, we include plots similar to those in the main paper for our prefix-tuning ablation experiments. Flat terrain results are shown in Figure C.1 and obstacle avoidance in Figure C.2. We also show similar ablation results for LoRA and prefix tuning for flat terrain in Figure C.3 and obstacle avoidance in Figure C.4.

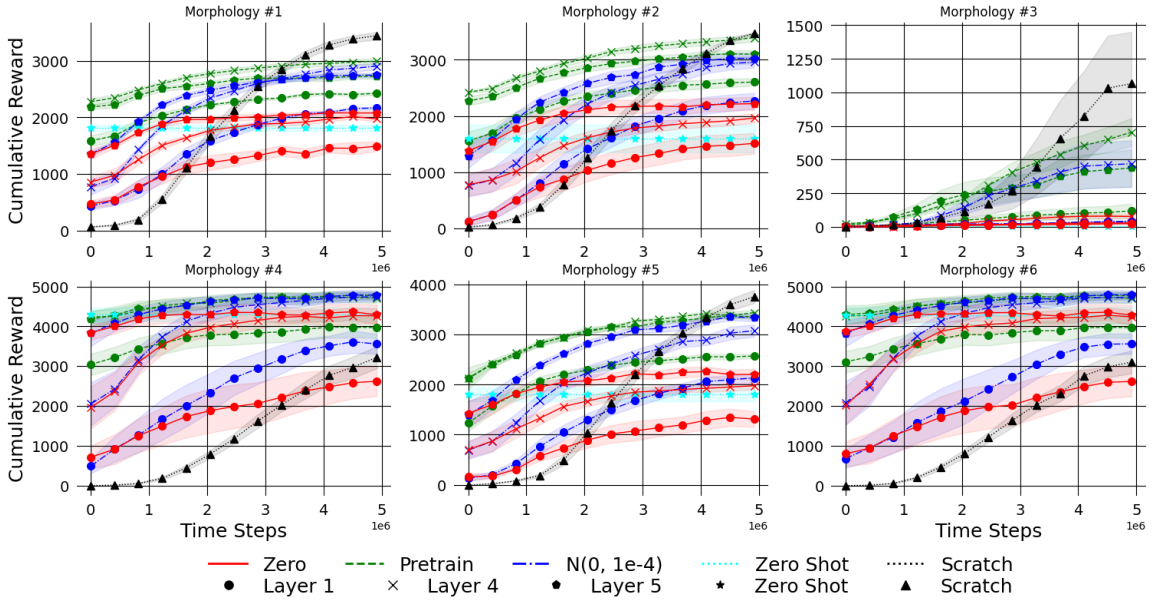


Figure C.1: Choice of initialization and injection layers of prefix tuning in flat terrain. Initial zero-shot results of E2E learning are plotted to compare affect of prefixes.

