Actuation Subspace Prediction with Neural Householder Transforms

by

Kerrick Johnstonbaugh

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Choosing an appropriate action representation is an integral part of solving robotic manipulation problems. Published approaches include latent action models, which train context-conditioned neural networks to map low-dimensional latent actions to high-dimensional actuation commands. Such models can have a large number of parameters and can be difficult to interpret from a user's perspective. In this thesis, we propose that similar performance gains in robotics tasks can be achieved by restructuring the neural network to map observations to a *basis* for a context-dependent linear actuation subspace. This results in an action interface wherein a user's actions determine a linear combination of state-conditioned actuation basis vectors. We introduce the Neural Householder Transform (NHT) as a method for computing this basis. This thesis describes the development of NHT: from computing an unconstrained basis for a state-conditioned linear map (SCL), to computing an orthonormal basis that changes smoothly with respect to the input context. Two teleoperation user studies indicated that participants preferred using SCL, and tend to achieve higher completion rates with SCL compared to latent action models. In addition, simulation results showed that reinforcement learning agents trained with NHT in kinematic manipulation and locomotion environments tend to be more robust to hyperparameter choice and achieve higher final success rates compared to agents trained with alternative action representations.

Preface

In my first summer at UofA, 2021, I worked closely with Michael Przystupa on SCL. Michael conceived State Conditioned Linear Maps - SCL. I contributed the Gram-Schmidt orthogonalization and the idea of a numerical methods projection (actuation projection) for actions instead of a separate neural network. Laura Petrich instructed us on how to use the Kinova Gen3 Lite robot arm in Aug 2021, and gave us sample code for joint velocity control. Michael and I both contributed equally to the SCL experiments in 2021. Zichen Zhang and Masood Dehghan assisted in the proof of the soft reversibility property of SCL. In my 2022 winter, summer and fall RA, I conceived Neural Household Transforms - NHT, and led the work, with remote assistance from Michael and Jacob Keller (Michael was on an internship in Montreal and Jacob is a graduate student at UC San Diego). In fall of 2022 I designed and implemented the second robot user study myself, with assistance from Faezeh Haghverd and Zichen Zhang in data collection during subject trials. Michael assisted remotely with writing, planning and logistical problem solving.

SCL is submitted with the two robot user studies in Chapter 5 as M. Przystupa, K. Johnstonbaugh, Z. Zhang, L. Petrich, M. Dehghan, F. Haghverd, M. Jagersand, "Learning state conditioned linear mappings for low-dimensional control of robotic manipulators," in *2023 IEEE International Conference on Robotics and Automation (ICRA 2023)*.

NHT is submitted with the reinforcement learning experiments described in Chapter 6 as K. Johnstonbaugh, M. Przystupa, J. Keller, and M. Jagersand, "Contextual subspace approximation with neural householder transforms," in *11th International Conference on Learning Representations, ICLR 2023*.

The thesis is entirely my own conception and writing. All user studies were approved under UofA ethics permit Pro00054665.

Acknowledgements

First and foremost, I would like to thank my advisor, Martin Jagersand, for his consistent support and insightful suggestions. Professor Jagersand's emphasis on geometric reasoning provided inspiration for the creation of NHT.

Secondly, I would like to acknowledge the significant contribution of Michael Przystupa to the work presented in this thesis. Michael invented the state-conditioned linear map (SCL) approach to teleoperation. This thesis contains work collected from over a year of collaboration between Michael and myself.

I would also like to thank Jacob Keller, who suggested the usage of exponential maps in NHT. His assistance in proving the Lipschitz continuity of NHT was invaluable.

Finally, I would like to thank my partner, Callie Muslimani, whose support I could not live without. Every single one of my days is made better thanks to you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis addresses the problem of learning task-specific mappings that serve as an interface between low-dimensional continuous inputs and high-dimensional robotic actuation commands. Such mappings can be used to enable users to control high degree-of-freedom robots with a simple control interface like a joystick. For autonomous agents learning to control robots, task-specific action mappings can constrain robot motions such that dangerous or irrelevant movements become impossible.

The motivation for learning these action mappings is primarily to compute an intuitive and useful interface with which individuals with upper extremity disabilities can control assistive robotic manipulators (e.g., using a 2-DOF joystick). While most activities of daily living, such as opening doors or eating, are trivial or mundane for able-bodied individuals, they can be frustrating or impossible for individuals with upper body disabilities. Wheelchair-mounted assistive robotic manipulators, such as the Kinova Jaco, have been developed as a way to bring more autonomy to such individuals in their daily lives. Unfortunately, the industry standard for control of these assistive devices has been identified in several studies as problematic in terms of both cognitive load, and the time it takes to perform activities of daily living [26], [51]. The difficult control problem is undoubtedly a factor in these assistive devices still being far from widespread, despite their availability for over a decade.

A secondary motivation for learning task-specific action mappings is as a means to impart a higher degree of safety during autonomous learning for

robotics. By limiting the range of motions afforded by these mappings, we can in certain situations ensure that dangerous or unhelpful actions are never available to an autonomous agent. This is of particular importance for reinforcement learning on physical robots, where an errant exploratory action has the potential to damage nearby objects, the robot itself, or injure nearby humans.

The problem of learning mappings from low-dimensional actions to high-dimensional robotic actuation has been previously formulated in the literature as learning low-dimensional representations of robotic actuations; essentially framing the problem as a dimensionality reduction problem. Typically, demonstrations of a particular task are provided by a human. The high-dimensional actuations used to perform the task are recorded, and then compressed to some lower-dimensional representation. The action mapping is then taken as the function that reconstructs the high-dimensional actuations from the compressed representation.

The two most common approaches to robotic actuation representation in the literature are **movement primitives** and **latent action models**. The study of movement primitives has been extensive and typically represents robotic actuation as some form of a stable dynamical system. The movement primitive literature is primarily focused on autonomous robotics, with comparably less work related to teleoperation [13], [54]. Latent action models, on the other hand, have been studied in the context of both teleoperation for assistive robotics, and as action representations for reinforcement learning agents [43]. The representation space (i.e., the bottleneck of an autoencoder) of latent action models naturally acts as an interface for control. However, while these autoencoders may converge to a local minimum in the reconstruction loss, the relationship between the low-dimensional latent space and the corresponding high-dimensional outputs may be quite unintuitive to a human user.

A third, relatively restrictive approach is action representation via principal component analysis (PCA) or singular value decomposition (SVD) [7]. The principal components of actuation commands in a dataset of demonstrations

can be computed and used as a basis for teleoperation. The linear structure of the space spanned by the principal components affords intuitive control and generalization. This approach is straightforward and easy to implement, but it may not be possible to represent every possible desirable actuation (for a given task) with only a small number (2-3) of principal components.

While the application of dimensionality reduction methods has been promising, one of the contributions of this thesis is to propose a more restrictive problem formulation: *actuation subspace prediction*. Actuation subspace prediction (ASP) has a well-defined objective and enforces a locally-linear reconstruction of actuation commands. The objective of actuation subspace prediction is to estimate, at any given moment, the optimal actuation subspace corresponding to the model's current context. We introduce the concept of the optimal actuation subspace in section 4.1. Informally, given the **context** of the current situation, the optimal actuation subspace is a k-dimensional linear subspace that best captures the n-dimensional actuations observed in similar contexts (where $n > k$).

The solution to ASP presented in this thesis takes the form of the Neural Householder Transform (NHT). Like the autoencoders of latent action models, NHT leverages neural networks as function approximators. However, rather than learning to compress observed actuations (as autoencoders do), NHT produces an orthonormal actuation basis corresponding to the current context. As an additional benefit, the basis vectors produced by NHT are smooth with respect to changes in the observation[1].

NHT can be leveraged to learn an interface for control. After optimization of NHT with an ASP loss, users can control a robot by choosing weighted combinations of the actuation bases output by NHT. As the situation/context evolves, so too do the actuation bases available to the operator. Applications of NHT for control explored in this thesis include 1) teleoperation with an NHT interface and 2) control by a reinforcement learning agent with an NHT interface.

[1] given that we train the associate neural network with Lipschitz regularization

1.1 Contributions and Contents

The remainder of this thesis is organized as follows. Chapter 2, Literature Review, surveys the literature related to teleoperation in assistive robotics, learning from demonstration, and reinforcement learning from offline demonstrations.

Chapter 3, Background, first covers Markov Decision Processes, a framework for dynamics necessary to formulate actuation subspace prediction. It then covers autoencoders, which have been utilized in the latent action framework approach to low-dimensional actuation interfaces. Next, section 3.6 describes householder reflections and how they have typically been used to compute the QR decomposition in numerical computing software. Finally, section 3.7 discusses the concept of the exponential map on Riemannian manifolds, a function used to guarantee that NHT is smooth with respect to changing contexts.

Chapter 4, Methods and Algorithms, begins by more precisely formulating actuation subspace prediction, and defining the objective. It then discusses the primary contribution of this thesis, the Neural Householder Transform, and develops the algorithm from the perspective of latent action models. Section 4.3.3 discusses the smoothness of NHT and presents a theorem that establishes its Lipschitz continuity.

Chapter 5, Teleoperation with SCL, begins with a set of synthetic experiments designed to probe the teleoperation properties of our action mapping approach. Next, section 5.2 presents the results from two user studies in which participants used SCL (an early form of NHT) to perform various robotic manipulation tasks using a 2-DOF joystick.

Chapter 6, Reinforcement Learning with NHT, describes a set of experiments in which reinforcement learning agents learn to interact with an NHT interface, and compares learning with NHT to learning in the high-dimensional actuation space. It then goes on to present the results of two ablation studies and analyzes the empirical effect of Lipschitz regularization with NHT.

Chapter 7, Conclusions, completes the thesis by summarizing the presented

work. Limitations and promising directions for future work are also discussed.

Finally, the appendix includes the complete proof for the Lipschitz continuity of NHT.

Chapter 2

Literature Review

In this chapter, we discuss branches of the robot learning and teleoperation literature that relate to the content presented in this thesis. We then identify the gap in this literature that the work in this thesis addresses. Table 2.1 summarizes the taxonomy of the literature related to NHT, with a focus on learning action representations and control policies.

2.1 Action Representation Learning

In real-world applications of reinforcement learning, it is imperative to choose appropriate representations when defining the Markov decision process. The consequences of poor design decisions can have adverse effects in domains like robotics, where safety [58] and sample efficiency [39] are desirable properties. Typically these properties can be captured by choice of action space. Choices of robot action types distinct from basic joint motor control, such as Cartesian control or impedance control, have been shown to influence the efficiency of robotic learning, depending on the task [44].

Researchers have typically focused on learning action representations that can capture a variety of robotic motions. This interest has led to the development of several different action representation frameworks. *Action maps* that transform low DOF inputs directly to relevant high-dimensional robotic commands have been explored both to reduce cognitive strain during teleoperation and to increase the learning efficiency of artificial agents [15], [18], [42]. The assumption is that joints work together in the high-dimensional ambi-

Literature/Problem	Approach	Examples
Action Representation Learning	Linear	Artemiadis et al. [7], Matrone et al. [45], Odest et al. [48], Santello et al. [53]
	Nonlinear	Allshire et al. [4], Li et al. [40], Losey et al. [43]
Learning from Demonstration (LfD)	Regression Based	Calinon et al. [14], K.-Zadeh et al. [32], Schaal et al. [54], Kober et al. [35]
	Behavioral Cloning	Bain et al. [9], Ross et al. [50], Yang et al. [60]
RL from Demonstration	Offline RL	Agarwal et al. [3], Kumar et al. [36], Zhou et al. [62]
	Inverse RL	Arora et al. [6], Jarboui et al. [28]

Table 2.1: Summary of literature related to learning action representations and policies from offline demonstrations.

ent space to produce concerted motions that lie on some lower-dimensional manifold. Previous research on hand poses – a high-DOF setting – supports this assumption, finding that principal component analysis (PCA) [21], [27] can capture 80% of configurations with only two principal components [53]. Authors have applied this result to develop linear action map teleoperation grasping control schemes [7], [45], [48] and planning algorithms [17]. Linear maps are advantageous because they are easy to analyze and provide intuitive mappings for teleoperation.

However, globally linear approaches assume all task-relevant high-DOF commands exist in a single low-dimensional subspace and need more dimensions for precision movements [7]. Other research has investigated nonlinear action mappings as an alternative. These mappings are typically learned with a conditional neural autoencoder (CAE) [42], [43]. Instead of a single linear map, low-dimensional actions are input to a state-dependent neural decoder

to predict the corresponding high-dimensional motion. Assistive robotic research has demonstrated that CAE models enable higher success rates and faster completion times in assistive eating tasks compared to alternative systems [30], [31], [40], [42], [43]. However, CAEs are difficult to analyze because of their nonlinear structure. Authors have relied on engineering auxiliary loss terms to implicitly add desirable user-control properties [4], [40], but still require empirical evaluation to verify enforcement on deployment.

2.2 Teleoperation of Robotic Manipulators

Teleoperation, also called remote operation, is the control of a machine or device from a distance. Teleoperation of robotic manipulators has a variety of applications, from manipulation in deep sea and space environments [56], [61], to control of assistive robotic manipulators in the homes of disabled individuals [47].

Robotic manipulators typically include 6 to 7 independent degrees-of-freedom. Simultaneous control of six degrees-of-freedom can be accomplished with a human using both hands to manipulate two independent 3-DoF joysticks [46]. Leader-Follower (or Master-Slave) haptic telemanipulation is an alternative approach that can enable 6-DoF control. In haptic teleoperation, the operator performs the manipulation task with their arm coupled to an exoskeleton [29] or manipulates a haptic controller [16]. The remote manipulator mirrors the motions performed by the operator.

While the teleoperation modes discussed above can offer simultaneous one-to-one control for the independent DoFs of a manipulator, they can be either cognitively strenuous (two joysticks), or prohibitively expensive (exoskeleton). In some cases, for example in assistive robotics, it is desirable to achieve effective control using an interface with only two or three degrees of freedom. A straightforward and practically useful approach to this is mode switching. In mode switching, the operator typically cycles through three different control modes, controlling two-degrees of freedom at a time. In this way, all 6-dof of a manipulator can be accessed, although not simultaneously. Previous research

suggests that mode-switching can be cognitively strenuous, with basic tasks like opening a door or picking an object requiring 30-60 mode switches [26], [51].

2.3 Learning from Demonstration in Robotics

Much work has been focused on fully autonomous completion of manipulation tasks by robotic manipulators. These approaches have applications in assistive robotics, where in some situations a user may prefer to give no input whatsoever; they would rather allow the robot to execute the task with full autonomy. In contrast to industrial applications for robotics, assistive/field robots often operate in unstructured environments. In addition, a manufacturer cannot account for every possible use case or task in these environments. For these reasons learning from demonstration (LfD) has emerged as a popular approach to fully autonomous control [13], [24], [37].

2.3.1 Regression methods in LfD

A large body of literature exists in which robots learn skills from demonstration by fitting linear functions that map input features to actuations such as joint velocity, or force/torque commands [12]. These methods typically involve some form of locally weighted regression (LWR) [8], and use radial basis functions (RBFs) as the input features. These nonlinear features enable the model to learn functions that are locally linear, but globally non-linear with respect to the observed variables (e.g., joint configuration, object pose).

Gaussian Mixture Regression

Gaussian Mixture Regression (GMR) is an approach to learning from demonstration in which the relationship between input features and actuations are modeled jointly with a Gaussian Mixture Model (GMM), typically optimized with the expectation maximization (EM) algorithm [14]. Once the GMM is fitted to accurately model the joint distribution in the training demonstrations, the conditional distribution can be computed through GMR, enabling

online generation of novel trajectories. The stable estimator of dynamical systems (SEDS) algorithm relies on a constrained optimization procedure rather than EM to compute the parameters of the GMM to obtain an asymptotically stable dynamical system [32].

Movement Primitives

Another framework for learning from demonstration that often relies on locally weighted regression is movement primitives (MPs), in which entire trajectories are encoded as the action [49], [54]. Dynamical movement primitives have seen much success in robotics leading to impressive real-world experimental results by constraining the action space [35], [58]. Dynamical movement primitives essentially model the demonstration trajectories as spring-damper systems with a transient forcing term that is learned from the demonstration data and gives the trajectories their shape.

2.3.2 Behavioral Cloning

Artificial neural networks provide a mechanism for flexible non-linear function approximation that can be naturally leveraged for the problem of learning from demonstration. Behavioral cloning (sometimes also called imitation learning) is an approach to LfD that users supervised learning to train a neural network to map observations to actions [9], [50]. Recent work in behavioral cloning has shown that sub-optimal demonstrations can be leveraged to improve sample efficiency when cloning expert policies [60]. This work is noteworthy in that it learns a latent policy, similar to the latent action models discussed above.

2.4 Offline and Inverse Reinforcement Learning

Offline reinforcement learning is similar to behavior cloning in that both rely on offline datasets of transitions. Offline RL additionally requires transitions to be labeled with rewards. While in the best case behavioral cloning algorithms will copy the policy that generated the dataset, access to reward

labels enables offline RL algorithms to outperform the behavior policy [3], [38], without any additional (online) interaction with the environment. Offline RL algorithms can learn policies that outperform the behavior policy that generated the offline dataset; that is, offline RL can learn from suboptimal demonstration trajectories. On the other hand, the performance of behavioral cloning is typically eroded when suboptimal demonstrations are included in the training dataset.

However, offline RL can suffer from other issues such as when there is a discrepancy between the evaluation task and the provided demonstrations. Extrapolation errors due to out of distribution actions can cause erroneous Q-value estimates in dynamic programming-based offline RL algorithms [36]. Recent work argues that training a variational autoencoder to encode the actions in the offline dataset and then performing offline RL to learn a latent policy can mitigate this issue [62].

Inverse reinforcement learning (IRL), in contrast to offline RL, assumes that reward labels are not available in the demonstration data. Inverse RL methods seek to compute reward functions that could accurately describe the behavior observed in the demonstrations [6]. Given a dataset contains some expert demonstrations, it is possible to first use inverse RL to learn a reward function to label the dataset, and then perform offline RL [28].