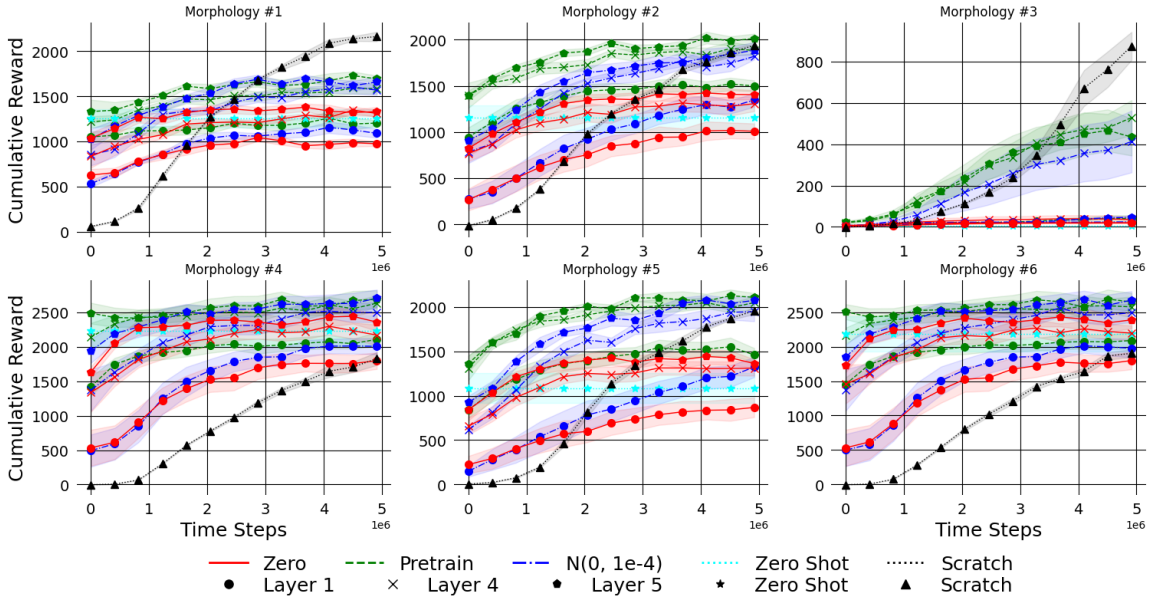


Figure C.2: Choice of initialization and injection layers of prefix tuning in obstacle avoidance. Initial zero-shot results of E2E learning are plotted to compare affect of prefixes.

Table C.2: Flat Terrain Cumulative Rewards for each testing morphology. Values show mean (top) and standard deviation (bottom). † statistical significance compared to Zero Shot and ‡ statistical significance to Scratch ($p < 0.01$). P-values and the hypothesis test run (T: t-test, M: Mann-Whitney) comparing against Zero shot and Scratch results.

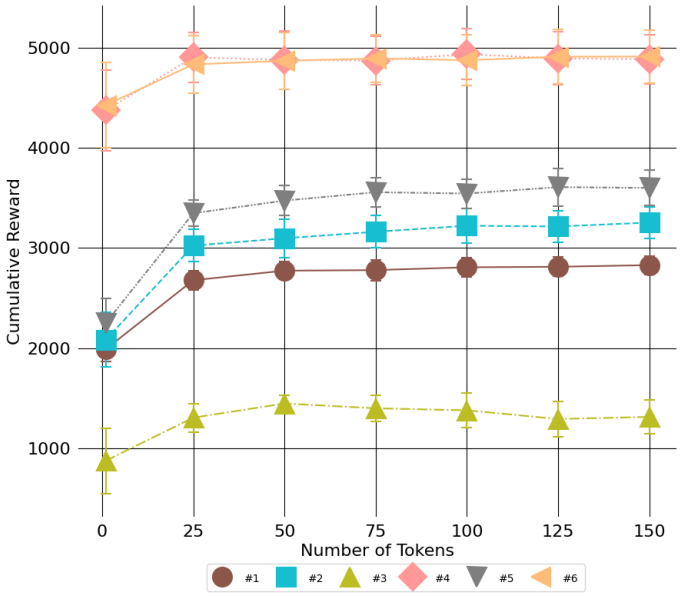
Morphology	1	2	3	4	5	6
Full Model	$4281.46^{\dagger\ddagger}$	$4552.77^{\dagger\ddagger}$	$1635.82^{\dagger}$	$5545.44^{\dagger\ddagger}$	$5019.71^{\dagger\ddagger}$	$5558.61^{\dagger\ddagger}$
	± 181.77	± 239.11	± 405.64	± 280.16	± 143.13	± 286.80
Layer 4	$3761.11^{\dagger}$	$4121.84^{\dagger}$	$1491.06^{\dagger}$	$5183.88^{\dagger\ddagger}$	$4666.95^{\dagger\ddagger}$	$5192.58^{\dagger\ddagger}$
	± 102.11	± 120.48	± 230.10	± 167.91	± 255.22	± 152.33
Lora	$3798.90^{\dagger}$	$4208.41^{\dagger\ddagger}$	$1639.69^{\dagger}$	$5223.47^{\dagger\ddagger}$	$4761.58^{\dagger\ddagger}$	$5223.47^{\dagger\ddagger}$
	± 138.55	± 77.12	± 41.31	± 174.25	± 210.57	± 174.25
Decoder Only	$2732.26^{\dagger\ddagger}$	$3112.46^{\dagger\ddagger}$	$1398.42^{\dagger}$	$4868.54^{\ddagger}$	$3404.67^{\dagger\ddagger}$	$4858.76^{\ddagger}$
	± 71.35	± 221.65	± 210.79	± 263.81	± 92.07	± 248.01
Embeding	$3308.43^{\dagger\ddagger}$	$3684.66^{\dagger}$	$1554.28^{\dagger}$	$4986.16^{\ddagger}$	$4062.05^{\dagger}$	$4997.82^{\ddagger}$
	± 115.83	± 104.99	± 190.82	± 183.78	± 248.20	± 191.07
Input Adapt	$3231.84^{\dagger\ddagger}$	$3529.41^{\dagger}$	$1510.46^{\dagger}$	$4927.72^{\ddagger}$	$3946.59^{\dagger}$	$4963.53^{\ddagger}$
	± 100.03	± 104.58	± 242.36	± 225.75	± 220.78	± 222.35
Prefix	$3332.33^{\dagger\ddagger}$	$3750.54^{\dagger}$	$1604.92^{\dagger}$	$5064.15^{\ddagger}$	$4199.89^{\dagger}$	$5066.47^{\ddagger}$
	± 126.28	± 201.62	± 336.88	± 133.91	± 276.27	± 137.63
Scratch	$3754.15^{\dagger}$	$3840.33^{\dagger}$	$2191.50^{\dagger}$	3727.29	$4085.82^{\dagger}$	3608.55
	± 210.65	± 211.59	± 624.72	± 733.60	± 217.00	± 777.33
Zero Shot	1867.58	1703.19	253.70	4392.08	1849.41	4431.93
	± 82.55	± 447.69	± 188.25	± 434.01	± 338.10	± 405.78
	P-Values comparing against Zeroshot Performance†					
Full Model	$9.1 \times 10^{-9}(T)$	$3.6 \times 10^{-6}(T)$	$2.6 \times 10^{-4}(T)$	$2.1 \times 10^{-3}(T)$	$1.3 \times 10^{-7}(T)$	$1.9 \times 10^{-3}(T)$
Layer 4	$2.3 \times 10^{-9}(T)$	$6.2 \times 10^{-6}(T)$	$3.3 \times 10^{-5}(T)$	$9.3 \times 10^{-3}(T)$	$9.7 \times 10^{-7}(T)$	$8.0 \times 10^{-3}(T)$
Lora	$9.8 \times 10^{-9}(T)$	$4.1 \times 10^{-6}(T)$	$5.3 \times 10^{-7}(T)$	$7.5 \times 10^{-3}(T)$	$4.7 \times 10^{-7}(T)$	$7.1 \times 10^{-3}(T)$