2.5 Nonlinear Dimensionality Reduction

Nonlinear dimensionality reduction algorithms provide a means to model data in a high-dimensional ambient space as a lower-dimensional manifold. The two most popular non-linear dimensionality reduction algorithms are isomap [57] and locally linear embedding (LLE) [52]. Both algorithms rely on local relationships between datapoints. LLE explicitly models each datapoint as a linear combination of its k nearest neighbors. In this sense, the global data manifold constructed from LLE is locally linear. To the best of our knowledge, neither LLE nor isomap have been used to compute action representations for robotic control.

The neural householder transform (NHT) should not be considered a non-linear dimensionality reduction algorithm because, unlike isomap and LLE, NHT does not aim to construct a global non-linear manifold (see the swiss roll dataset [52]). Rather, given some context, NHT models the data encountered in the neighborhood of that context as existing on a linear subspace. As the context changes, the linear subspace also changes. However, there is no assumption that the data exists on some global non-linear manifold.

Chapter 3

Background

This section begins by explicitly discussing the notation used throughout the thesis, then goes on to briefly review Markov Decision Processes, autoencoders, the Latent Action Framework for teleoperation, Householder Reflections, QR Decomposition, and the Exponential Map to the Unit Sphere.

3.1 Notation

We denote matrices with upper case bold letters, and vectors with lower case bold letters. For example, a system of linear equations can be written as $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x} \in \mathbb{R}^n$ is an n-dimensional vector, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is an $m \times n$ matrix, and $\mathbf{b} \in \mathbb{R}^m$ is an m-dimensional vector. Likewise, general (non-linear) vector-valued functions are denoted with lowercase bold letters. For example, a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ mapping $\mathbb{R}^n$ to $\mathbb{R}^m$ could be written as $\mathbf{f} : \mathbf{x} \mapsto \mathbf{f}(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$. Unless otherwise indicated, all vectors are treated as column vectors. Row vectors are indicated by explicitly transposing column vectors.

3.2 Markov Decision Processes

The teleoperation studies presented in section 5.2 can be understood in the framework of Markov Decision Processes without reward, MDP/R [1]. However, chapter 6 involves reinforcement learning, and thus requires description of the full MDP.

An MDP is defined by a tuple $(\mathbb{S}, \mathbb{A}, T, p(\mathbf{s}_0), r, \gamma)$ where $\mathbb{S}$ is a continuous state space and $\mathbb{A}$ is a continuous action space. The transition probability operator $T(\mathbf{s}, \mathbf{a}, \mathbf{s}') : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0, 1]$ denotes the probability of transitioning to state $\mathbf{s}' \in \mathbb{S}$ when taking an action $\mathbf{a} \in \mathbb{A}$ from a state $\mathbf{s} \in \mathbb{S}$. If the transition function is deterministic, we write it as $\mathbf{s}' = T(\mathbf{s}, \mathbf{a})$. The reward function $r(\mathbf{s}, \mathbf{a})$ gives the scalar reward received after taking action $\mathbf{a}$ in state $\mathbf{s}$. The discount factor γ indicates preference for immediate vs. long-term reward, and $p(\mathbf{s}_0)$ is the initial state distribution.

3.3 Autoencoders

An autoencoder (AE) is an unsupervised learning model that attempts to summarize high-dimensional data $\mathbf{x} \in \mathbb{R}^m$ in some lower-dimensional space. They include an encoder $f(\mathbf{x}) = \mathbf{z}$ to compress the data, and a decoder to reconstruct the data $g(\mathbf{z}) = \hat{\mathbf{x}}$. Typically both f and g are neural networks with parameters (ϕ, θ) respectively (f_ϕ and g_θ throughout the thesis). A basic AE will be trained to reconstruct the data with mean squared error: $\min_{\phi,\theta} \mathbb{E}_{\mathbf{x}} \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$. Recent papers suggest AEs can be sufficient for learning mappings of low-dimensional actions to high-dimensional control [31].

3.4 Latent Action Framework

The latent actions framework assumes that the actuation commands produced by the optimal policy π^* exist on some lower-dimensional manifold. In latent action models, latent actions $\mathbf{z} \in \mathbb{R}^k$ are mapped to this manifold. These models have typically been studied in settings where there exists a dataset of transition tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$. We follow this paradigm and leave the study of learning latent action models online as future work, noting that some experiments along these lines have been previously reported [4].

Broadly, the class of models previously studied are conditional autoencoders. These models include a neural encoder $f_\theta(\mathbf{s}, \mathbf{a}) = \mathbf{z}$ which predicts the latent action. If the model is a variational CAE, then $f_\theta(\mathbf{s}, \mathbf{a}) = (\boldsymbol{\mu}, \boldsymbol{\sigma})$, and $\mathbf{z}$ is sampled from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ by the reparameterization trick [33]. These latent

actions are then reconstructed with a decoder $g_\theta(\mathbf{z}, \mathbf{s}) = \mathbf{a}$, where $\mathbf{s}$ is assumed to contextualize how the latent action $\mathbf{z}$ should map to the higher-dimensional space. In some works, there is also a latent transition model $T_\theta(\mathbf{z}, \mathbf{s}) = \mathbf{s}'$, which is trained to encourage the latent space to be predictive of transitions.

The most general loss function incorporating the above models is the following:

$$\arg\min_\theta L_{\text{recon}}(\mathbf{a}, \mathbf{s}, g_\theta, f_\theta) + \beta L_{\text{reg}}(\mathbf{s}, \mathbf{a}, f_\theta) + \alpha L_{\text{dyn}}(\mathbf{z}, \mathbf{s}, \mathbf{s}', T_\theta). \tag{3.1}$$

The first term L_{recon} is responsible for enforcing that the model's latent actions map to the original action space. The second term, L_{reg} incorporates all the terms that enforce additional requirements of the latent space. The typical choice are compression terms that pack the latent codes into some desired distribution which can include the Kullback-Leibler divergence, maximum mean discrepancy, or even the L2-norm of $\mathbf{z}$. The third term L_{dyn} is for training and utilizing the latent dynamics model, which is tasked with predicting the dynamics of observations, given the current observation and an action $\mathbf{z}$. The LASER model described by Allshire et al. [4] is an example of a latent action model that includes nonzero coefficients on all three terms of equation 3.1. The LASER model serves as a baseline in the reinforcement learning experiments described in section 6.

The notation above reflects the notation from the literature [40], [42], [43]. In later sections, we modify the notation to reflect that agents are selecting actions in the low-dimensional action space. Throughout the thesis we consider the low-dimensional action space to be k-dimensional, and the high-dimensional actuations $\mathbf{u}$ to be n-dimensional. We also deal with context vectors $\mathbf{c}$ for our action maps, rather than directly with states. We will then write the decoder as $g_\theta(\mathbf{a}, \mathbf{c}) = \mathbf{u}$.

3.5 Linear Least Squares

The linear least squares problem is fundamental in statistics and machine learning. The problem arises when one has a matrix $\mathbf{A} \in \mathbb{R}^{n \times k}$ and an n-vector

$\mathbf{b} \in \mathbb{R}^n$, and it is necessary to find the k-vector $\mathbf{x} \in \mathbb{R}^k$ that approximately solves the system

$$\mathbf{A}\mathbf{x} \cong \mathbf{b} \tag{3.2}$$

where $\mathbf{b}$ generally does not lie in the column space of $\mathbf{A}$. Because we often cannot express $\mathbf{b}$ as a linear combination of the columns of $\mathbf{A}$, there may not exist any $\mathbf{x} \in \mathbb{R}^k$ that solves equation 3.2 in the ordinary sense of equality. This is the reason we use the symbol $\cong$ when expressing the least squares problem. Since we may not be able to reduce the residual $\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b}$ to exactly the zero vector, the solution is given by a k-vector $\mathbf{x}$ that minimizes the Euclidean norm of the residual:

$$\min_{\mathbf{x}} \|\mathbf{r}\| = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\| \tag{3.3}$$

3.6 QR Decomposition and Householder Reflections

Every $\mathbf{A} \in \mathbb{R}^{n \times k}$ can be factorized as a product of an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix [59]. This is called the QR factorization of $\mathbf{A}$, and it can be written as:

$$\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix} \tag{3.4}$$

where $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is orthogonal, $\mathbf{R} \in \mathbb{R}^{k \times k}$ is upper triangular, and $\mathbf{O} \in \mathbb{R}^{(n-k) \times k}$ is the $(n-k) \times k$ zero matrix. Orthogonal matrices play an important role in the Neural Householder Transform, and here we will note a few properties of orthogonal matrices that will be immediately useful. Suppose $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is orthogonal. Then:

1. The columns of $\mathbf{Q}$ are all mutually orthogonal.
2. Each column of $\mathbf{Q}$ has unit norm.
3. $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}_n$, where $\mathbf{I}_n$ is the $n \times n$ identity matrix.
4. $\|\mathbf{Q}\mathbf{x}\| = \|\mathbf{x}\|$ for all $\mathbf{x} \in \mathbb{R}^{n \times n}$.

The QR factorization is most commonly used as a numerically robust approach to solving linear least squares problems [25].

$$\mathbf{r} = \mathbf{Q}\begin{bmatrix}\mathbf{R}\\ \mathbf{O}\end{bmatrix}\mathbf{x} - \mathbf{b} \tag{3.5}$$

Because $\mathbf{Q}^\top$ is orthogonal, it preserves the norm of any vector it transforms. Thus,

$$\|\mathbf{r}\| = \|\mathbf{Q}^\top\mathbf{r}\| = \|\mathbf{Q}^\top\left(\mathbf{Q}\begin{bmatrix}\mathbf{R}\\ \mathbf{O}\end{bmatrix}\mathbf{x} - \mathbf{b}\right)\| = \|\begin{bmatrix}\mathbf{R}\\ \mathbf{O}\end{bmatrix}\mathbf{x} - \mathbf{Q}^\top\mathbf{b}\| \tag{3.6}$$

Now, since $\mathbf{r}$ and $\mathbf{Q}^\top\mathbf{r}$ have the same norm, we can minimize $\|\mathbf{r}\|$ by minimizing the right hand side of equation 3.6. That is,

$$\min_{\mathbf{x}}\|\mathbf{A}\mathbf{x} - \mathbf{b}\| = \min_{\mathbf{x}}\|\mathbf{Q}^\top\mathbf{b} - \begin{bmatrix}\mathbf{R}\\ \mathbf{O}\end{bmatrix}\mathbf{x}\| \tag{3.7}$$

We can partition $\mathbf{Q}^\top\mathbf{b} \in \mathbb{R}^n$ as follows,

$$\mathbf{Q}^\top\mathbf{b} = \begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix} \tag{3.8}$$

such that $\mathbf{c}_1$ is a k-vector and $\mathbf{c}_2$ is an $n-k$-vector. Now the right hand side of equation 3.6 can be rewritten as:

$$\|\mathbf{Q}^\top\mathbf{b} - \begin{bmatrix}\mathbf{R}\\ \mathbf{O}\end{bmatrix}\mathbf{x}\| = \|\begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix} - \begin{bmatrix}\mathbf{R}\mathbf{x}\\ \mathbf{O}\end{bmatrix}\| = \|\begin{bmatrix}\mathbf{c}_1 - \mathbf{R}\mathbf{x}\\ \mathbf{c}_2\end{bmatrix}\| \tag{3.9}$$

By the definition of the Euclidean norm, we have:

$$\|\begin{bmatrix}\mathbf{c}_1 - \mathbf{R}\mathbf{x}\\ \mathbf{c}_2\end{bmatrix}\|^2 = \|\mathbf{c}_1 - \mathbf{R}\mathbf{x}\|^2 + \|\mathbf{c}_2\|^2 \tag{3.10}$$

Thus, we can reformulate the minimization in equation 3.7 as:

$$\min_{\mathbf{x}}\left(\|\mathbf{c}_1 - \mathbf{R}\mathbf{x}\|^2 + \|\mathbf{c}_2\|^2\right) \tag{3.11}$$

The $\|\mathbf{c}_2\|^2$ term does not depend on $\mathbf{x}$, so

$$\min_{\mathbf{x}}\|\mathbf{A}\mathbf{x} - \mathbf{b}\| = \min_{\mathbf{x}}\|\mathbf{c}_1 - \mathbf{R}\mathbf{x}\|^2. \tag{3.12}$$

And so minimization of equation 3.12 minimizes the norm of the residual $\mathbf{A}\mathbf{x} - \mathbf{b}$. Since $\mathbf{R}$ is upper triangular, the solution $\mathbf{x}$ to the linear system $\mathbf{R}\mathbf{x} = \mathbf{c}_1$ can easily be computed by back-substitution [25], resulting in the least squares solution to the overdetermined system $\mathbf{A}\mathbf{x} \cong \mathbf{b}$.

3.6.1 Householder Reflections

A common method to compute the QR factorization of a matrix $\mathbf{A} \in \mathbb{R}^{n \times k}$ constructs $\mathbf{Q}^\top$ by choosing a sequence of Householder reflections in order to successively reduce $\mathbf{A}$ to an upper triangular form:

$$\mathbf{H}_k \mathbf{H}_{k-1} \dots \mathbf{H}_1 \mathbf{A} = \mathbf{Q}^\top \mathbf{A} = \begin{bmatrix} \mathbf{R} \\ \mathbf{O} \end{bmatrix} \tag{3.13}$$

where $\mathbf{H}_i \in \mathbb{R}^{n \times n}$, $1 \leq i \leq k$ are Householder reflections, constructed from a corresponding sequence of n-vectors $\mathbf{v}_i$:

$$\mathbf{H}_i = \mathbf{I} - 2\frac{\mathbf{v}_i \mathbf{v}_i^\top}{\mathbf{v}_i^\top \mathbf{v}_i}. \tag{3.14}$$

The numerical methods literature contains ways to compute $\mathbf{v}_i$ to compute the QR decomposition [25]; however, we will not go into the details here. Instead, a contribution of this thesis is to train neural networks to map context vectors to these $\mathbf{v}_i$ such that the product of the corresponding Householder reflections form a basis for an optimal k-dimensional actuation subspace in the given context.

We do note, however, that these Householder reflections, also called Householder transformations, are symmetric and orthogonal. Hence $\mathbf{H}_i^{-1} = \mathbf{H}_i^\top = \mathbf{H}_i$. Following the notation from equation 3.13, $\mathbf{Q}$ can then be written as

$$\mathbf{Q} = \mathbf{H}_1 \mathbf{H}_2 \dots \mathbf{H}_k. \tag{3.15}$$

And thus both $\mathbf{Q}$ and $\mathbf{R}$ for the QR factorization can be obtained by selecting an appropriate sequence of Householder vectors $\mathbf{v}_i$.

3.7 Exponential Maps on Riemannian Manifolds

Riemannian geometry studies smooth manifolds with a Riemannian metric. This metric enables a formal notion of distance and angles on the manifold. While the formalization of this theory is outside the scope of this thesis (see [2] for a formal description), the aim of this subsection is to provide some informal intuition about exponential maps on Riemannian manifolds. In order

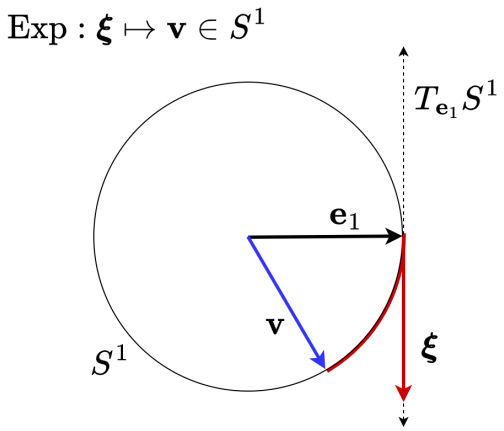


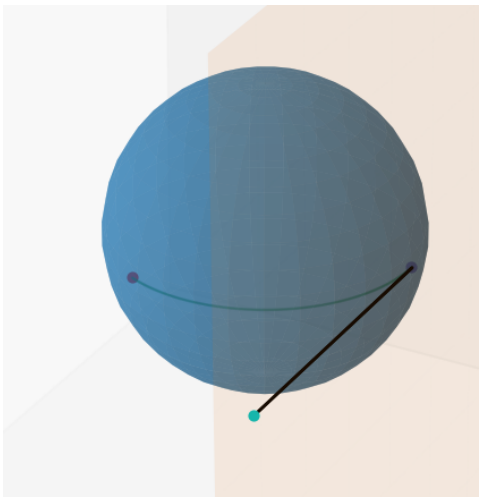
Figure 3.1: Illustration of exponential map on S^1, the unit sphere in $\mathbb{R}^2$. The tangent vector $\boldsymbol{\xi} \in \mathbb{R}^1$ indicates the direction and distance to travel along the surface of the manifold (here the manifold is a circle).

to perform actuation subspace prediction in a way that smoothly varies with respect to changes in context, NHT uses the exponential map on the unit sphere as part of its computation of actuation basis vectors.

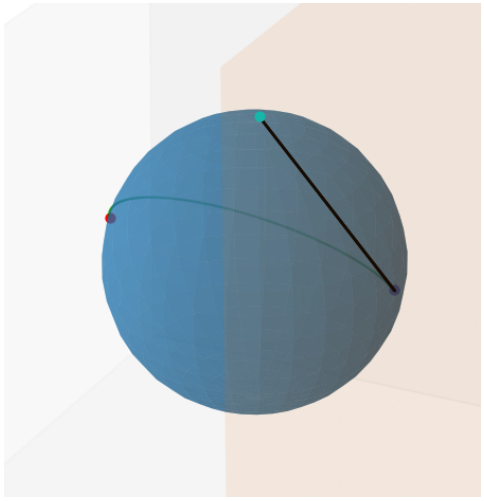
Every point $\mathbf{x}$ on a Riemannian manifold has a corresponding tangent space. The tangent space is a higher-dimensional generalization of tangent planes to surfaces in three dimensions or tangent lines to curves in two dimensions. The example illustration in figure 3.1 shows the tangent space on the unit circle S^1 at the point $\mathbf{e}_1$, denoted as $T_{\mathbf{e}_1} S^1$. The unit circle is a 1-dimensional surface, embedded in 2-dimensional space. The tangent space at $\mathbf{e}_1$ is a 1-dimensional vector space: any point in $T_{\mathbf{e}_1} S^1$ can be described by a single scalar, representing the direction and length of the vector. In general, the dimension of the tangent space is equal to the dimension of the surface, and one less than the dimension of the ambient space. For the unit sphere S^2, tangent spaces take the form of planes that touch the sphere at a single point. The exponential map on a manifold $\mathcal{M}$ at a point $\mathbf{x}$ is a function that maps vectors in the tangent space at $\mathbf{x}$ to the manifold itself. Informally, one can imagine the vector $\boldsymbol{\xi}$ in the tangent space $T_{\mathbf{x}}\mathcal{M}$ as indicating a direction and distance along which to travel on the surface of the manifold. The image of $\boldsymbol{\xi}$ under the exponential map $\text{Exp}_{\mathbf{x}}$ is the point reached starting at $\mathbf{x}$ and travelling on the manifold in the direction $\boldsymbol{\xi}$ for a distance of $\|\boldsymbol{\xi}\|$. The illustration

in figure 3.1 shows an example $\boldsymbol{\xi}$ and its image labeled $\mathbf{v}$ on a unit circle. The red arc has a length equal to $\|\boldsymbol{\xi}\|$, the length of $\boldsymbol{\xi}$. Figure 3.2 shows three example $\boldsymbol{\xi}$'s and their images on S^2, the unit sphere in three dimensions.