Decoder Only	$2.5 \times 10^{-7}(T)$	$4.9 \times 10^{-4}(T)$	$4.0 \times 10^{-5}(T)$	$9.7 \times 10^{-2}(T)$	$2.1 \times 10^{-5}(T)$	$1.1 \times 10^{-1}(T)$
Embeding	$3.7 \times 10^{-8}(T)$	$2.5 \times 10^{-5}(T)$	$7.9 \times 10^{-3}(M)$	$3.6 \times 10^{-2}(T)$	$5.7 \times 10^{-6}(T)$	$3.6 \times 10^{-2}(T)$
Input Adapt	$2.7 \times 10^{-8}(T)$	$4.6 \times 10^{-5}(T)$	$7.9 \times 10^{-3}(M)$	$6.0 \times 10^{-2}(T)$	$6.4 \times 10^{-6}(T)$	$5.1 \times 10^{-2}(T)$
Prefix	$5.1 \times 10^{-8}(T)$	$3.2 \times 10^{-5}(T)$	$1.1 \times 10^{-4}(T)$	$1.8 \times 10^{-2}(T)$	$4.9 \times 10^{-6}(T)$	$1.8 \times 10^{-2}(T)$
Scratch	$0.0(T)$	$2.9 \times 10^{-8}(T)$	$2.7 \times 10^{-5}(T)$	$1.1 \times 10^{-1}(T)$	$2.2 \times 10^{-9}(T)$	$5.9 \times 10^{-2}(T)$
Zero Shot	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$
	P-Values comparing against Scratch Performance‡					
Full Model	$6.6 \times 10^{-4}(T)$	$1.1 \times 10^{-4}(T)$	$1.2 \times 10^{-1}(T)$	$2.5 \times 10^{-4}(T)$	$1.9 \times 10^{-6}(T)$	$2.2 \times 10^{-4}(T)$
Layer 4	$9.5 \times 10^{-1}(T)$	$2.3 \times 10^{-2}(T)$	$4.2 \times 10^{-2}(T)$	$1.3 \times 10^{-3}(T)$	$8.9 \times 10^{-4}(T)$	$1.0 \times 10^{-3}(T)$
Lora	$6.9 \times 10^{-1}(T)$	$3.9 \times 10^{-3}(T)$	$8.9 \times 10^{-2}(T)$	$1.1 \times 10^{-3}(T)$	$1.3 \times 10^{-4}(T)$	$9.1 \times 10^{-4}(T)$
Decoder Only	$2.2 \times 10^{-7}(T)$	$6.7 \times 10^{-5}(T)$	$2.3 \times 10^{-2}(T)$	$7.8 \times 10^{-3}(T)$	$2.9 \times 10^{-5}(T)$	$6.1 \times 10^{-3}(T)$
Embeding	$1.2 \times 10^{-3}(T)$	$1.7 \times 10^{-1}(T)$	$1.6 \times 10^{-1}(M)$	$3.8 \times 10^{-3}(T)$	$8.6 \times 10^{-1}(T)$	$2.9 \times 10^{-3}(T)$
Input Adapt	$2.9 \times 10^{-4}(T)$	$1.3 \times 10^{-2}(T)$	$9.9 \times 10^{-2}(M)$	$5.4 \times 10^{-3}(T)$	$3.0 \times 10^{-1}(T)$	$3.5 \times 10^{-3}(T)$
Prefix	$2.1 \times 10^{-3}(T)$	$4.8 \times 10^{-1}(T)$	$9.1 \times 10^{-2}(T)$	$2.4 \times 10^{-3}(T)$	$4.3 \times 10^{-1}(T)$	$1.9 \times 10^{-3}(T)$
Scratch	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$
Zero Shot	$0.0(T)$	$2.9 \times 10^{-8}(T)$	$2.7 \times 10^{-5}(T)$	$1.1 \times 10^{-1}(T)$	$2.2 \times 10^{-9}(T)$	$5.9 \times 10^{-2}(T)$

Table C.3: Variable Terrain Cumulative Rewards for each testing morphology. Values show mean (top) and standard deviation (bottom). † statistical significance compared to Zero Shot and ‡ statistical significance to Scratch ($p < 0.01$). P-values and the hypothesis test run (T: t-test, M: Mann-Whitney) comparing against Zero shot and Scratch results.

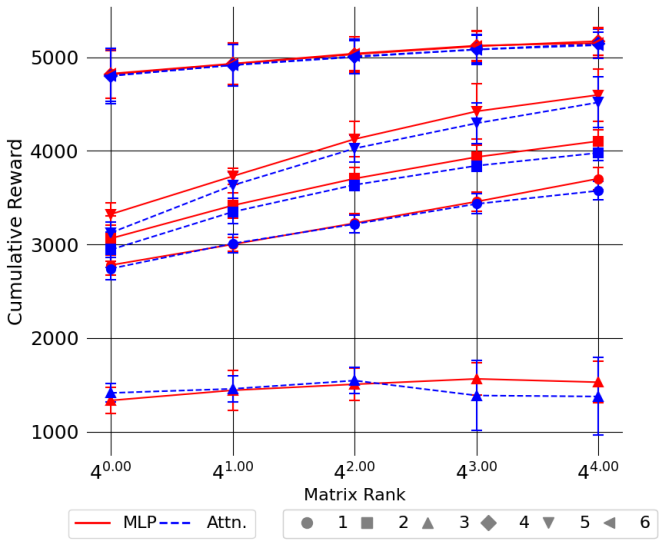
Morphology	1	2	3	4	5	6
Full Model	$2253.96^{\dagger\ddagger}$	$1983.81^{\dagger\ddagger}$	$2001.18^{\dagger\ddagger}$	$3560.43^{\dagger\ddagger}$	$2047.49^{\dagger\ddagger}$	$3595.38^{\dagger\ddagger}$
	± 41.47	± 154.82	± 42.14	± 317.89	± 117.06	± 368.99
Layer 4	$2093.75^{\dagger\ddagger}$	$1871.09^{\dagger\ddagger}$	$1879.22^{\dagger\ddagger}$	$3254.06^{\dagger\ddagger}$	$1912.17^{\dagger\ddagger}$	$3279.03^{\ddagger}$
	± 34.23	± 79.86	± 33.91	± 353.70	± 135.29	± 379.46
Lora	$2141.39^{\dagger\ddagger}$	$1848.53^{\dagger\ddagger}$	$1786.88^{\dagger\ddagger}$	$3230.13^{\dagger\ddagger}$	$1878.93^{\dagger\ddagger}$	$3234.25^{\ddagger}$
	± 53.29	± 113.44	± 72.97	± 327.39	± 107.89	± 329.42
Decoder Only	$1969.63^{\dagger\ddagger}$	$1623.70^{\dagger\ddagger}$	$1299.89^{\dagger}$	$3164.72^{\dagger\ddagger}$	$1672.47^{\dagger\ddagger}$	$3180.90^{\ddagger}$
	± 28.01	± 126.14	± 70.71	± 307.28	± 112.14	± 316.43
Embeding	$1836.54^{\dagger\ddagger}$	$1529.38^{\dagger}$	$1441.65^{\dagger}$	$2872.51^{\ddagger}$	$1549.29^{\dagger}$	$2887.67^{\ddagger}$
	± 25.22	± 84.38	± 41.51	± 307.30	± 106.71	± 311.40
Input Adapt	$1820.01^{\dagger\ddagger}$	$1521.18^{\dagger}$	$1338.57^{\dagger}$	$2869.53^{\ddagger}$	$1512.25^{\dagger}$	$2895.01^{\ddagger}$
	± 48.63	± 106.76	± 61.81	± 293.57	± 109.46	± 299.56
Prefix	$1902.95^{\dagger\ddagger}$	$1643.33^{\dagger\ddagger}$	$1406.55^{\dagger}$	$2930.13^{\ddagger}$	$1601.95^{\dagger}$	$2918.47^{\ddagger}$
	± 43.36	± 165.26	± 83.55	± 261.58	± 134.90	± 300.10
Scratch	$1679.33^{\dagger}$	$1406.59^{\dagger}$	$1406.58^{\dagger}$	$1735.22^{\dagger}$	$1449.59^{\dagger}$	$1758.99^{\dagger}$
	± 82.91	± 101.71	± 164.55	± 166.72	± 69.66	± 168.21
Zero Shot	1259.92	591.83	136.82	2452.59	685.54	2476.77
	± 61.93	± 67.70	± 103.66	± 291.96	± 71.67	± 349.00
	P-Values comparing against Zeroshot Performance†					
Full Model	$4.6 \times 10^{-9}(T)$	$2.1 \times 10^{-6}(T)$	$5.2 \times 10^{-6}(T)$	$4.7 \times 10^{-9}(T)$	$3.5 \times 10^{-8}(T)$	$1.5 \times 10^{-8}(T)$