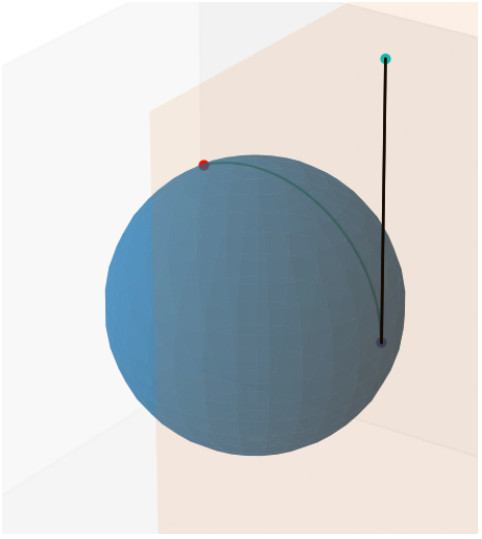
Note that for every $\boldsymbol{\xi}$ in the domain $T_{\mathbf{x}}\mathcal{M}$ of the exponential map, the image remains on the manifold. If the manifold $\mathcal{M} = S^{n-1}$, a unit sphere in n-dimensions, the exponential map sends any arbitrary $(n-1)$-dimensional tangent space vector to a **unit length** n-dimensional vector. This ability to map arbitrary vectors to vectors of unit norm in a smooth manner is the reason we use the exponential map as a component in NHT; these properties enable us to guarantee Lipschitz continuity (see section 4.3.3).



(a) $\boldsymbol{\xi} = [-2, 0]^\top$



(b) $\boldsymbol{\xi} = [-2, 2]^\top$



(c) $\boldsymbol{\xi} = [0, 2]^\top$

Figure 3.2: Examples of inputs $\boldsymbol{\xi}$ (cyan dot) and their images $\mathrm{Exp}_{\mathbf{e}_1}(\boldsymbol{\xi})$ (red dot on sphere) for the exponential map on S^2 (sphere in three dimensions) at $\mathbf{e}_1$.

Chapter 4

Methods and Algorithms: Neural Householder Transform

In this chapter, we discuss the general teleoperation paradigm of actuation subspace prediction. While actuation subspace prediction could be approached with several solution methods, we develop the Neural Householder Transform approach to actuation subspace prediction.

4.1 Problem Statement

We assume that the actuations we wish to model were observed in some *context*, and the resulting dataset is a collection of context-actuation pairs (actuations and context are both represented by vectors). We formulate the problem of **actuation subspace prediction** by supposing that, for every context $\mathbf{c}$, there exists an associated subspace that best approximates the actuations observed in the neighborhood of $\mathbf{c}$.

We use $\mathbf{x} = (\mathbf{c}, \mathbf{u})$ to denote a tuple consisting of an actuation $\mathbf{u}$ and the context $\mathbf{c}$ in which it was observed. For convenience, we define the following functions to extract the actuation and context vectors from a tuple $\mathbf{x}$, respectively: $\mathcal{C}(\mathbf{x}) = \mathbf{c}$; $\mathcal{U}(\mathbf{x}) = \mathbf{u}$. In addition, we denote the neighborhood of a context vector as $\mathcal{N}(\mathbf{c}) = \{\mathbf{c}' : \|\mathbf{c} - \mathbf{c}'\| < \delta\}$ for some $\delta \in \mathbb{R}$.

Definition 4.1.1 (Optimal Actuation Subspace)**.** We define $W^*(\mathbf{c})$, the optimal k-dimensional actuation subspace associated with context $\mathbf{c}$, as the k-dimensional subspace that minimizes the expected projection error of actua-

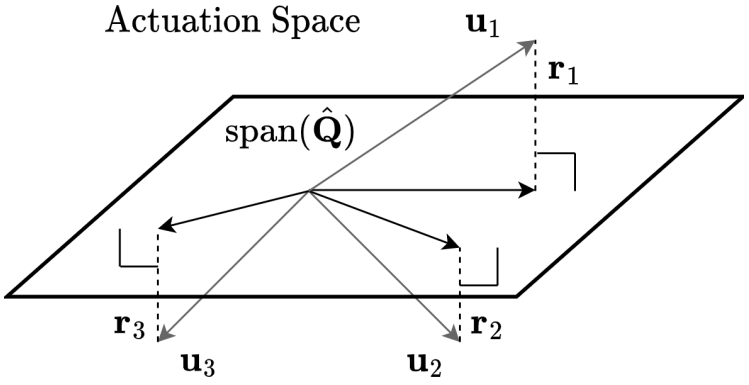


Figure 4.1: Diagram of actuation subspace.

tions observed in the neighborhood of $\mathbf{c}$:

$$W^*(\mathbf{c}) \doteq \arg\min_W \mathbb{E}_{\mathbf{x}|\mathcal{C}(\mathbf{x})\in\mathcal{N}(\mathbf{c})} \|\mathcal{U}(\mathbf{x}) - \text{proj}_W\left(\mathcal{U}(\mathbf{x})\right)\|_2^2 \tag{4.1}$$

where W is a k-dimensional linear subspace of $\mathbb{R}^n$, and $\text{proj}_W(\mathcal{U}(\mathbf{x}))$ is the orthogonal projection of the actuation $\mathbf{u}$ onto W.

Our goal is to approximate a function $\mathcal{Q}^*(\mathbf{c})$ that maps context vectors to an orthonormal basis for the associated optimal actuation subspace $W^*(\mathbf{c})$.

$$\mathcal{Q}^* : \mathbf{c} \mapsto \hat{\mathbf{Q}} \mid \text{col}(\hat{\mathbf{Q}}) = W^*(\mathbf{c}) \tag{4.2}$$

where $\hat{\mathbf{Q}} \in \mathbb{R}^{n\times k}$ is a $n \times k$ matrix of real numbers, and $\text{col}(\hat{\mathbf{Q}})$ is the column space of $\hat{\mathbf{Q}}$. We assume access to a dataset of actuation-context pairs. This dataset may take the form of demonstrations provided by a human, or some other expert control system.

4.2 State Conditioned Linear Maps

In this section, we describe our proposed alternative to the conditional autoencoder paradigm of latent action models. The goal is to compute a useful map from low-dimensional actions $\mathbf{a} \in \mathbb{R}^k$ to high-dimensional actuation commands (e.g. motor torques in a robotic manipulator) $\mathbf{u} \in \mathbb{R}^n$, where $n > k$.

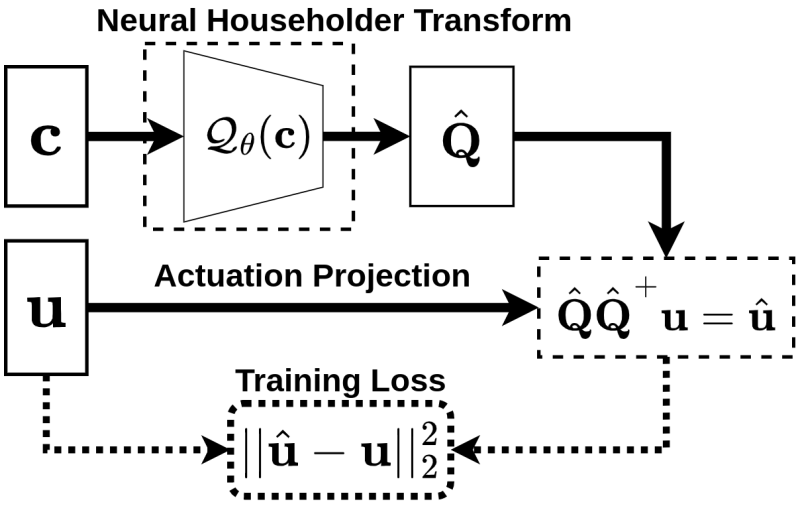


Figure 4.2: Training procedure for NHT. $\mathcal{Q}_\theta$ uses a neural network and Householder transformations to map a context vector to an $n \times k$ matrix $\hat{\mathbf{Q}}$ with orthonormal columns. The actuation $\mathbf{u}$ associated with context $\mathbf{c}$ is projected onto the column space of $\hat{\mathbf{Q}}$.

Our approach centers on optimizing a parameterized approximation of $\mathcal{Q}^*$ (see Equation 4.2).

First, let us consider using a linear map from the latent space to the actuation space. Instead of a non-linear function $g_\theta : \mathbf{a}, \mathbf{c} \mapsto \mathbf{u}$ that jointly maps context vectors and actions to the actuation space, we work with a non-linear function $\mathcal{Q}_\theta : \mathbf{c} \mapsto \hat{\mathbf{Q}}$ that maps context vectors to a matrix. The matrix $\hat{\mathbf{Q}} : \mathbf{a} \mapsto \mathbf{u}$ itself is a linear map from low-dimensional actions to high-dimensional actuation commands.

As $\hat{\mathbf{Q}}$ serves a similar purpose to g_θ (both map actions to actuation commands), we could consider $\hat{\mathbf{Q}}$ to be a linear decoder in the latent action framework. Then, the optimization of the standard reconstruction loss is formulated as follows:

$$\theta^* = \arg\min_\theta \mathbb{E}_\pi \|\mathbf{u} - \hat{\mathbf{Q}}\mathbf{a}\|_2^2 \tag{4.3}$$

Where $\hat{\mathbf{Q}} = \mathcal{Q}_\theta(\mathbf{c})$ is a function of the context $\mathbf{c}$. θ^* represents the optimal parameter vector for $\mathcal{Q}_\theta$.

The problem now becomes how to select the action $\mathbf{a} \in \mathbb{R}^k$ to use in this optimization. One approach is to follow the conditional autoencoder paradigm

and predict $\mathbf{a}$ with an encoder neural network. We opt instead to compute the optimal action $\mathbf{a}^*$, which we define as the action that minimizes equation (4.3) for fixed $\mathbf{u}$ and $\hat{\mathbf{Q}}$ when θ is held constant:

$$\mathbf{a}^* = \arg\min_{\mathbf{a}} \|\mathbf{u} - \hat{\mathbf{Q}}\mathbf{a}\|_2^2 \tag{4.4}$$

Finding $\mathbf{a}^*$ is a least squares problem, and the solution can be computed with the Moore-Penrose left pseudoinverse $\hat{\mathbf{Q}}^+ = (\hat{\mathbf{Q}}^\top \hat{\mathbf{Q}})^{-1}\hat{\mathbf{Q}}^\top$. The solution to (4.4) is given by $\mathbf{a}^* = \hat{\mathbf{Q}}^+\mathbf{u}$. Now our optimization problem becomes:

$$\theta^* = \arg\min_{\theta} \mathbb{E}||\mathbf{u} - \hat{\mathbf{Q}}\hat{\mathbf{Q}}^+\mathbf{u}||_2^2 \tag{4.5}$$

Note that the matrix $\hat{\mathbf{Q}}\hat{\mathbf{Q}}^+$ is an orthogonal projector onto span($\hat{\mathbf{Q}}$). Therefore, when we calculate $\hat{\mathbf{u}} = \hat{\mathbf{Q}}\hat{\mathbf{Q}}^+\mathbf{u}$, we are performing an orthogonal projection of $\mathbf{u}$ onto span($\hat{\mathbf{Q}}$). That is, $\hat{\mathbf{u}} = \text{proj}_{\text{span}(\hat{\mathbf{Q}})}(\mathbf{u})$.

Now it is clear that the solution to Equation (4.5) is the best approximation attainable by $\mathcal{Q}_\theta$ to the optimal actuation subspace $W^*(\mathbf{c})$ defined in Equation (4.1).

4.2.1 Gram-Schmidt SCL

It can be desirable for the matrix $\hat{\mathbf{Q}}$ produced by $\mathcal{Q}_\theta$ to have orthonormal columns. One reason is that $\hat{\mathbf{Q}}^+$ can be trivially computed as $\hat{\mathbf{Q}}^+ = \hat{\mathbf{Q}}^\top$, which is computationally cheaper to perform. Our experimental results also indicate that learning an $\mathcal{Q}_\theta$ that produces $\hat{\mathbf{Q}}$ with orthonormal columns tends to be more robust to hyperparameter choices (see Appendix). For these reasons we compute $\hat{\mathbf{Q}}$ by first computing an orthogonal matrix $\mathbf{Q}$, and then extracting the first k columns:

$$\mathbf{Q} = \begin{bmatrix} \hat{\mathbf{Q}} & \mathbf{Q}_\perp \end{bmatrix} \tag{4.6}$$

where $\mathbf{Q} \in \mathbb{R}^{n \times n}$, $\hat{\mathbf{Q}} \in \mathbb{R}^{n \times k}$, and $\mathbf{Q}_\perp \in \mathbb{R}^{n \times (n-k)}$. Note that we do not make use of $\mathbf{Q}_\perp$; it is only named to provide insight into the structure of $\mathbf{Q}$.

The Gram-Schmidt (GS) process is a straightforward way to compute an orthonormal basis that spans the same column space as an arbitrary matrix. Luckily, the Gram-Schmidt process is easy to implement, and differentiable. A

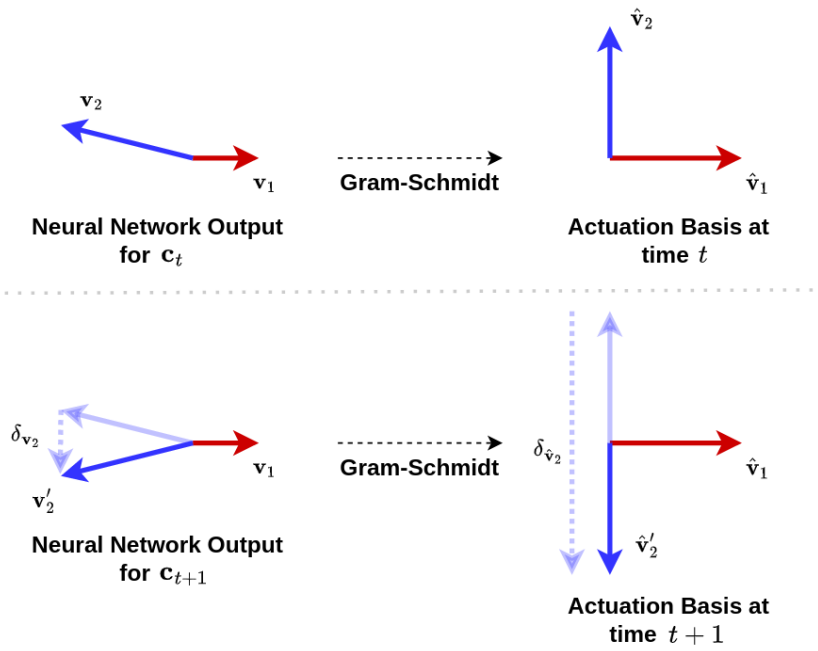


Figure 4.3: Illustration of how a small change in neural network output can cause a large change in actuation basis when using Gram-Schmidt orthogonalization.

Gram-Schmidt orthonormalized SCL can be easily implemented by coding a Gram-Schmidt routine and then relying on auto-differentiation software such as Pytorch or Tensorflow to pass gradients through the GS process during training of the state-conditioned linear map model.

However, there is an important issue with using the Gram-Schmidt process to obtain an orthonormal basis for a linear actuation subspace. The Gram-Schmidt process is not smooth with respect to changes in the input matrix. Small changes in the input matrix can cause abrupt flips of the basis vectors output by the GS process. From the perspective of a user that interacts with an SCL interface, this can cause serious disorientation or confusion. The motion of the robot could change abruptly at any given moment even when the user input remains unchanged. Figure 4.3 shows an illustration of this basis flipping effect. The next section describes an alternative method for computing a smoothly changing orthonormal basis from an arbitrary input vector.

4.3 Neural Householder Transform

We can obtain $n \times n$ orthogonal matrices by computing Householder transformations. However, in order to span an arbitrary k-dimensional subspace, we need to chain together k reflections:

$$\mathbf{Q} = \mathbf{H}(\mathbf{v}_1)\mathbf{H}(\mathbf{v}_2)\cdots\mathbf{H}(\mathbf{v}_k) \tag{4.7}$$

where $\mathbf{H} : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ computes the Householder matrix that reflects about the hyperplane orthogonal to $\mathbf{v}_i$:

$$\mathbf{H}(\mathbf{v}_i) = \mathbf{I} - 2\mathbf{v}_i\mathbf{v}_i^\top, \quad i \in \{1, ..., k\} \tag{4.8}$$

where each $\mathbf{v}_i$ has unit norm. Next, we describe how NHT uses a neural network to compute these $\mathbf{v}_i$ unit vectors.

4.3.1 Exponential Map on Unit Sphere

We would like to leverage neural networks to learn a map from contexts $\mathbf{c}$ to the $\mathbf{v}_i$ needed to compute $\hat{\mathbf{Q}}$. We can readily obtain unit $\mathbf{v}_i$ from the output of a typical neural network h_θ by simple normalization: $\mathbf{v}_i = h_\theta(\mathbf{c})/\|h_\theta(\mathbf{c})\|$. Unfortunately, this approach can result in unstable approximations. As the norm of $h_\theta(\mathbf{c})$ shrinks, arbitrarily small perturbations to the context can cause disproportionate changes in $\mathbf{v}_i$.