Layer 4	$1.8 \times 10^{-7}(T)$	$1.5 \times 10^{-6}(T)$	$5.1 \times 10^{-5}(T)$	$6.7 \times 10^{-4}(M)$	$1.9 \times 10^{-6}(T)$	$6.7 \times 10^{-4}(M)$
Lora	$9.5 \times 10^{-8}(T)$	$8.0 \times 10^{-6}(T)$	$5.1 \times 10^{-4}(T)$	$6.7 \times 10^{-4}(M)$	$9.4 \times 10^{-7}(T)$	$6.7 \times 10^{-4}(M)$
Decoder Only	$8.2 \times 10^{-6}(T)$	$5.3 \times 10^{-3}(T)$	$2.2 \times 10^{-1}(T)$	$6.7 \times 10^{-4}(M)$	$7.2 \times 10^{-4}(T)$	$6.7 \times 10^{-4}(M)$
Embeding	$2.0 \times 10^{-3}(T)$	$4.9 \times 10^{-2}(T)$	$6.7 \times 10^{-1}(T)$	$6.7 \times 10^{-4}(M)$	$6.4 \times 10^{-2}(T)$	$6.7 \times 10^{-4}(M)$
Input Adapt	$6.2 \times 10^{-3}(T)$	$8.2 \times 10^{-2}(T)$	$4.2 \times 10^{-1}(T)$	$6.7 \times 10^{-4}(M)$	$2.3 \times 10^{-1}(T)$	$6.7 \times 10^{-4}(M)$
Prefix	$4.8 \times 10^{-4}(T)$	$7.2 \times 10^{-3}(T)$	$1.0 \times 10^{0}(T)$	$6.7 \times 10^{-4}(M)$	$1.9 \times 10^{-2}(T)$	$6.7 \times 10^{-4}(M)$
Scratch	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$
Zero Shot	$4.1 \times 10^{-7}(T)$	$1.3 \times 10^{-9}(T)$	$1.8 \times 10^{-9}(T)$	$2.7 \times 10^{-3}(M)$	$0.0(T)$	$2.4 \times 10^{-4}(T)$
	P-Values comparing against Scratch Performance‡					
Full Model	$1.5 \times 10^{-3}(T)$	$3.2 \times 10^{-8}(T)$	$1.3 \times 10^{-2}(T)$	$3.7 \times 10^{-7}(T)$	$9.3 \times 10^{-8}(T)$	$1.3 \times 10^{-7}(T)$
Layer 4	$2.7 \times 10^{-1}(T)$	$1.1 \times 10^{-6}(T)$	$2.6 \times 10^{-3}(T)$	$2.9 \times 10^{-6}(T)$	$2.3 \times 10^{-7}(T)$	$2.7 \times 10^{-6}(T)$
Lora	$5.6 \times 10^{-3}(T)$	$1.5 \times 10^{-4}(T)$	$3.6 \times 10^{-3}(T)$	$7.3 \times 10^{-6}(T)$	$1.3 \times 10^{-6}(T)$	$1.2 \times 10^{-6}(T)$
Decoder Only	$3.8 \times 10^{-1}(T)$	$2.1 \times 10^{-4}(T)$	$7.5 \times 10^{-4}(T)$	$5.2 \times 10^{-5}(T)$	$2.3 \times 10^{-4}(T)$	$1.5 \times 10^{-6}(T)$
Embeding	$9.8 \times 10^{-7}(T)$	$4.1 \times 10^{-1}(T)$	$6.7 \times 10^{-4}(M)$	$3.6 \times 10^{-4}(T)$	$1.0 \times 10^{-1}(T)$	$3.7 \times 10^{-4}(T)$
Input Adapt	$1.5 \times 10^{-6}(T)$	$5.9 \times 10^{-1}(T)$	$5.7 \times 10^{-4}(T)$	$2.0 \times 10^{-4}(T)$	$2.1 \times 10^{-1}(T)$	$7.7 \times 10^{-5}(T)$
Prefix	$1.3 \times 10^{-5}(T)$	$1.3 \times 10^{-2}(T)$	$1.2 \times 10^{-3}(T)$	$2.2 \times 10^{-4}(T)$	$2.3 \times 10^{-3}(T)$	$1.2 \times 10^{-5}(T)$
Scratch	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$
Zero Shot	$0.0(T)$	$5.6 \times 10^{-7}(T)$	$0.0(T)$	$2.0 \times 10^{-2}(T)$	$8.8 \times 10^{-8}(T)$	$8.3 \times 10^{-3}(T)$