As a more stable alternative, we make use of the exponential map from Riemannian geometry [2], which maps points in the tangent space of a manifold to the manifold itself (in our case, the sphere). We seek unit vectors in $\mathbb{R}^n$, which lie on the (n-1)-sphere. We can therefore make use of the exponential map on $S^{(n-1)}$ at $\mathbf{e}_1$ (the first standard basis vector, $\mathbf{e}_1 = [1, 0, \ldots, 0]^\top$). The mapping

$$\text{Exp}_{\mathbf{e}_1} : \boldsymbol{\xi}_i \mapsto \mathbf{v}_i \tag{4.9}$$

maps[1] tangent vectors $\boldsymbol{\xi}_i \in \mathbb{R}^{(n-1)}$ to unit vectors $\mathbf{v} \in \mathbb{R}^n$. We require k tangent vectors $\boldsymbol{\xi}_i$ that will map to the $\mathbf{v}_i$ vectors used to compute $\mathbf{Q}$. We

[1] For the sphere, the exponential map at $\mathbf{e}_1$ is computed as $\mathbf{v}_i = \mathbf{e}_1\cos(\|\boldsymbol{\xi}_i\|) + \frac{1}{\|\boldsymbol{\xi}_i\|}\begin{bmatrix}0\\ \boldsymbol{\xi}_i\end{bmatrix}\sin(\|\boldsymbol{\xi}_i\|)$.

therefore configure the neural network h_θ to output a vector $\bar{\boldsymbol{\xi}} \in \mathbb{R}^{k(n-1)}$. We treat the output of h_θ as k stacked tangent vectors:

$$h_\theta : \mathbf{c} \mapsto \bar{\boldsymbol{\xi}}; \quad \bar{\boldsymbol{\xi}} = \left[\boldsymbol{\xi}_1^\top, \boldsymbol{\xi}_2^\top, \ldots, \boldsymbol{\xi}_k^\top\right]^\top \tag{4.10}$$

We then use the exponential map on each tangent vector, resulting in a vector $\bar{\mathbf{v}} \in \mathbb{R}^{nk}$ of stacked unit n-vectors:

$$\begin{bmatrix} \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_k \end{bmatrix} \overset{\text{Exp}}{\longmapsto} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_k \end{bmatrix} \tag{4.11}$$

where $\bar{\mathbf{v}} = \left[\mathbf{v}_1^\top, \mathbf{v}_2^\top, \ldots, \mathbf{v}_k^\top\right]^\top$. Each $\mathbf{v}_i$ is then used to compute a Householder matrix (Eq. 4.8), which are composed to obtain $\mathbf{Q}(\bar{\mathbf{v}})$ (Eq. 4.7). Overall, NHT ($\mathcal{Q}_\theta : \mathbf{c} \mapsto \hat{\mathbf{Q}}$) can be understood as the composition of each of these computations:

$$\underbrace{\mathbf{c} \overset{h_\theta}{\longmapsto} \bar{\boldsymbol{\xi}} \overset{\text{Exp}}{\longmapsto} \bar{\mathbf{v}} \overset{\mathbf{Q}}{\longmapsto} \mathbf{Q}(\bar{\mathbf{v}}) \mapsto \hat{\mathbf{Q}}(\bar{\mathbf{v}})}_{\text{NHT}} \tag{4.12}$$

where $\mathbf{c}$ is a context vector of arbitrary dimension, and $\mathbf{Q}(\bar{\mathbf{v}})$ is an $n \times n$ orthogonal matrix, and $\hat{\mathbf{Q}}(\bar{\mathbf{v}})$ is the matrix formed by the first k columns of $\mathbf{Q}(\bar{\mathbf{v}})$.

4.3.2 Existence

If we hope to use NHT to approximate arbitrary subspaces, it is important to ensure that for every k-dimensional subspace W of $\mathbb{R}^n$, there exists a vector $\bar{\mathbf{v}} \in \mathbb{R}^{nk}$ such that $W = \text{span}\left(\hat{\mathbf{Q}}(\bar{\mathbf{v}})\right)$.

Remark. *Let $W \subseteq \mathbb{R}^n$ be an arbitrary k-dimensional subspace. There is sequence of k Householder reflectors $\mathbf{Q} = \mathbf{H}_1\mathbf{H}_2\cdots\mathbf{H}_k$ such that the first k columns of $\mathbf{Q}$ are an orthonormal basis of W.*

Proof. Let M be an a $n \times k$ matrix whose column space is W. By trefethen1997numerical Algorithm 10.1 we can construct a QR decomposition, $M = QR$ where Q is the product of exactly k Householder reflections. Now we are done because it is a basic property of QR decompositions that the first

k columns of Q are an orthonormal basis for the column space of M, which is W. □

Thus, given the existence of an optimal contextual subspace $W^*(\mathbf{c})$, we can be sure that there exists some $\bar{\mathbf{v}}^*$ such that $\hat{\mathbf{Q}}(\bar{\mathbf{v}}^*)$ spans W^*. It is left to the neural network h_θ to approximate a set of tangent vectors $\bar{\boldsymbol{\xi}}$ that map to $\bar{\mathbf{v}}^*$, given $\mathbf{c}$.

4.3.3 Smoothness of $\mathcal{Q}_\theta$

We conjecture that low-dimensional action interfaces that change abruptly from state-to-state may degrade learning in RL agents. Thus we are interested in whether or not $\hat{\mathbf{Q}} = \mathcal{Q}_\theta(\mathbf{c})$ changes smoothly with respect to changes in the context. Concretely, we would like to limit the change in the high-dimensional actuation $\hat{\mathbf{u}}$ corresponding to an identical low-dimensional action $\mathbf{a}$ given that the change in the context is small. Let $\hat{\mathbf{Q}}_1 = \mathcal{Q}_\theta(\mathbf{c}_1)$ and $\hat{\mathbf{Q}}_2 = \mathcal{Q}_\theta(\mathbf{c}_2)$ for two nearby contexts, $\mathbf{c}_1$ and $\mathbf{c}_2$. Suppose an agent takes the same low-dimensional action $\mathbf{a}$ in both contexts. Denote the corresponding actuation commands as $\hat{\mathbf{u}}_1 = \hat{\mathbf{Q}}_1\mathbf{a}$ and $\hat{\mathbf{u}}_2 = \hat{\mathbf{Q}}_2\mathbf{a}$, respectively. We would like to limit the magnitude of the change in $\hat{\mathbf{u}}$ (i.e. $\|\hat{\mathbf{u}}_1 - \hat{\mathbf{u}}_2\|$) with respect to changes in context. That is, we would like to find a constant L such that:

$$\|\hat{\mathbf{u}}_1 - \hat{\mathbf{u}}_2\| \leq L\|\mathbf{c}_1 - \mathbf{c}_2\| \tag{4.13}$$

where $\|\cdot\|$ refers to the vector 2-norm. We begin by assuming that the agent is limited to low-dimensional actions with norm less than or equal to M. Then we have:

$$\begin{aligned}
\|\hat{\mathbf{u}}_1 - \hat{\mathbf{u}}_2\| &= \|\hat{\mathbf{Q}}_1\mathbf{a} - \hat{\mathbf{Q}}_2\mathbf{a}\| && (4.14)\\
&= \|(\hat{\mathbf{Q}}_1 - \hat{\mathbf{Q}}_2)\mathbf{a}\| && (4.15)\\
&\leq \|(\hat{\mathbf{Q}}_1 - \hat{\mathbf{Q}}_2)\| \cdot \|\mathbf{a}\| && (4.16)\\
&\leq M\|(\hat{\mathbf{Q}}_1 - \hat{\mathbf{Q}}_2)\| && (4.17)
\end{aligned}$$

where the norm in Eq. 4.17 is the matrix norm induced by the vector 2-norm. We now seek to limit this norm by finding a scalar constant L such that

$$\|\mathcal{Q}(\mathbf{c}_1) - \mathcal{Q}(\mathbf{c}_2)\| \leq L\|\mathbf{c}_1 - \mathbf{c}_2\| \tag{4.18}$$

Given that such an L exists, it is called a Lipschitz constant, and $\mathcal{Q}$ is considered to be L-Lipschitz. It turns out that there is a well understood procedure for training Lipschitz continuous neural networks [22]. Using this Lipschitz regularization, we can choose a constant L_h such that

$$\|\bar{\boldsymbol{\xi}}_1 - \bar{\boldsymbol{\xi}}_2\| \leq L_h \|\mathbf{c}_1 - \mathbf{c}_2\| \tag{4.19}$$

where $\bar{\boldsymbol{\xi}}_1 = h(\mathbf{c_1})$ and $\bar{\boldsymbol{\xi}}_2 = h(\mathbf{c_2})$. The exponential map on the sphere has a Lipschitz constant of 1, so we have the same result for the Lipschitz continuity of $\bar{\mathbf{v}}$ with respect to changes in context. All that remains is to show that $\mathbf{Q}(\bar{\mathbf{v}})$ is Lipschitz continuous.

Theorem 1. *Let $\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2 \in \mathbb{R}^{nk}$ be constructed from k stacked unit n-vectors, and $\mathbf{Q}(\bar{\mathbf{v}})$ be the product of the corresponding Householder reflections (as defined in Eq. 4.7, 4.8). Then,*

$$\|\mathbf{Q}(\bar{\mathbf{v}}_1) - \mathbf{Q}(\bar{\mathbf{v}}_2)\| \leq L_Q \|\bar{\mathbf{v}}_1 - \bar{\mathbf{v}}_2\| \tag{4.20}$$

where $L_Q = 2\sqrt{k}$.

Proof. Please see section A.2 in the appendix. □

Using the fact that the Lipschitz constant of a composition of Lipschitz continuous functions is upper bounded by the product of the constituent Lipschitz constants [22], we combine the results of equation 4.19 and theorem 1 to obtain a Lipschitz constant for $\mathcal{Q}$: NHT is Lipschitz continuous with $L = 2L_h\sqrt{k}$.

Thus, the low-dimensional action $\mathbf{a}$ is guaranteed to result in similar actuations in nearby contexts:

$$\|\hat{\mathbf{u}}_1 - \hat{\mathbf{u}}_2\| \leq 2L_h M\sqrt{k} \cdot \|\mathbf{c}_1 - \mathbf{c}_2\| \tag{4.21}$$

where $\hat{\mathbf{u}}_1 = \hat{\mathbf{Q}}_1\mathbf{a}$ and $\hat{\mathbf{u}}_2 = \hat{\mathbf{Q}}_2\mathbf{a}$, respectively.

Chapter 5

Experiments I — Teleoperation with SCL

In this chapter, we present results from a set of teleoperation experiments that compare SCL – an early approach to actuation subspace prediction from which NHT evolved – to other action dimensionality reduction techniques including PCA, mode-switching, and latent action models. In these experiments, human participants controlled a 7-DOF robotic manipulator with a 2-DOF joystick interface while using SCL to map the joystick inputs to joint velocity commands. A diagram showing the usage of NHT/SCL for teleoperation is shown in figure 5.1.

We begin this section with a discussion of how SCL satisfies several teleoperation properties that have been identified in the literature in section 5.1. We then present results from a latent action user study that compares SCL to PCA and a basic conditional autoencoder (CAE) approach in section 5.2.1. Finally, in section 5.2.2 we compare SCL to mode-switching and a CAE augmented with human-prior losses in a more challenging pouring task. Throughout this section, the context $\mathbf{c}$ available to SCL is the vector of joint angles $\mathbf{q}$, and the actuations $\mathbf{u}$ are joint velocities $\dot{\mathbf{q}}$. We make the corresponding substitutions in the notation.

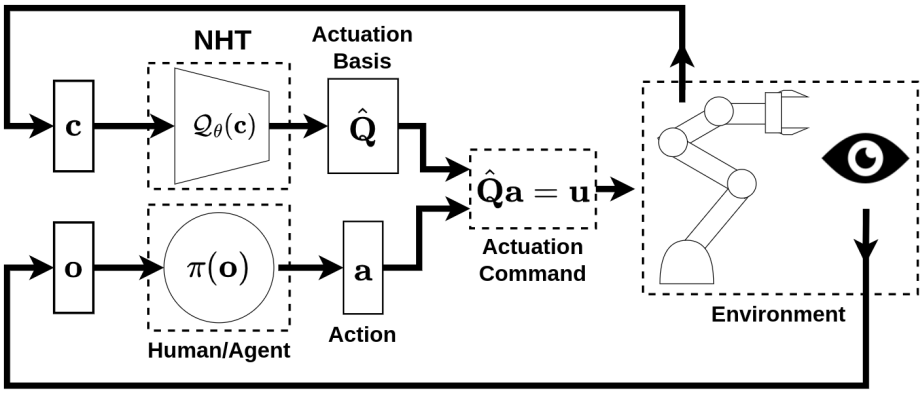


Figure 5.1: System diagram illustrating how an agent (human or artificial) can control a high-DoF robot using NHT.

5.1 Study of Teleoperation Properties

In this section, we show theoretically and empirically how SCL maps guarantee both *proportionality* and *reversibility* which are desirable properties in robotic manipulation. We use $\Psi(\mathbf{q}) = \mathbf{x}$ as the kinematics function of the end-effector's position and orientation [19]. The transition operator is defined as $T(\mathbf{q}, \mathbf{a}; \theta) = \mathbf{q} + \mathcal{Q}(\mathbf{q})\mathbf{a}$.[1]

This section was motivated by the work of Li et. al. [40], whose work we based our properties on, but is otherwise different for several reasons. Their method is semi-supervised targeted. Therefore, it requires a human operator to label subsets of transition data by providing the desired joystick input. Our method is unsupervised thus eliminating the labeling process. Their system also needs additional loss terms to enforce these properties, which require additional hyperparameter tuning.

Proportionality For scalar $\alpha \in \mathbb{R}$ the resulting action $\mathbf{a}' = \alpha\mathbf{a}$ will lead to a proportional change in the end effectors current position:

$$\alpha\|\Psi(T(\mathbf{q}, \mathbf{a}; \theta)) - \Psi(\mathbf{q})\| \approx \|\Psi(T(\mathbf{q}, \alpha\mathbf{a}; \theta)) - \Psi(\mathbf{q})\|.$$

Intuitively, the end-effector is expected to move in proportion to the magnitude of the input. As SCL maps predict a linear weight matrix, proportionality

[1] In practice, the inputs of $\mathcal{Q}$ are not limited to joint angles and can still contain additional task-specific context information.

is locally achieved (where kinematics Ψ is approximately linear) by design. Suppose we have that $\mathbf{a}' = \alpha\mathbf{a}$ for $\alpha \in \mathbb{R}^+$. This will lead to a proportional increase of the response ($\hat{\mathbf{Q}}\mathbf{a}' = \alpha\hat{\mathbf{Q}}\mathbf{a} = \alpha\dot{\mathbf{q}}$).

Soft Reversibility For two states $\mathbf{q}_i$ and $\mathbf{q}_j$ and an action $\mathbf{a} \neq 0$ such that $\mathbf{q}_j = T(\mathbf{q}_i, \mathbf{a}; \theta)$, "Reversibility" means that:

$$\mathbf{q}_j = T(\mathbf{q}_i, \mathbf{a}; \theta) \Rightarrow \mathbf{q}_i = T(\mathbf{q}_j, -\mathbf{a}; \theta).$$

For this definition to hold for SCL maps, $\mathcal{Q}(\mathbf{q}_j)$ must be equivalent to $\mathcal{Q}(\mathbf{q}_i)$, which unfortunately is not guaranteed. Instead, we show that SCL satisfies a "soft reversibility" property: the state $\mathbf{q}_k$ reached by the inverse action from $\mathbf{q}_j$ will be closer than $\mathbf{q}_j$ to $\mathbf{q}_i$.

Theorem 2 (Soft Reversibility)**.** *Let* $\|\boldsymbol{a}\|_2 < 1$ *and* $T(\boldsymbol{q}, \boldsymbol{a}; \theta) = \boldsymbol{q} + \mathcal{Q}(\mathbf{q})\mathbf{a}$. $\mathcal{Q}(\mathbf{q}) \in \mathbb{R}^{n\times k}$ *is a matrix reshaped from the vector form of hidden layer activations* $\mathbf{v}(\mathbf{q}) \in \mathbb{R}^{nk}$*. Suppose* $\mathbf{v}$ *is Lipschitz continuous for some* $L_v \leq 1$*, that is,* $\|\mathbf{v}(\boldsymbol{q}_j) - \mathbf{v}(\boldsymbol{q}_i)\|_2 \leq L_v\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2$*. Then for some* $\boldsymbol{q}_j = T(\boldsymbol{q}_i, \boldsymbol{a}; \theta)$ *we have:*

$$\|T(\boldsymbol{q}_j, -\boldsymbol{a}; \theta) - \boldsymbol{q}_i\|_2 < \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2$$

Proof. From our definition of $T(\mathbf{q}, \mathbf{a}; \theta)$:

$$\begin{aligned} \mathbf{q}_j &= \mathbf{q}_i + \hat{\mathbf{Q}}(\mathbf{q}_i)\mathbf{a}_i \\ \mathbf{q}_k &= \mathbf{q}_j + \hat{\mathbf{Q}}(\mathbf{q}_j)\mathbf{a}_j \end{aligned} \tag{5.1}$$

where $\mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k$ are three consecutive states, $\mathbf{a}_i$ and $\mathbf{a}_j$ are the actions at current time step and the next time step respectively. If the action $\mathbf{a}_j$ is the reverse action $\mathbf{a}_j = -\mathbf{a}_i$, we can rewrite Eq. (5.1) as

$$\begin{aligned} \mathbf{q}_k &= \mathbf{q}_i + \hat{\mathbf{Q}}(\mathbf{q}_i)\mathbf{a}_i + \hat{\mathbf{Q}}(\mathbf{q}_j)(-\mathbf{a}_i) \\ &= \mathbf{q}_i + (\hat{\mathbf{Q}}(\mathbf{q}_i) - \hat{\mathbf{Q}}(\mathbf{q}_j))\mathbf{a}_i \end{aligned}$$

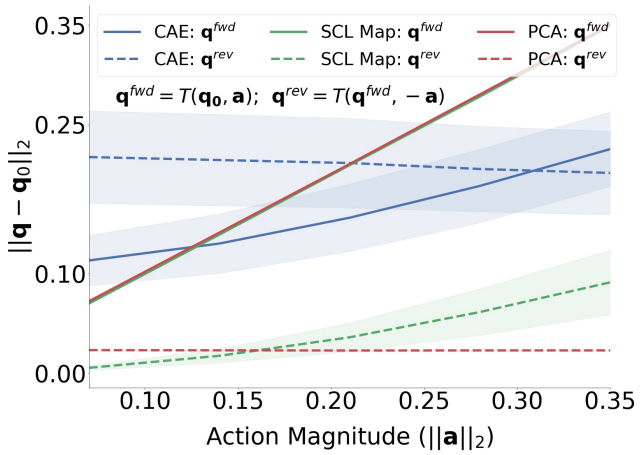


Figure 5.2: Empirical results from an experiment demonstrating proportionality and soft reversibility of SCL maps on the Kinova Gen-3 lite robot. The line and shaded area show the mean and one-half standard deviation from 100 runs (10 trained models, 10 random states), respectively. Note $||\mathbf{q}^{fwd} - \mathbf{q}_0||_2$ for SCL and PCA overlap.

From the Lipschitz continuity assumption, it follows that:

$$\begin{aligned}
\|\mathbf{q}_k - \mathbf{q}_i\|_2 &= \|(\hat{\mathbf{Q}}(\mathbf{q}_i) - \hat{\mathbf{Q}}(\mathbf{q}_j))\mathbf{a}_i\|_2 \\
&\leq \|\hat{\mathbf{Q}}(\mathbf{q}_i) - \hat{\mathbf{Q}}(\mathbf{q}_j)\|_2 \, \|\mathbf{a}_i\|_2 && (5.2)\\
&\leq \|\hat{\mathbf{Q}}(\mathbf{q}_i) - \hat{\mathbf{Q}}(\mathbf{q}_j)\|_F \, \|\mathbf{a}_i\|_2 && (5.3)\\
&= \|\mathbf{v}(\mathbf{q}_i) - \mathbf{v}(\mathbf{q}_j)\|_2 \, \|\mathbf{a}_i\|_2 && (5.4)\\
&\leq L_v \|\mathbf{q}_i - \mathbf{q}_j\|_2 \, \|\mathbf{a}_i\|_2 && (5.5)\\
&\leq (L_v \|\mathbf{a}_i\|_2) \, \|\mathbf{q}_i - \mathbf{q}_j\|_2 && (5.6)\\
&< \|\mathbf{q}_i - \mathbf{q}_j\|_2 && (5.7)
\end{aligned}$$

for any pairs of joint angles in a robot's set of joint configurations $\mathbf{q}_i, \mathbf{q}_j \in \mathcal{Q}$ such that $\mathbf{q}_j = T(\mathbf{q}_i, \mathbf{a}_i; \theta)$, where $\mathcal{Q}$ is the set of joint angles of the robot. We can tell from Eq. (5.7) that if $L_v \|\mathbf{a}_i\|_2 < 1$, taking the reverse action $\mathbf{a}_j = -\mathbf{a}_i$ brings the robot to a state closer to $\mathbf{q}_i$ than before.

□

Experiments Validating Properties We empirically validated our theoretical results for proportionality and soft-reversibility on a *Kinova Gen-3 lite*

[34], comparing the properties of SCL maps to those of CAE and PCA models. For the SCL and CAE algorithms, we trained 10 models each using the Adam optimizer for 200 epochs, with a learning rate of $1e^{-3}$ and mini-batches of 256. Both SCL and CAE were trained with encoders that had two hidden layers each, with 256 neurons and tanh activation functions. The decoders had similar configurations, except that the SCL decoder outputs a matrix as described in section 4.2. During training of the SCL maps, we enforced Lipschitz continuity with $L_v = 1$ by applying the Lipschitz training procedure in Algorithm 1 of Gouk et. al. [22]. We forced the linear maps $\hat{\mathbf{Q}}(\mathbf{q})$ to be orthonormal with the Gram-Schmidt process during training and deployment.

To compare the proportionality and reversibility properties of each algorithm, we generate an angle $\theta \sim \mathcal{U}(0, 2\pi)$ which we transform into actions $\mathbf{a} = \alpha[\cos(\theta), \sin(\theta)]^\top$. We control the norm of the actions with the scalar α. For each value of α, we first apply the action $\mathbf{a}$ for one second, stop the motion, and then apply the inverse action $-\mathbf{a}$. We denote $\mathbf{q}^{fwd}$ and $\mathbf{q}^{rev}$ as the joint configurations of the robot after the forward and inverse actions, respectively. For each run of the experiment we measure first the distance from the start state $\mathbf{q}^0$ to $\mathbf{q}^{fwd}$, and then the distance of $\mathbf{q}^{rev}$ to $\mathbf{q}^0$.

Our results are shown in Fig. 5.2. Ten random states from our demonstration data were chosen as initial states, and each algorithm (CAE, SCL, PCA) was used to compute joint velocities given a random action with increasing magnitude. For SCL and PCA, the reverse action consistently brings the joint positions closer to their original state $\mathbf{q}_0$: this demonstrates the soft reversibility property. For the CAE, the reverse action tended to move the joint positions farther away from the original configuration. Likewise, both PCA and SCL demonstrated an exact linear proportionality in the forward actions, while the CAE did not.

5.2 Teleoperation User Studies

We conducted two sets of user studies to validate the efficacy of SCL as a learnable action mapping approach. We chose to evaluate SCL in teleoperation

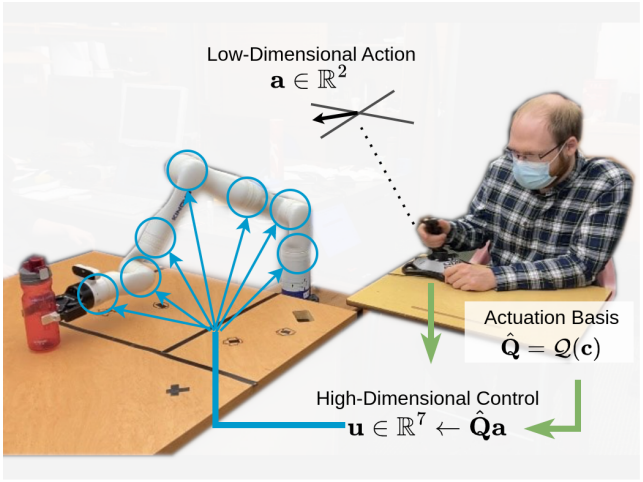


Figure 5.3: A user controls a 7-DOF robotic manipulator through low-dimensional actions of a 2-DOF joystick. The state-conditioned linear mapping $\mathcal{Q}(\mathbf{q}) \in \mathbf{R}^{7\times 2}$ transforms the joystick inputs $\mathbf{a}$ to high-dimensional motor commands $\dot{\mathbf{q}}$.

because of previous research in assistive robotics [42], [43]. The first user study compares across the spectrum of action mapping approaches on two synthetic tasks that empirically could be solved with actions existing in a two-dimensional subspace. The second user study pushes the limits of SCL and compares against a mode-switching control scheme, as well as a more advanced CAE model. Moreover, the second user study task could not be solved with the first two principal components of PCA and was excluded. All experiments were conducted on a Kinova Gen3 lite robot with a control rate of 40 Hz [34]. Reported statistical significance used a 0.05 p-value. A Kruskal-Wallis test was performed before the reported Dunn test significance.

5.2.1 Latent Action User Study

Our first user study compared SCL against other action mapping methods in the literature. 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 compared to existing action mapping approaches.

Experimental Set-up The Table-Cleaning task setup is shown in Figure 5.2.2. Participants were asked to push five small objects on a table out of a designated boundary in 60 seconds. For the table cleaning task, we collected five demonstrations of circular scrubbing motions on a table. Three trajectories were used to train the PCA, SCL, and CAE models, and the remaining two were held out for validation. The total amount of training data was 509 ($\mathbf{q}$,$\dot{\mathbf{q}}$) tuples.

The setup for the Pick and Place trials can be seen in Figure 5.3. The task was 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 the pick and place task. We used six trajectories for training and two for validation of each model. The six training trajectories accounted for 2001 ($\mathbf{q}$,$\dot{\mathbf{q}}$) 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.[2]

Models We compared SCL against PCA [7], [48] and a conditional autoencoder (CAE) [42], [43]. Respectively, they represented a linear and non-linear approach for learning low-to-high dimensional mappings for control. Our CAE model is based on the open-source version of Karamcheti et. al. [31]. 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 were trained with the same data, learning parameters, and architectures described in section 5.1. 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

[2]The studies were approved by the University of Alberta Research Ethics and Managements (Pro00054665).

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 5.4 where we report success rates. Our results suggest 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/context of the manipulator. This meant that the joint velocity commands available to the participants (via the output of action map $g(\mathbf{c}, \mathbf{a})$) 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 context 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 (See Fig. 5.4).

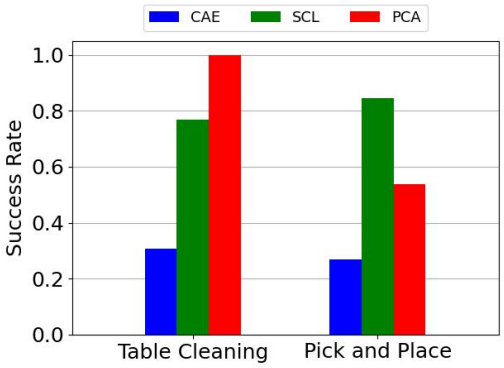


Figure 5.4: The number of trials in which participants successfully completed the Table Cleaning and Pick and Place tasks.

Based on our observations, the low success rate of the CAE model could be attributed to two primary reasons. First, the CAE internally uses affine

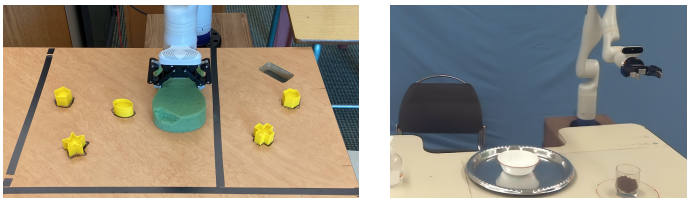


Figure 5.5: Table Cleaning and Pouring Experiments.

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. 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 unpredicted behavior have been previously seen in the literature [43]. In contrast, the soft reversibility property of SCL enabled participants to recover from configurations outside of the training distribution.

5.2.2 Assistive Robotic User Study

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) [47]. We set up a *Pouring* task (Figure 5.2.2), 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, 12 male). Participants were given four two-minute practice trials with each of the control systems to familiarize themselves with the action mappings. We then collected

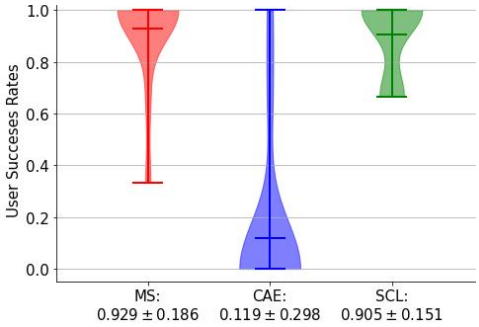


Figure 5.6: Completion rates.

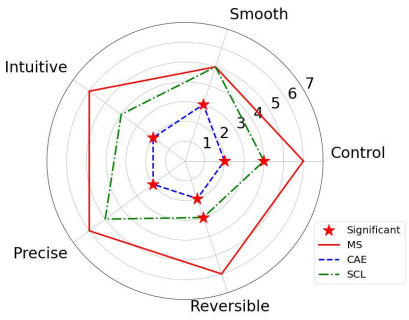


Figure 5.7: Median Likert Scale results.

Pouring User Study Results. Figure 5.6 Violin plots of completion with mean. Figure 5.7 Radar plot with stars indicating significance w.r.t mode-switching.

results for each method over three test trials.

We report the average success rates of participants in Figure 5.6. We found a significant difference between mean success rates when comparing CAE to both SCL and MS, but not comparing SCL to MS. 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 5.7. We found that only Control and Reversibility were statistically significant by Dunn's Test. We also collected the NASA Workload Index [23] featured in Figure 5.8. 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 [47]). 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,

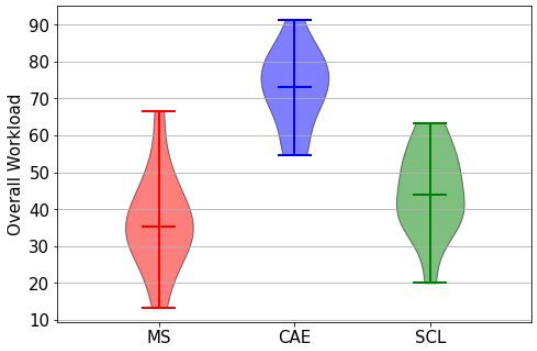


Figure 5.8: Violin plots of NASA TLX scores for each method in the pouring task.

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. As discussed in several instances, we have found the CAE models move without user input. It's 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.

Chapter 6

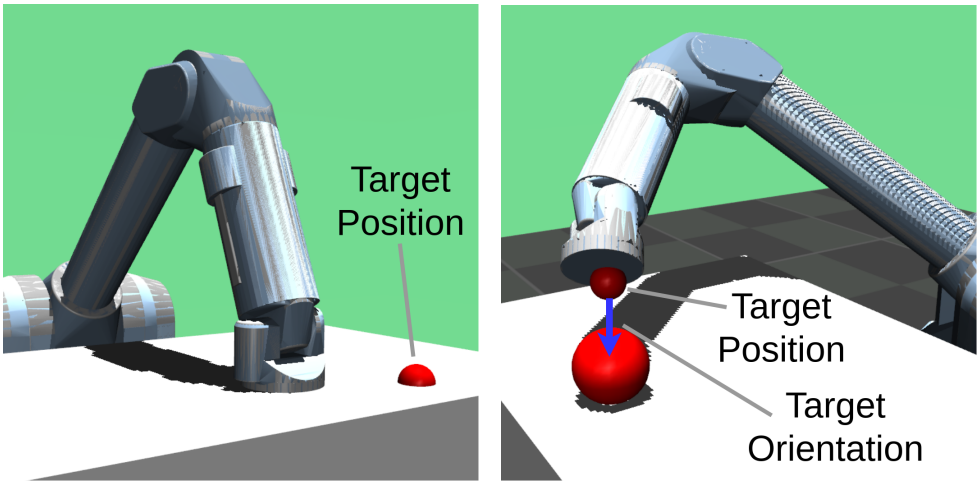
Experiments II — Reinforcement Learning with NHT

Our empirical reinforcement learning results focus on validating the efficacy of neural householder transforms for learning kinematic tasks within a custom MuJoCo simulation of a Barrett WAM robotic manipulator with 7-DOF (see Figure 6.1). We model each task as an MDP and report results in two environments: WAMWipe and WAMGrasp. Learning involves first training an NHT model on an offline dataset of demonstrations, and then fixing the parameters of NHT and using Deep Deterministic Policy Gradient (DDPG) [41] to learn a policy online. We compare DDPG agents trained with an NHT action interface against agents trained with a state-of-the-art latent action model [4], agents trained with an actuation basis computed by SVD, and agents trained in the raw actuation space of the task (i.e., 7-DOF joint velocity control). In our experiments, we used a publicly available implementation of deep deterministic policy gradient [5][1].

6.1 RL Environment Details

In both WAMWipe and WAMGrasp, one environment step corresponds to 10 MuJoCo simulation time-steps of length 0.002 seconds. Both environments

[1] Although we used the implementation of DDPG introduced in the HER paper, we did not use HER in any of our experiments.



(a) WAMWipe (b) WAMGrasp

Figure 6.1: Simulated kinematic manipulation environments with distinct goal types and constraints.

Env	**a** Dim.	**u** Dim.	Goal Type	Constraints
WAMWipe	2	7	Pos	Safety, Contact, Orient.
WAMGrasp	3	7	Pos, Orientation	Safety

Table 6.1: Properties of reinforcement learning environments in simulation experiments.

allow a maximum of 200 environment steps per episode.

6.1.1 WAMWipe

In WAMWipe the goal is to control the manipulator such that the flat face of the last link remains flush against a table while sliding to a randomly sampled goal position. The reward is -1 for every step unless the end-effector is within a small distance of the goal position, in which case the reward is 0. Episode failure occurs if the end-effector: (1) pushes into the table, (2) lifts off of the table, or (3) the end-effector tilts such that it is no longer flush with the table. Let $\mathbf{p}$ denote the unit vector orthogonal to the face of the end-effector, pictured as a purple arrow in figure 6.2. Constraint (3), the orientation constraint, was considered violated when the angle between $\mathbf{p}$ and the vector orthogonal to the surface of the table (not pictured) was greater than $\pi/16$ radians. The agent's observation in our experiments was a concatenated vector of joint

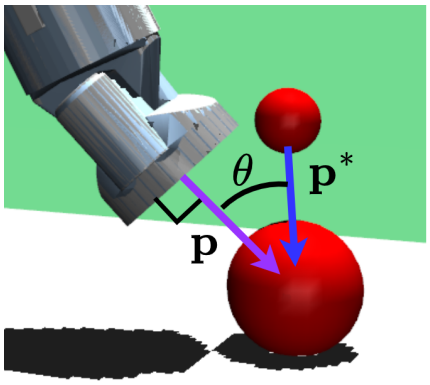


Figure 6.2: The 3-vector **p** pictured here was used to determine whether the orientation constraint/goal-condition was satisfied in WAMWipe/WAMGrasp, respectively.

angles, Cartesian coordinates of the end-effector, Cartesian coordinates of the goal position, and the unit vector orthogonal to the face of the end-effector. The actions in our WAMWipe experiments were either 7-DOF joint velocity commands, or 2-dimensional actions input to an NHT, SVD, or LASER action interface.