Table C.4: Obstacle Avoidance Cumulative Rewards for each testing morphology. Values show mean (top) and standard deviation (bottom). † statistical significance compared to Zero Shot and ‡ statistical significance to Scratch ($p < 0.01$). P-values and the hypothesis test run (T: t-test, M: Mann-Whitney) comparing against Zero shot and Scratch results.

Morphology	1	2	3	4	5	6
Full Model	$2652.41^{\dagger\ddagger}$	$3101.42^{\dagger\ddagger}$	$1705.64^{\dagger}$	$3577.09^{\dagger\ddagger}$	$3219.75^{\dagger\ddagger}$	$3558.26^{\dagger\ddagger}$
	± 193.57	± 177.17	± 4.16	± 341.74	± 199.13	± 365.21
Layer 4	$2246.88^{\dagger}$	$2684.70^{\dagger\ddagger}$	$1592.29^{\dagger\ddagger}$	$3276.76^{\dagger\ddagger}$	$2888.34^{\dagger\ddagger}$	$3194.19^{\ddagger}$
	± 184.03	± 85.63	± 140.05	± 314.17	± 64.04	± 351.87
Lora	$2137.75^{\dagger\ddagger}$	$2585.71^{\dagger\ddagger}$	$1672.75^{\dagger\ddagger}$	$3191.40^{\dagger\ddagger}$	$2851.48^{\dagger\ddagger}$	$3189.01^{\ddagger}$
	± 116.21	± 191.00	± 12.63	± 320.51	± 107.16	± 319.76
Decoder Only	$2263.74^{\dagger}$	$2531.13^{\dagger\ddagger}$	$1456.02^{\dagger\ddagger}$	$3061.26^{\ddagger}$	$2672.06^{\dagger\ddagger}$	$3132.18^{\ddagger}$
	± 186.99	± 161.72	± 218.88	± 360.37	± 160.55	± 302.67
Embeding	$1863.25^{\dagger\ddagger}$	$2189.46^{\dagger}$	$1556.72^{\dagger\ddagger}$	$2882.24^{\ddagger}$	$2398.29^{\dagger}$	$2877.35^{\ddagger}$
	± 94.00	± 167.27	± 151.65	± 361.86	± 139.50	± 417.09
Input Adapt	$1839.40^{\dagger\ddagger}$	$2159.73^{\dagger}$	$1458.49^{\dagger\ddagger}$	$2929.91^{\ddagger}$	$2367.39^{\dagger}$	$2833.04^{\ddagger}$
	± 117.50	± 125.84	± 206.98	± 356.45	± 142.90	± 312.07
Prefix	$1841.45^{\dagger\ddagger}$	$2334.01^{\dagger}$	$1514.42^{\dagger\ddagger}$	$2877.43^{\ddagger}$	$2538.31^{\dagger\ddagger}$	$2935.63^{\ddagger}$
	± 142.33	± 133.00	± 186.34	± 324.42	± 119.05	± 293.07
Scratch	$2334.75^{\dagger}$	$2119.16^{\dagger}$	$1843.23^{\dagger}$	2112.34	$2265.47^{\dagger}$	$2144.60^{\dagger}$
	± 92.45	± 124.11	± 99.74	± 216.38	± 124.23	± 123.50
Zero Shot	1300.21	1184.87	332.64	2467.45	1295.92	2488.64
	± 117.01	± 248.07	± 114.42	± 246.89	± 202.99	± 274.84
	P-Values comparing against Zeroshot Performance†					
Full Model	$2.2 \times 10^{-6}(T)$	$1.5 \times 10^{-6}(T)$	$9.7 \times 10^{-9}(T)$	$7.6 \times 10^{-4}(T)$	$8.5 \times 10^{-7}(T)$	$1.6 \times 10^{-3}(T)$
Layer 4	$2.4 \times 10^{-5}(T)$	$3.1 \times 10^{-6}(T)$	$6.8 \times 10^{-7}(T)$	$3.7 \times 10^{-3}(T)$	$3.9 \times 10^{-7}(T)$	$1.3 \times 10^{-2}(T)$
Lora	$7.6 \times 10^{-6}(T)$	$1.9 \times 10^{-5}(T)$	$1.2 \times 10^{-8}(T)$	$7.2 \times 10^{-3}(T)$	$8.4 \times 10^{-7}(T)$	$1.1 \times 10^{-2}(T)$
Decoder Only	$2.3 \times 10^{-5}(T)$	$1.7 \times 10^{-5}(T)$	$1.7 \times 10^{-5}(T)$	$2.6 \times 10^{-2}(T)$	$5.4 \times 10^{-6}(T)$	$1.4 \times 10^{-2}(T)$
Embeding	$6.9 \times 10^{-5}(T)$	$1.5 \times 10^{-4}(T)$	$7.9 \times 10^{-3}(M)$	$9.5 \times 10^{-2}(T)$	$1.9 \times 10^{-5}(T)$	$1.6 \times 10^{-1}(T)$
Input Adapt	$1.9 \times 10^{-4}(T)$	$1.1 \times 10^{-4}(T)$	$1.2 \times 10^{-5}(T)$	$6.5 \times 10^{-2}(T)$	$2.5 \times 10^{-5}(T)$	$1.4 \times 10^{-1}(T)$
Prefix	$8.9 \times 10^{-4}(T)$	$3.8 \times 10^{-5}(T)$	$4.7 \times 10^{-6}(T)$	$7.9 \times 10^{-2}(T)$	$5.6 \times 10^{-6}(T)$	$5.7 \times 10^{-2}(T)$
Scratch	$0.0(T)$	$5.6 \times 10^{-7}(T)$	$0.0(T)$	$2.0 \times 10^{-2}(T)$	$8.8 \times 10^{-8}(T)$	$8.3 \times 10^{-3}(T)$
Zero Shot	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$
	P-Values comparing against Scratch Performance‡					
Full Model	$1.5 \times 10^{-3}(T)$	$3.2 \times 10^{-8}(T)$	$1.3 \times 10^{-2}(T)$	$3.7 \times 10^{-7}(T)$	$9.3 \times 10^{-8}(T)$	$1.3 \times 10^{-7}(T)$
Layer 4	$2.7 \times 10^{-1}(T)$	$1.1 \times 10^{-6}(T)$	$2.6 \times 10^{-3}(T)$	$2.9 \times 10^{-6}(T)$	$2.3 \times 10^{-7}(T)$	$2.7 \times 10^{-6}(T)$
Lora	$5.6 \times 10^{-3}(T)$	$1.5 \times 10^{-4}(T)$	$3.6 \times 10^{-3}(T)$	$7.3 \times 10^{-6}(T)$	$1.3 \times 10^{-6}(T)$	$1.2 \times 10^{-6}(T)$
Decoder Only	$3.8 \times 10^{-1}(T)$	$2.1 \times 10^{-4}(T)$	$7.5 \times 10^{-4}(T)$	$5.2 \times 10^{-5}(T)$	$2.3 \times 10^{-4}(T)$	$1.5 \times 10^{-6}(T)$
Embeding	$9.8 \times 10^{-7}(T)$	$4.1 \times 10^{-1}(T)$	$6.7 \times 10^{-4}(M)$	$3.6 \times 10^{-4}(T)$	$1.0 \times 10^{-1}(T)$	$3.7 \times 10^{-4}(T)$
Input Adapt	$1.5 \times 10^{-6}(T)$	$5.9 \times 10^{-1}(T)$	$5.7 \times 10^{-4}(T)$	$2.0 \times 10^{-4}(T)$	$2.1 \times 10^{-1}(T)$	$7.7 \times 10^{-5}(T)$
Prefix	$1.3 \times 10^{-5}(T)$	$1.3 \times 10^{-2}(T)$	$1.2 \times 10^{-3}(T)$	$2.2 \times 10^{-4}(T)$	$2.3 \times 10^{-3}(T)$	$1.2 \times 10^{-5}(T)$
Scratch	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$	$1.0(T)$
Zero Shot	$0.0(T)$	$5.6 \times 10^{-7}(T)$	$0.0(T)$	$2.0 \times 10^{-2}(T)$	$8.8 \times 10^{-8}(T)$	$8.3 \times 10^{-3}(T)$