6.1.2 WAMGrasp

In WAMGrasp the goal is to simultaneously reach a randomly sampled grasp-point while achieving a goal orientation that is determined by the grasp-point. Let $\mathbf{p}^*$ denote the unit vector pointing from the grasp-point (small sphere in Figure 6.2) to the object being grasped (large sphere in Figure 6.2). We consider the orientation satisfactory if the angle θ between $\mathbf{p}^*$ and the vector orthogonal to the face of the manipulator, $\mathbf{p}$, is less than $\pi/16$ radians. The reward in WAMGrasp is -1 at every step unless the end-effector is within a small distance of the grasp-point with a satisfactory orientation. Episode failure occurs if the end-effector collides with either the object being grasped (large red sphere in Figure 6.2) or the table. In each episode, the grasp-point is randomly sampled from the surface of a sphere with the same center but a larger radius than the large red sphere in Figure 6.2. The agent observation was a concatenated vector of joint angles, Cartesian coordinates of the end-effector, and Cartesian coordinates of the grasp point. The actions in our

WAMGrasp experiments were either 7-DOF joint velocity commands, or 3-dimensional actions input to an NHT, SVD, or LASER action interface.

6.2 Action Interface Baselines

In addition to training NHT from a dataset of demonstrations, we trained LASER [4] from the same dataset and computed the singular value decomposition (SVD) of the dataset of joint velocities executed during the demonstrations. In our experiments, the state-conditioned actuation basis computed by NHT, static basis computed by SVD, and nonlinear decoder of LASER all serve as different choices of an interface between DDPG and the raw actuation commands that determine the next state of the environment. In our WAMWipe experiments, NHT, LASER, and SVD all exposed a two-dimensional action interface to DDPG, while in WAMGrasp they all exposed three-dimensional interfaces. The $k \in \{2, 3\}$ actuation bases provided by SVD were the vectors in $\mathbb{R}^7$ corresponding to the k largest singular values.

Demonstrations were collected by recording context-actuation $(\mathbf{c}, \dot{\mathbf{q}})$ pairs from PD controllers that were hand-engineered for each environment. In both environments, the context $\mathbf{c}$ upon which the output of NHT and LASER are conditioned was the concatenation of joint angles and Cartesian coordinates of the end-effector. LASER is regularized by the KL and dynamics terms in its loss function (see equation 3.1), while we regularize NHT by enforcing Lipschitz continuity with Lipschitz constant L at each layer during training [22]. Moreover, the Adam optimizer is used for both NHT and LASER, with learning rate α_{map}, and otherwise default parameters. Likewise, Adam is used as the optimizer in our chosen implementation of DDPG, with learning rates α_{actor} and α_{critic} for the policy and value function, respectively.

6.3 Hyperparameter Search

The hyperparameter search experiment described here was designed to estimate the performance of the best policy that could be learned by DDPG in a finite amount of time for agents trained with (1) an NHT action interface, (2)

a LASER [4] action interface, (3) an actuation basis computed by SVD, and (4) 7-DOF joint velocity actions.

The hyperparameter search also enabled us to study the sensitivity of agent learning-dynamics to different hyperparameter configurations for each action interface. The results serve as empirical evidence with which to answer questions such as: "Could changing the neural architecture of NHT cause a significant drop in the final success rate of a policy learned by DDPG?" Answering such questions is non-trivial since there may or may not be complex interactions between map (i.e., NHT, LASER) hyperparameters, DDPG hyperparameters, and final agent performance. It is unknown whether NHT hyperparameters tuned for an agent with arbitrary configuration A will be the best NHT hyperparameters for an agent with a different configuration B. For example, it is conceivable that a DDPG agent with hyperparameter configuration A may perform best with NHT configuration C, while DDPG with configuration B performs best with NHT configuration D. Thus a meaningful search should jointly vary the hyperparameters of the mapping models and the DDPG agent.

Although a grid search over hyperparameters is a common approach to answer such questions, it has been shown that random search in the space of hyperparameters may be more efficient [10] (i.e., find better hyperparameter configurations at a lower computational cost). As such, we performed a random search over map (i.e., NHT, LASER) and DDPG hyperparameters. We jointly sampled 128 configurations each for NHT + DDPG, LASER + DDPG, DDPG with SVD, and DDPG with joint velocity actions. For the latter two conditions, the only hyperparameters of interest are those of DDPG itself. For each configuration, we first trained the mapping function (if applicable), and then trained the DDPG agent, over five runs with different random seeds. The ranges and method of sampling used for each hyperparameter are listed in Table 6.2. Moreover, for both WAMWipe and WAMGrasp, each agent was trained for one million environment steps, using three workers to generate experience. This resulted in 100 training epochs of 10,000 steps each.

Figure 6.3 summarizes the results of the random hyperparameter search

Parameter	Range	Sampling
α_{map}	(1e-6, 1e-1)	Exponential
β_{KL} (LASER)	(1e-6, 1e0)	Exponential
β_{dyn} (LASER)	(1e-6, 1e0)	Exponential
L (NHT)	(1e-1, 1e2)	Exponential
Map Batch Size	(32, 256)	Uniform
Map Activation	{ReLU, sigmoid, tanh}	Uniform
Map Hidden Layers	{1, 2, 3, 4}	Uniform
Map Hidden Units	(16, 1024)	Geometric
α_{actor}	(1e-4, 1e-2)	Exponential
α_{critic}	(1e-4, 1e-2)	Exponential
Rand Action ϵ	(0, 0.4)	Uniform
Action Noise σ	(0, 0.4)	Uniform
Penalty on $\|\mathbf{a}\|$	(0, 1)	Uniform
Max $\|\mathbf{a}\|$	(1, 10)*	Exponential
DDPG Batch Size	(32, 256)	Uniform
Polyak	(0.9, 0.99)	Uniform

Table 6.2: Hyperparameter ranges and sampling methods for pre-trained mapping functions (top) and DDPG (bottom). We use 1ex as shorthand to denote 1×10^x. Exponential and Geometric sampling of parameters was carried out as described in [10]. *For the agent with 7-DOF actions we use the range (0.1, 10).

for NHT and DDPG, LASER and DDPG, DDPG with SVD, and DDPG with joint velocity actions. The violin plots represent the distribution of final success rates (success rate after 100 epochs of training) across every randomly sampled hyperparameter configuration. We found that the distributions of final success rates across hyperparameter configurations suggest that agents trained with NHT tend to be more robust to different hyperparameter configurations. Although in some runs the 7-DOF agent managed to reach a success rate of 100% in WAMGrasp, the variance of final success rates amongst 7-DOF agents is much larger than the variance of success rates for NHT agents. In general, there was not a strong correlation between any one hyperparameter and the final performance of the agents (coefficient of determination < 0.1). One exception was that we found small values for max$\|\mathbf{a}\|$ significantly harmed the agents with low-dimensional actions, but helped the agents with 7-DOF

Parameter	Range	Sampling
α	(1e-4, 1e-2)	Exponential
Discount Factor γ	(0.9, 0.99)	Uniform
GAE Parameter λ	(0.9, 0.99)	Uniform
VF coefficient	(0.25,1)	Uniform
Clipping Parameter ϵ	(0.05,0.5)	Uniform
Steps-per-update	{1024,2048,4096}	Uniform

Table 6.3: Hyperparameter ranges for PPO in the HalfCheetah-v4 experiments. We use 1ex as shorthand to denote 1×10^x. Exponential sampling of parameters was carried out as described in [10].

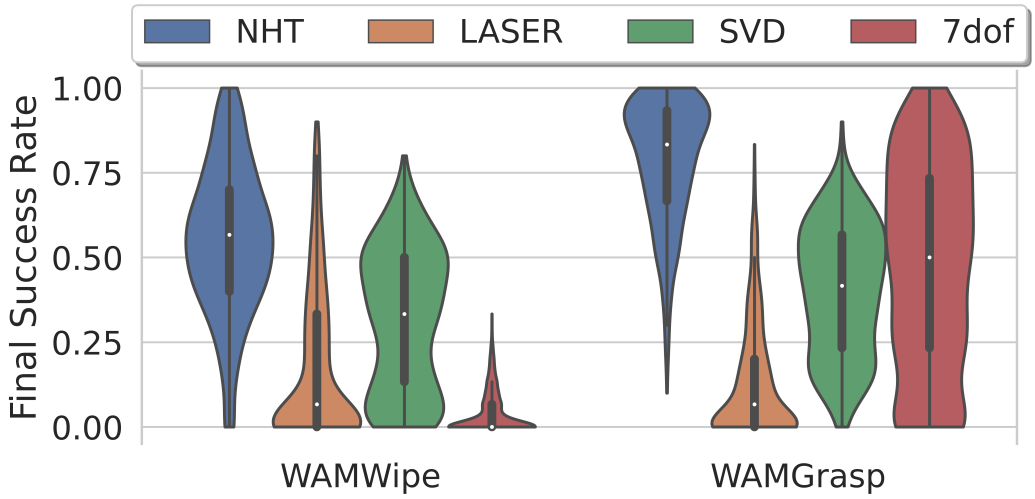


Figure 6.3: Violin plots of final success rates across 128 randomly sampled hyperparameter configurations (five runs each).

actions, particularly in WAMWipe.

We now seek to compare NHT + DDPG with the baselines LASER + DDPG, DDPG with SVD, and DDPG with joint velocity actions after hyperparameter optimization (i.e., using their best hyperparameters respectively). The learning curves in Figure 6.4 plot the mean success rate during training for the best performing agent in each condition, averaged over five runs. It can be seen that DDPG agents trained with an NHT action interface produced the best performing agents after hyperparameter optimization (higher success rates in fewer epochs) in both WAMWipe and WAMGrasp.

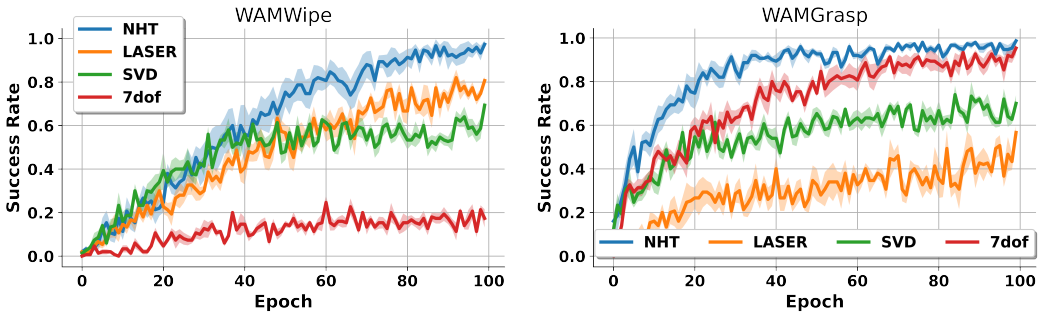


Figure 6.4: Learning curves corresponding to the configurations with the best average final success rate, overall hyperparameter configurations, for each method. Each curve shows the mean success rate over five runs of the best configuration, with the shaded regions indicating the standard error.

6.4 Comparison to Jacobian Pseudoinverse Interface

Here we compare NHT to an additional choice of action interface that, unlike the baselines discussed in the main text, is not learned from demonstrations. The Jacobian of the robotic manipulator describes the relationship between the joint velocities and the Cartesian and angular velocity of the end-effector. The pseudoinverse of the Jacobian can be used to define a six-dimensional action interface for an RL agent: three dimensions in the agent's action space correspond to Cartesian velocity, and the remaining three correspond to the angular velocity.

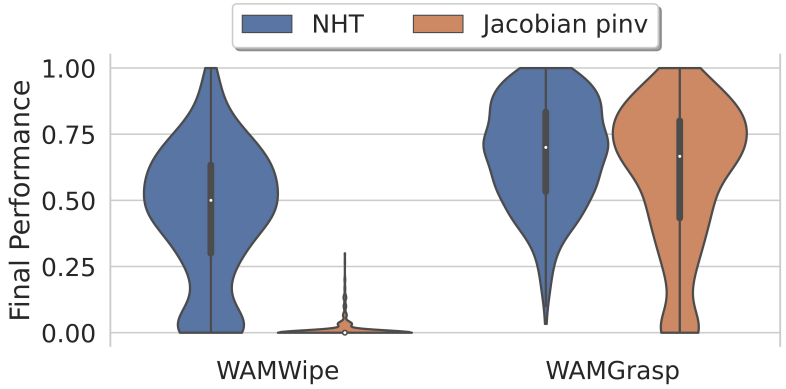


Figure 6.5: Violin plots of final success rates across 128 randomly sampled hyperparameter configurations (five runs each) for NHT vs a Jacobian pseudoinverse (Jacobian pinv) action mapping baseline.

We compared NHT to the Jacobian pseudoinverse as an action interface

for a DDPG agent in WAMGrasp and WAMWipe in a hyperparameter search experiment with the same methodology described in section 6.3(128 hyperparameter configurations, five seeds for each configuration). As already noted, the dimensionality of the agent's action space was six when using the Jacobian pseudoinverse interface. NHT was used to learn a two-dimensional action interface for WAMWipe, and a three-dimensional action interface for WAMGrasp. The hyperparameter search results are plotted in figure 6.5. The variation in performance for the Jacobian pseudoinverse agents is entirely due to different DDPG agent configurations (the Jacobian has no hyperparameters).

As expected, the agent with the Jacobian pseudoinverse action interface performed poorly in WAMWipe; like the 7DOF joint velocity agent, the Jacobian pseudoinverse agent was able to freely jam the end-effector of the robot into the table, or lift the end-effector from the table, resulting in an automatic failure for its training episodes. Without a learned action interface, the exploratory behavior inherent in reinforcement learning resulted in destructive behavior that made learning in the highly constrained environment of WAMWipe difficult.

In WAMGrasp, the Jacobian pseudoinverse agents were sometimes able to learn to achieve 100% success rate. However, it is clear from the violin plots in figure 6.5 that limiting the joint velocity commands to useful subspaces learned by NHT has some benefit over allowing free exploration with the Jacobian pseudoinverse interface. Some of the poorer hyperparameter configurations resulted in close to 0% success rate when interacting with WAMGrasp through the Jacobian pseudoinverse interface. NHT tended to concentrate agent performance, overall hyperparameter configurations, toward a success rate of 75% to 100%.

6.5 HalfCheetah

While our main interest in the application of NHT lies in constrained/safe robotic manipulation, there is value in validating the utility of NHT in more standard reinforcement learning environments. In addition, it is important to

show that the action interface learned by NHT is useful for agents trained with various RL algorithms; not only for DDPG agents. We therefore performed a hyperparameter search experiment with a standard implementation [20] of PPO [55] on the HalfCheetah-v4 environment from OpenAI Gym [11]. This environment has a 17-dimensional observation space that includes angular positions and velocities, and a six-dimensional torque actuation space (compared to the joint velocity actuation space in WAMGrasp and WAMWipe). We compared NHT agents to agents that learned in the standard 6DOF actuation space of HalfCheetah, and agents with LASER [4] and SVD action interfaces. NHT, SVD, and LASER all learned 2-dimensional action interfaces. This experiment precisely mirrored the hyperparameter search experiments reported in section 6.3, except that the demonstrations used to train NHT, SVD, and LASER were collected from the best-performing policy learned by the standard 6DOF agent. A total of just 1,000 transitions were recorded from this expert policy. The return on this demonstration episode was over 6,000. The hyperparameter ranges and sampling methods for this experiment are summarized in Table 6.3.

The learning curves of the agents with the best average final performance, and the distribution of final agent performances for each method in HalfCheetah-v4 are shown in Figure 6.6. Interestingly, we found that the constant (i.e. *not* state-dependent) action interface of SVD was sufficient to learn more efficiently than the standard 6-DOF agent while still achieving the same asymptotic performance. This suggests that all of the instantaneous actuations used by an expert ($> 6,000$ return) HalfCheetah agent lie close to a fixed two-dimensional linear subspace! There appears to be some benefit to the agent learning in an adaptive actuation subspace with NHT, although the performance gains are small in this environment. The agents learning with NHT tended again to be more robust to different hyperparameter configurations.

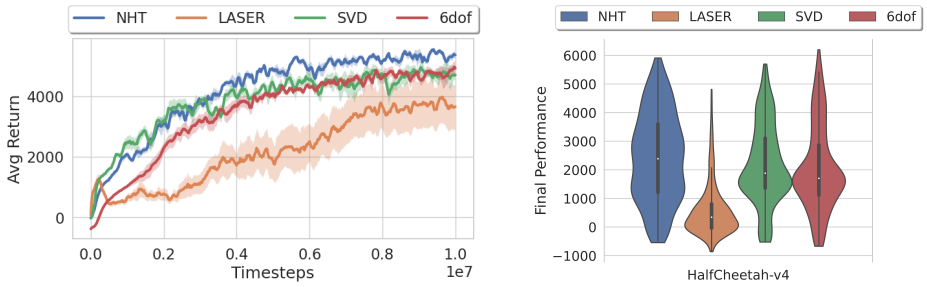


Figure 6.6: Results of hyperparameter search for action mapping methods in HalfCheetah-v4. **Left**: Learning curves corresponding to the configurations with the best average final success rate. **Right**: Violin plots of final success rates across 128 randomly sampled hyperparameter configurations (five runs each).

6.6 Ablations

6.6.1 Ablation of Orthonormal Constraint

What is the benefit of enforcing orthonormal actuation bases in NHT? Besides the fact that the pseudoinverse of $\hat{\mathbf{Q}}$ can be computed trivially as the transpose during training, we wanted to find out if there was any empirical benefit. To answer this question we performed an experiment in which we trained a neural network to produce an arbitrary state-conditioned matrix as an actuation basis for WAMWipe and WAMGrasp. Unlike NHT, this baseline is not constrained to output a matrix with orthonormal columns. We will refer to the baseline as the state-conditioned linear map (SCL) model. The SCL baseline is a neural network $h_\theta : \mathbf{c} \mapsto \hat{\mathbf{Q}} \in \mathbb{R}^{n \times k}$ that maps context vectors to an $n \times k$ matrix $\hat{\mathbf{Q}}$. In our WAMWipe experiment $n = 7$ and $k = 2$, while for WAMGrasp $n = 7$ and $k = 3$.

As in section 6.3, we performed a hyperparameter search in which we sampled 128 hyperparameter configurations and trained NHT/SCL and then a DDPG agent with five different random seeds. The sampling method and ranges for each parameter were the same for both NHT and SCL, and are listed in Table 6.2. Figure 6.7 summarizes the results of this hyperparameter search with violin plots of the final performance attained by agents with NHT and SCL action interfaces for the WAMWipe and WAMGrasp environments.