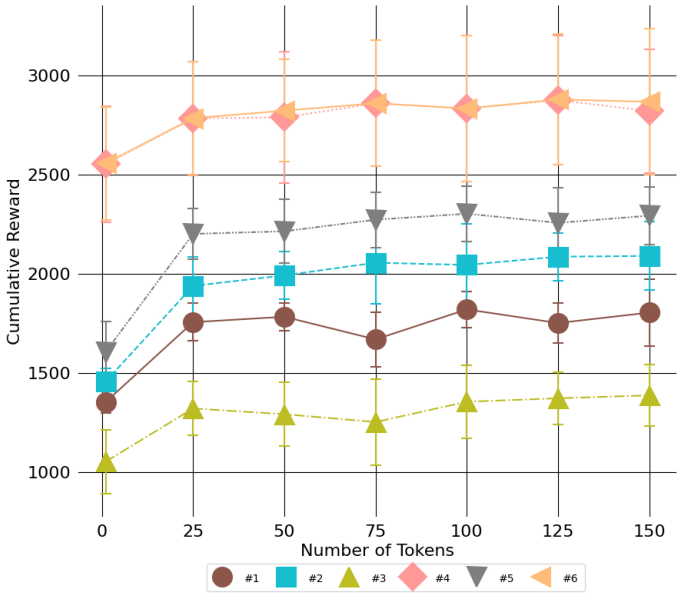


(a) Number of randomly initialized prefix tokens

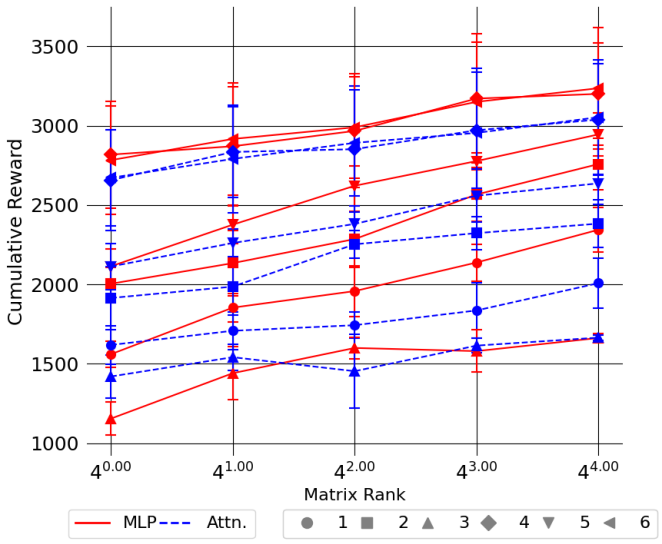


(b) Lora in different layers of fifth transformer block.

Figure C.3: Ablation studies on number of prefix tokens and LoRA in flat terrain task.



(a) Number of randomly initialized prefix tokens



(b) Lora in different layers of fifth transformer block.

Figure C.4: Ablation studies on number of prefix tokens and LoRA in obstacle avoidance task.