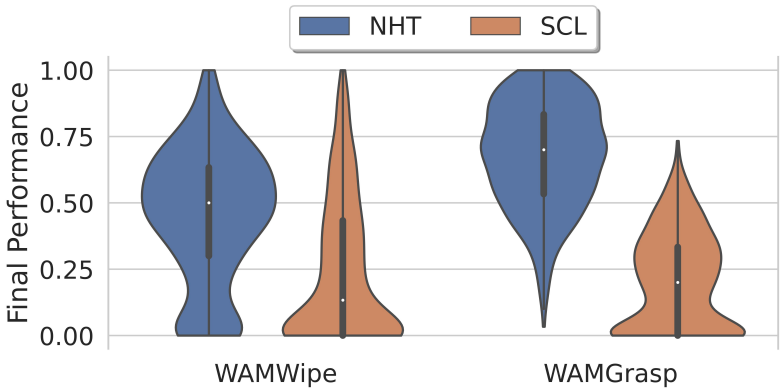


Figure 6.7: Violin plots of final success rates across 128 randomly sampled hyperparameter configurations (five runs each) for NHT vs a non-orthonormal state-conditioned linear (SCL) action mapping baseline.

We found that 64 out of 640 (10%) of the runs for SCL in WAMWipe failed due to numerical instability. In these cases the matrices output by the unconstrained neural network had large norms, resulting in very large joint velocity actuations that caused the MuJoCo simulations to fail. Interestingly, we did not observe the same numerical stability issues in the SCL models that were trained for WAMGrasp. Note that, in contrast, for NHT numerical stability is not an empirical issue. The 2-norm of the matrix produced by NHT is guaranteed to be equal to one.

In WAMWipe, the best hyperparameter configurations of SCL resulted in actuation interfaces that were suitable for the DDPG agent to achieve 100% success rate. However, in both WAMWipe and WAMGrasp, the distributions of final agent performance in Figure 6.7 indicate that NHT was more robust than SCL with respect to variation in hyperparameter configurations. This suggests that NHT may be less sensitive to different choices of hyperparameters, making it easier to tune in practice.

6.6.2 Ablation of Actuation Projection

The latent action framework advanced by Losey et al. [43] and described in section 3.4 involves an encoder neural network $f_\phi(\mathbf{c}, \mathbf{u}) = \hat{\mathbf{a}}$ which outputs a low-dimensional latent action, and a decoder network $g_\theta(\hat{\mathbf{a}}, \mathbf{c}) = \hat{\mathbf{u}}$ that reconstructs the actuation given the low-dimensional action and its context.

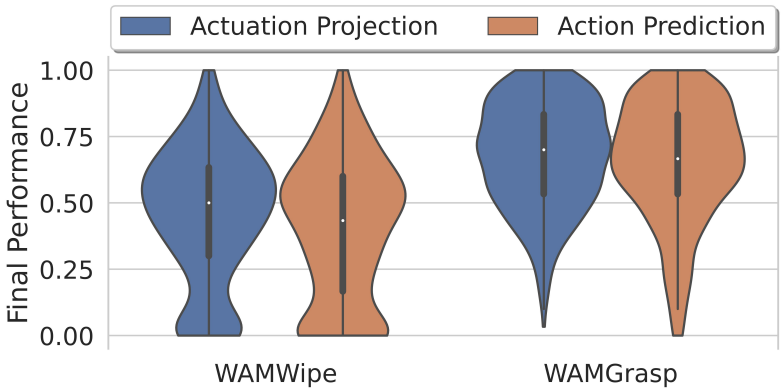


Figure 6.8: Performance distribution of DDPG agents trained on WAMWipe and WAMGrasp, with NHT interfaces optimized using actuation projection and action prediction.

Recall that in section 4.1 we began our discussion of NHT/SCL by noting that $\hat{\mathbf{Q}} = \mathcal{Q}_\theta(\mathbf{c})$ can be interpreted as a linear decoder in the latent action framework. We continued our development of NHT by discussing how actuation projection can be used to obtain the optimal low-dimensional action for reconstruction of an actuation given a fixed actuation basis $\hat{\mathbf{Q}}$. In this subsection we present the results of an ablation experiment that was designed to provide evidence regarding the question "How would using an encoder neural network instead of actuation projection affect the performance of NHT?".

In fact, early versions of SCL did use an encoder neural network to predict low-dimensional actions during training. At deployment (i.e., when NHT is used as a control interface), the encoder is discarded, as only the decoder is necessary to compute the actuation basis. We will refer to the training of NHT with an encoder neural network as action prediction, as opposed to the default training procedure that relies on actuation projection.

The optimization problem of NHT trained with action prediction becomes:

$$\min_{\phi,\theta} \mathbb{E}_\pi \|\mathbf{u} - \hat{\mathbf{Q}}\hat{\mathbf{a}}\|_2^2 \tag{6.1}$$

where $\hat{\mathbf{Q}} = \mathcal{Q}_\theta(\mathbf{c})$ and $f_\phi(\mathbf{c}, \mathbf{u}) = \hat{\mathbf{a}}$.

We conducted an experiment in which we performed a large-scale random hyperparameter search, comparing the final performance of agents trained with NHT action interfaces optimized using either actuation projection or action

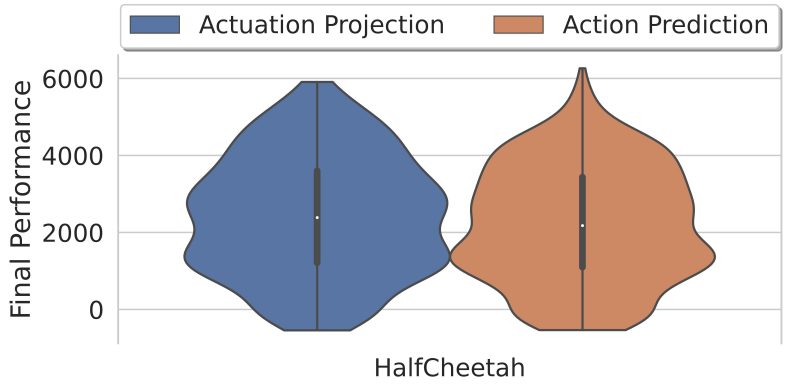


Figure 6.9: Performance distribution of PPO agents trained on HalfCheetah, with NHT interfaces optimized using actuation projection and action prediction.

prediction. For both conditions, we sampled 128 hyperparameter configurations and trained each configuration with five different seeds.

Figure 6.8 shows the results of the ablation of actuation projection in the WAM environments. In this experiment, we found that the median final performance of agents trained using action prediction NHT interfaces was slightly lower than the median performance of agents trained using an actuation projection NHT interface. For both WAMGrasp and WAMWipe, the shape of the performance distributions in the respective environment was quite similar.

We observed roughly the same result for the HalfCheetah environment, depicted in figure 6.9. In this experiment, the actuation projection agents had a slightly higher median performance compared to agents trained with an NHT interface optimized using action prediction. The highest performing agent in the action prediction condition performed slightly above its actuation projection counterpart, but it is not clear if this result is repeatable.

Overall, a conclusion we can draw from these ablation experiments is that removing the encoder neural network and training NHT instead with actuation projection does not harm agent performance. This is a positive result because actuation projection is computationally cheaper compared to training an encoder neural network. With actuation projection, we entirely eliminate the memory footprint of the encoder neural network and cut the number of learnable parameters in half.

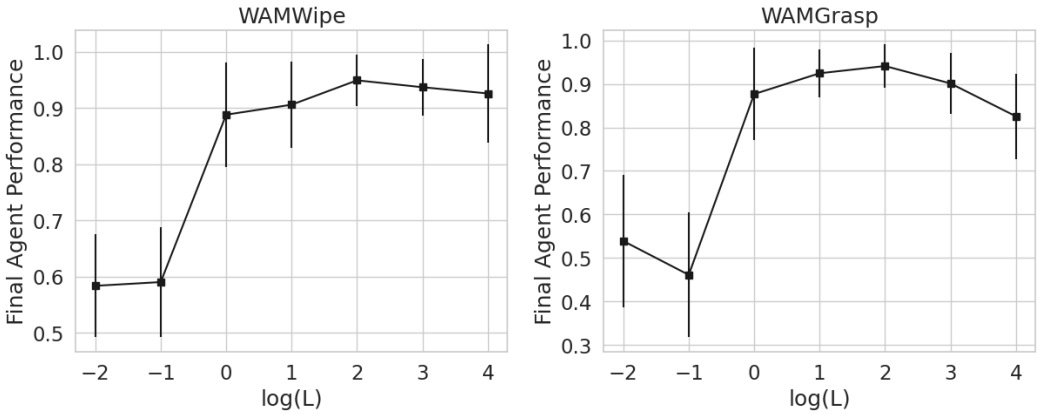


Figure 6.10: Results of Lipschitz regularization sensitivity study in WAMWipe and WAMGrasp. Vertical bars indicate standard deviation of agent performance.

6.6.3 Sensitivity to Lipschitz Regularization

We have gone to lengths to ensure the Lipschitz continuity of NHT. If an orthonormal actuation basis was our only desideratum, we would have simply performed Gram-Schmidt orthonormalization on a neural network output, as described in section 4.2.1. We opted to go further, making use of exponential maps and householder reflections, in order to ensure that $\hat{\mathbf{Q}}$ changes smoothly as a function of context. These computational design choices result in an NHT that includes a Lipschitz regularization hyperparameter, L (see section 4.3.3 for details). Recall that for a fixed low-dimensional action, the Lipschitz continuity of NHT guarantees that $\|\hat{\mathbf{u}}_1 - \hat{\mathbf{u}}_2\| \leq L\|\mathbf{c}_1 - \mathbf{c}_2\|$, where an agent takes the same low-dimensional action $\mathbf{a}$ in both contexts $\mathbf{c}_1$ and $\mathbf{c}_2$. This subsection presents results from experiments that aimed to study the sensitivity of agent performance to different values of this Lipschitz regularization parameter.

In this experiment, we set every hyperparameter aside from the Lipschitz parameter L to the best-performing configuration from our random hyperparameter search experiments (see section 6.3). We chose 6-7 values of the Lipschitz parameter, and for each value trained NHT and then the agent with 30 random seeds. Figure 6.10 shows the results of this sensitivity study in the WAM environments, while Figure 6.11 shows the results in the HalfCheetah environment.

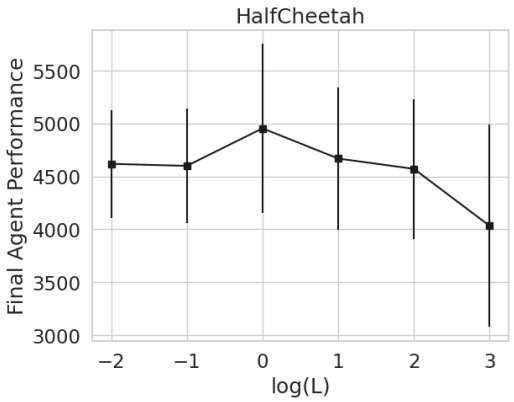


Figure 6.11: Results of Lipschitz regularization sensitivity study in HalfCheetah. Vertical bars indicate standard deviation of agent performance.

The first observation we note from these results is that the optimal value for L will be dependent on the environment; more precisely, it depends on the relative scale of changes in context vs actuation vectors. In WAMWipe and WAMGrasp, the best performing agents had a Lipschitz parameter value of 100 (note the log scale of the x-axis in both Figures). In HalfCheetah, the best performing agent had a Lipschitz parameter value of 1.

Another observation is that the Lipschitz parameter may have an effect on the variance of agent performance. In the WAM environments, the best performing values of L resulted in smaller variance in performance across the 30 runs. In HalfCheetah, we did not observe a similar effect. Future work should seek to better understand this potential correlation between the Lipschitz parameter and variance in agent performance, and analyze results for statistical significance.

We also note that there are complex interactions between the hyperparameters of NHT and the hyperparameters of the agents in these experiments. These interactions are not well understood. Setting aside the complexity introduced by NHT, even the interactions between different hyperparameters of any particular implementation of DDPG are not well understood, for example. The results presented in Figure 6.10 and Figure 6.11 should not be interpreted as evidence that any particular value for L will always be optimal in these environments. These are case studies for a particular fixed hyperparameter

configuration in each environment.

One conclusion we may draw from these experiments is that for certain configurations, there is value in enforcing the Lipschitz continuity of NHT. The results on WAMGrasp, for example, show this. The right-most value of the Lipschitz parameter (10^4) is essentially no Lipschitz regularization; this is the result one would obtain if they simply trained NHT with a standard optimizer (e.g., unmodified Adam optimizer). We can see that the introduction of Lipschitz regularization (i.e., decreasing the value of the L to around 100) has a positive effect on performance. Too much Lipschitz regularization of course damages performance. In the extreme, a Lipschitz parameter with a value of zero would result in a fixed actuation basis that cannot change with changing context (similar to the SVD action interface of section 6.3).

Chapter 7

Conclusions

In this thesis, we proposed actuation subspace prediction as a novel problem formulation for robotic control and an alternative to deep latent action models. We derived the Neural Householder Transform model as a solution to actuation subspace prediction and showed that it is smooth with respect to changes in context. Through two teleoperation user studies, we found that SCL (an earlier form of NHT) is a promising approach to enable humans to control high-DOF manipulators with a simple joystick interface. In a large hyperparameter search experiment, we found that reinforcement learning agents trained with NHT outperformed agents trained to act in (1) the original actuation space, (2) a global linear actuation basis computed by SVD, and (3) a state-of-the-art deep latent action model, LASER, in novel WAMWipe and WAMGrasp robotic manipulation environments, as well as in the standard HalfCheetah environment.

The experiments involving reinforcement learning agents trained with an NHT action interface were designed with the aim of minimizing the danger of false positive results. Extensive hyperparameter searches were conducted not just for the NHT agents but also for all of the baselines, including ablated variants of NHT. Conclusions were generally drawn with respect to distributions of performance for each method.

Limitations and Future Work The approach to action mappings presented in this thesis has limitations that may be inherent to the problem of controlling high degree-of-freedom systems with low-dimensional action inter-

faces. Our results seem to empirically suggest that being locally constrained to a subset of the actuation space has the potential to degrade the performance of RL agents (see e.g., the performance of LASER in WAMGrasp). We hypothesize that this is a result of the agent's options being constrained (in some states) to a subset of the actuation space that mostly contains joint velocity vectors that are not useful/harmful. This can occur if the action mapping model was not fully trained, or learned a non-ideal mapping.

An additional open problem is that it is not clear how NHT will perform in manipulation tasks that involve significant discontinuous forces. In contrast to the discontinuous nature of contact interactions, we have shown that the actuation bases of NHT change smoothly with respect to context. However, it is imaginable that discontinuities could be accounted for by a discontinuous policy in the low-dimensional action space. We can also consider augmenting our method with ideas from the force control and impedance control literature as a promising future research direction.

Several open questions relating to NHT remain. First, how can we predict which pre-trained NHT models will be successful as action interfaces for teleoperation or RL agents? A second open question is whether NHT can be useful as an interface for visual servoing, or other alternative robotic control methods. Third, will NHT with joint torque actuation bases be useful for teleoperation? The NHT interfaces used in the HalfCheetah experiments did include joint torque actuations, but this modality has not yet been explored in the context of teleoperation.

Overall, NHT shows promise in providing a low-dimensional interface with which to control high-dimensional systems for both humans and artificial agents. As NHT can be viewed as a neural network architecture, or as a computational tool for learning subspaces that correspond to continuous context vectors, there may be additional applications outside of actuation subspace prediction for which NHT is utilized in the future.

References

[1] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first International Conference on Machine Learning (ICML)*, 2004, p. 1.

[2] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*. Dec. 2008, vol. 78, ISBN: 978-0-691-13298-3. DOI: 10.1515/9781400830244.

[3] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20, JMLR.org, 2020.

[4] A. Allshire, R. Martin-Martin, C. Lin, S. Manuel, S. Savarese, and A. Garg, "LASER: Learning a latent action space for efficient reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2021. DOI: 10.1109/icra48506.2021.9561232. [Online]. Available: https://doi.org/10.1109%2Ficra48506.2021.9561232.

[5] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

[6] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103 500, 2021, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2021.103500. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370221000515.

[7] P. K. Artemiadis and K. J. Kyriakopoulos, "Emg-based control of a robot arm using low-dimensional embeddings," *IEEE Transactions on Robotics*, vol. 26, no. 2, pp. 393–398, 2010. DOI: 10.1109/TRO.2009.2039378.

[8] C. Atkeson, A. Moore, and S. Schaal, "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, pp. 75–113, Feb. 1997. DOI: 10.1023/A:1006511328852.

[9] M. Bain and C. Sammut, "A framework for behavioural cloning," in *Machine Intelligence 15*, 1995.

[10] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html.

[11] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *Openai gym*, 2016. eprint: arXiv:1606.01540.

[12] S. Calinon and D. Lee, "Learning control," in *Humanoid Robotics: a Reference*, P. Vadakkepat and A. Goswami, Eds., Springer, 2019, pp. 1261–1312. DOI: 10.1007/978-94-007-6046-2_68.

[13] S. Calinon, "Learning from demonstration (programming by demonstration)," 2018.

[14] S. Calinon, Z. Li, T. Alizadeh, N. G. Tsagarakis, and D. G. Caldwell, "Statistical dynamical systems for skills acquisition in humanoids," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, 2012, pp. 323–329. DOI: 10.1109/HUMANOIDS.2012.6651539.

[15] N. Chen, M. Karl, and P. van der Smagt, "Dynamic movement primitives in latent space of time-dependent variational autoencoders," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 629–636. DOI: 10.1109/HUMANOIDS.2016.7803340.

[16] J. Cheng, F. Abi-Farraj, F. Farshidian, and M. Hutter, *Haptic teleoperation of high-dimensional robotic systems using a feedback mpc framework*, 2022. DOI: 10.48550/ARXIV.2207.14635. [Online]. Available: https://arxiv.org/abs/2207.14635.

[17] M. Ciocarlie and P. Allen, "Hand posture subspaces for dexterous robotic grasping," *I. J. Robotic Res.*, vol. 28, pp. 851–867, Jun. 2009. DOI: 10.1177/0278364909105606.

[18] A. Colomé, G. Neumann, J. Peters, and C. Torras, "Dimensionality reduction for probabilistic movement primitives," in *2014 IEEE-RAS International Conference on Humanoid Robots*, ISSN: 2164-0580, Nov. 2014, pp. 794–800. DOI: 10.1109/HUMANOIDS.2014.7041454.

[19] J. J. Craig, *Introduction to Robotics: Mechanics & Control*, en. Addison-Wesley Publishing Company, 1986, ISBN: 978-0-201-10326-7.

[20] P. Dhariwal, C. Hesse, O. Klimov, *et al.*, *Openai baselines*, https://github.com/openai/baselines, 2017.

[21] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. [Online]. Available: https://doi.org/10.1080/14786440109462720.

[22] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree, "Regularisation of neural networks by enforcing lipschitz continuity," *Mach. Learn.*, vol. 110, no. 2, pp. 393–416, Feb. 2021, ISSN: 0885-6125. DOI: 10.1007/s10994-020-05929-w. [Online]. Available: https://doi.org/10.1007/s10994-020-05929-w.

[23] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research," in *Advances in psychology*, vol. 52, Elsevier, 1988, pp. 139–183.

[24] I. Havoutis and S. Calinon, "Learning from demonstration for semi-autonomous teleoperation," *Autonomous Robots*, vol. 43, Mar. 2019. DOI: 10.1007/s10514-018-9745-2.

[25] M. Heath, *Scientific Computing: An Introductory Survey, Revised Second Edition*, ser. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2018, ISBN: 9781611975574. [Online]. Available: https://books.google.ca/books?id=faZ8DwAAQBAJ.

[26] L. V. Herlant, R. M. Holladay, and S. S. Srinivasa, "Assistive teleoperation of robot arms via automatic time-optimal mode switching," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016, pp. 35–42. DOI: 10.1109/HRI.2016.7451731.

[27] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of Educational Psychology*, vol. 24, pp. 498–520, 1933.

[28] F. Jarboui and V. Perchet, "Offline inverse reinforcement learning," *CoRR*, vol. abs/2106.05068, 2021.

[29] I. Jo, Y. Park, and J. Bae, "A teleoperation system with an exoskeleton interface," in *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2013, pp. 1649–1654. DOI: 10.1109/AIM.2013.6584333.

[30] H. Jun Jeon, D. Losey, and D. Sadigh, "Shared Autonomy with Learned Latent Actions," en, in *Robotics: Science and Systems XVI*, Robotics: Science and Systems Foundation, Jul. 2020, ISBN: 978-0-9923747-6-1. DOI: 10.15607/RSS.2020.XVI.011. [Online]. Available: http://www.roboticsproceedings.org/rss16/p011.pdf (visited on 09/15/2021).

[31] S. Karamcheti, A. J. Zhai, D. P. Losey, and D. Sadigh, "Learning visually guided latent actions for assistive teleoperation," in *Proceedings of the 3rd Conference on Learning for Dynamics and Control*, A. Jadbabaie, J. Lygeros, G. J. Pappas, *et al.*, Eds., ser. Proceedings of Machine Learning Research, vol. 144, PMLR, Jul. 2021, pp. 1230–1241. [Online]. Available: http://proceedings.mlr.press/v144/karamcheti21a.html.

[32] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011. DOI: 10.1109/TRO.2011.2159412.

[33] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *ICLR*, 2014.

[34] Kinova Robotics, *Ros kortex*. [Online]. Available: https://github.com/Kinovarobotics/ros_kortex.

[35] J. Kober and J. Peters, "Learning motor primitives for robotics," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 2112–2118. DOI: 10.1109/ROBOT.2009.5152577.

[36] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20, Vancouver, BC, Canada: Curran Associates Inc., 2020, ISBN: 9781713829546.

[37] C. Lauretti, F. Cordella, E. Guglielmelli, and L. Zollo, "Learning by demonstration for planning activities of daily living in rehabilitation and assistive robotics," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1375–1382, 2017. DOI: 10.1109/LRA.2017.2669369.

[38] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020. arXiv: 2005.01643. [Online]. Available: https://arxiv.org/abs/2005.01643.

[39] G. Li, L. Shi, Y. Chen, Y. Gu, and Y. Chi, "Breaking the sample complexity barrier to regret-optimal model-free reinforcement learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 17 762–17 776.

[40] M. Li, D. P. Losey, J. Bohg, and D. Sadigh, "Learning user-preferred mappings for intuitive robot control," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10 960–10 967, 2020.

[41] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971.

[42] D. Losey, H. Jeon, M. Li, *et al.*, "Learning latent actions to control assistive robots," *Autonomous Robots*, vol. 46, Jan. 2022. DOI: 10.1007/s10514-021-10005-w.

[43] D. P. Losey, K. Srinivasan, A. Mandlekar, A. Garg, and D. Sadigh, "Controlling Assistive Robots with Learned Latent Actions," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020. DOI: 10.1109/ICRA40945.2020.9197197.

[44] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1010–1017. DOI: 10.1109/IROS40897.2019.8968201.

[45] G. Matrone, C. Cipriani, M. C. Carrozza, and G. Magenes, "Real-time myoelectric control of a multi-fingered hand prosthesis using principal components analysis," *Journal of neuroengineering and rehabilitation*, vol. 9, p. 40, Jun. 2012. DOI: 10.1186/1743-0003-9-40.

[46] N. Mavridis, G. Pierris, P. Gallina, Z. Papamitsiou, and U. Saad, "On the subjective difficulty of joystick-based robot arm teleoperation with auditory feedback," in *2015 IEEE 8th GCC Conference Exhibition*, 2015, pp. 1–6. DOI: 10.1109/IEEEGCC.2015.7060097.

[47] B. A. Newman, R. M. Aronson, S. S. Srinivasa, K. Kitani, and H. Admoni, "Harmonic: A multimodal dataset of assistive human–robot collaboration," *The International Journal of Robotics Research*, vol. 41, no. 1, pp. 3–11, 2022. DOI: 10.1177/02783649211050677. eprint: https://doi.org/10.1177/02783649211050677. [Online]. Available: https://doi.org/10.1177/02783649211050677.

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

[49] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26, Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf.

[50] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., ser. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 661–668. [Online]. Available: https://proceedings.mlr.press/v9/ross10a.html.

[51] F. Routhier and P. S. Archambault, "Usability of a joystick-controlled six degree-of-freedom robotic manipulator.," in *RESNA ANNUAL Conference 2010*, 2010.

[52] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000. DOI: 10.1126/science.290.5500.2323. eprint: https://www.science.org/doi/pdf/10.1126/science.290.5500.2323. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.290.5500.2323.

[53] M. Santello, M. Flanders, and J. Soechting, "Postural hand synergies for tool use," *The Journal of Neuroscience*, vol. 18, pp. 10 105–15, Dec. 1998. DOI: 10.1523/JNEUROSCI.18-23-10105.1998.

[54] S. Schaal, "Dynamic movement primitives -a framework for motor control in humans and humanoid robotics," in *Adaptive Motion of Animals and Machines*, H. Kimura, K. Tsuchiya, A. Ishiguro, and H. Witte, Eds. Tokyo: Springer Tokyo, 2006, pp. 261–280, ISBN: 978-4-431-31381-6. DOI: 10.1007/4-431-31381-8_23. [Online]. Available: https://doi.org/10.1007/4-431-31381-8_23.

[55] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. DOI: 10.48550/ARXIV.1707.06347. [Online]. Available: https://arxiv.org/abs/1707.06347.

[56] S. Sivcev, J. Coleman, E. Omerdic, G. Dooly, and D. Toal, "Underwater manipulators: A review," *Ocean Engineering*, vol. 163, pp. 431–450, Sep. 2018. DOI: 10.1016/j.oceaneng.2018.06.018.

[57] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000. DOI: 10.1126/science.290.5500.2319. eprint: https://www.science.org/doi/pdf/10.1126/science.290.5500.2319. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.290.5500.2319.

[58] S. Tosatto, G. Chalvatzaki, and J. Peters, "Contextual latent-movements off-policy optimization for robotic manipulation skills," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 10 815–10 821. DOI: 10.1109/ICRA48506.2021.9561870.

[59] L. Trefethen and D. Bau, *Numerical Linear Algebra*, ser. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 1997, ISBN: 9780898713619. [Online]. Available: https://books.google.com/books?id=4Mou5YpRD%5C_kC.

[60] M. Yang, S. Levine, and O. Nachum, "TRAIL: Near-optimal imitation learning with suboptimal data," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=6q_2b6u0BnJ.

[61] W.-K. Yoon, T. Goshozono, H. Kawabe, *et al.*, "Model-based space robot teleoperation of ets-vii manipulator," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 602–612, 2004. DOI: 10.1109/TRA.2004.824700.

[62] W. Zhou, S. Bajracharya, and D. Held, "Plas: Latent action space for offline reinforcement learning," 2020.

Appendix A

Proofs

A.1 Smoothness of Exponential Map on Unit Sphere

For the sphere, the exponential map at $\mathbf{e}_1$ is computed as

$$\mathbf{v}(\boldsymbol{\xi}) = \mathbf{e}_1 \cos(\|\boldsymbol{\xi}\|) + \frac{1}{\|\boldsymbol{\xi}\|} \begin{bmatrix} 0 \\ \boldsymbol{\xi} \end{bmatrix} \sin(\|\boldsymbol{\xi}\|) \tag{A.1}$$

where $\boldsymbol{\xi} \in \mathbb{R}^{n-1}$ and $\mathbf{v} \in \mathbb{R}^n$.

We are interested in the Jacobian of the Exponential map:

$$\mathbf{J}_{Exp}(\boldsymbol{\xi}) = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \frac{\partial \mathbf{v}(\boldsymbol{\xi})}{\partial \xi_1} & \frac{\partial \mathbf{v}(\boldsymbol{\xi})}{\partial \xi_2} & \cdots & \frac{\partial \mathbf{v}(\boldsymbol{\xi})}{\partial \xi_{n-1}} \\ \vdots & \vdots & & \vdots \end{bmatrix} \tag{A.2}$$

Given $\mathbf{J}_{Exp}(\boldsymbol{\xi})$, we can compute the directional derivative of $\mathbf{v}$ with respect to a perturbation in the direction of unit vector $\boldsymbol{\delta}$ as:

$$\nabla_{\boldsymbol{\delta}} \mathbf{v}(\boldsymbol{\xi}) = \mathbf{J}_{Exp}(\boldsymbol{\xi}) \boldsymbol{\delta} \tag{A.3}$$

We will now show that $\|\nabla_{\boldsymbol{\delta}} \mathbf{v}\| \leq 1$, which implies that $\mathbf{v}$ is Lipschitz continuous with Lipschitz constant 1:

$$\|\mathbf{v}(\boldsymbol{\xi_1}) - \mathbf{v}(\boldsymbol{\xi_2})\| \leq L_{Exp} \|\boldsymbol{\xi_1} - \boldsymbol{\xi_2}\| \tag{A.4}$$

with $L_{Exp} = 1$.

Proof.

$$\mathbf{v}(\boldsymbol{\xi}) = \begin{bmatrix} \cos\|\boldsymbol{\xi}\| \\ \xi_1\|\boldsymbol{\xi}\|^{-1}\sin\|\boldsymbol{\xi}\| \\ \xi_2\|\boldsymbol{\xi}\|^{-1}\sin\|\boldsymbol{\xi}\| \\ \vdots \\ \xi_{n-1}\|\boldsymbol{\xi}\|^{-1}\sin\|\boldsymbol{\xi}\| \end{bmatrix} \tag{A.5}$$

We will compute $\frac{\partial \mathbf{v}}{\partial \xi_j}$, the partial derivative of $\mathbf{v}$ with respect to each $\boldsymbol{\xi}_i$. We consider the first element of $\mathbf{v}$ independently, and then we write a general expression for the remaining $n-2$ terms of $\frac{\partial \mathbf{v}}{\partial \xi_j}$.

First element of $\partial \mathbf{v}/\partial \xi_j$ First note that the derivative of the 2-norm of a vector with respect to a specific element in that vector is:

$$\frac{\partial}{\partial \xi_j}\|\boldsymbol{\xi}\| = \frac{\xi_j}{\|\boldsymbol{\xi}\|} \tag{A.6}$$

Now, by the chain rule, we have

$$\frac{\partial v_1}{\partial \xi_j} = \frac{\partial}{\partial \xi_j}\cos\|\boldsymbol{\xi}\| = -\xi_j\|\boldsymbol{\xi}\|^{-1}\sin\|\boldsymbol{\xi}\| \tag{A.7}$$

Remaining elements of $\partial \mathbf{v}/\partial \xi_j$ First, we note the following:

$$\frac{\partial}{\partial \xi_j}\|\boldsymbol{\xi}\|^{-1} = -\frac{\xi_j}{\|\boldsymbol{\xi}\|^3} \tag{A.8}$$

which can be shown using the chain rule with our previous result regarding $\frac{\partial}{\partial \xi_j}\|\boldsymbol{\xi}\|$. Now, consider $\frac{\partial}{\partial \xi_j}\xi_i\|\boldsymbol{\xi}\|^{-1}$, which has a different form depending on whether $i = j$:

$$\frac{\partial}{\partial \xi_j}\xi_i\|\boldsymbol{\xi}\|^{-1} = \xi_i\frac{\partial}{\partial \xi_j}\|\boldsymbol{\xi}\|^{-1} + \|\boldsymbol{\xi}\|^{-1}\frac{\partial}{\partial \xi_j}\xi_i \tag{A.9}$$

which is just an application of the product rule. The left hand term includes a factor $\frac{\partial}{\partial \xi_j}\xi_i$, which is equal to zero if $i \neq j$, and equal to one if $i = j$. Hence we can write:

$$\frac{\partial}{\partial \xi_j}\xi_i\|\boldsymbol{\xi}\|^{-1} = \xi_i\frac{\partial}{\partial \xi_j}\|\boldsymbol{\xi}\|^{-1} + \mathbb{1}_{i=j}\|\boldsymbol{\xi}\|^{-1} \tag{A.10}$$

$$= -\xi_i\xi_j\|\boldsymbol{\xi}\|^{-3} + \mathbb{1}_{i=j}\|\boldsymbol{\xi}\|^{-1} \tag{A.11}$$

where $\mathbb{1}$ denotes the indicator function, that is:

$$\mathbb{1}_{i=j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{A.12}$$

Finally, it can be shown with an additional application of the product rule that

$$\frac{\partial v_{i+1}}{\partial \xi_j} = \frac{\partial}{\partial \xi_j} \xi_i \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| \tag{A.13}$$

$$= \xi_i \xi_j \|\boldsymbol{\xi}\|^{-2} \cos\|\boldsymbol{\xi}\| + \left(\mathbb{1}_{i=j} - \xi_i \xi_j \|\boldsymbol{\xi}\|^{-2}\right) \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| \tag{A.14}$$

Thus the elements of $\partial \mathbf{v}/\partial \xi_j$ can be written as:

$$\frac{\partial v_{i+1}}{\partial \xi_j} = \begin{cases} -\xi_j \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| & i = 0 \\ \xi_i \xi_j \|\boldsymbol{\xi}\|^{-2} \cos\|\boldsymbol{\xi}\| + \left(\mathbb{1}_{i=j} - \xi_i \xi_j \|\boldsymbol{\xi}\|^{-2}\right) \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| & 1 \leq i \leq n-1 \end{cases} \tag{A.15}$$

We now return to $\nabla_{\boldsymbol{\delta}} \mathbf{v}(\boldsymbol{\xi})$:

$$\nabla_{\boldsymbol{\delta}} \mathbf{v}(\boldsymbol{\xi}) = \mathbf{J}_{Exp}(\boldsymbol{\xi}) \boldsymbol{\delta} = \sum_{j=1}^{n-1} \delta_j \frac{\partial \mathbf{v}(\boldsymbol{\xi})}{\partial \xi_j} \tag{A.16}$$

Here we will again treat the first element of $\nabla_{\boldsymbol{\delta}} \mathbf{v}$ separately. We will denote the ith element of $\nabla_{\boldsymbol{\delta}} \mathbf{v}$ as $[\nabla_{\boldsymbol{\delta}} \mathbf{v}]_i$.

$$[\nabla_{\boldsymbol{\delta}} \mathbf{v}]_1 = \sum_{j=1}^{n-1} -\delta_j \xi_j \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| \tag{A.17}$$

$$= -\|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| \sum_{j=1}^{n-1} \delta_j \xi_j \tag{A.18}$$

$$= -\langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| \tag{A.19}$$

where $\langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle$ denotes the inner (dot) product of $\boldsymbol{\delta}$ and $\boldsymbol{\xi}$.

Now, for the remaining elements of $\nabla_{\boldsymbol{\delta}} \mathbf{v}$:

$$[\nabla_{\boldsymbol{\delta}} \mathbf{v}]_{i+1} = \sum_{j=1}^{n-1} \delta_j \left(\xi_i \xi_j \|\boldsymbol{\xi}\|^{-2} \cos\|\boldsymbol{\xi}\|\right) \tag{A.20}$$

$$+ \sum_{j=1}^{n-1} \delta_j \left(\mathbb{1}_{i=j} - \xi_i \xi_j \|\boldsymbol{\xi}\|^{-2}\right) \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| \tag{A.21}$$

$$= \xi_i \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle \|\boldsymbol{\xi}\|^{-2} \cos\|\boldsymbol{\xi}\| + \left(\delta_i - \xi_i \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle \|\boldsymbol{\xi}\|^{-2}\right) \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\| \tag{A.22}$$

To simplify the remaining algebra, we will define:

$$\alpha_i = \xi_i \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle \|\boldsymbol{\xi}\|^{-2} \tag{A.23}$$

$$\beta_i = \|\boldsymbol{\xi}\|^{-1} (\delta_i - \alpha_i) \tag{A.24}$$

which allows us to write $[\nabla_{\boldsymbol{\delta}}\mathbf{v}]_{i+1}$ simply as:

$$[\nabla_{\boldsymbol{\delta}}\mathbf{v}]_{i+1} = \alpha_i \cos\|\boldsymbol{\xi}\| + \beta_i \sin\|\boldsymbol{\xi}\| \tag{A.25}$$

Now we consider the squared norm of $\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})$:

$$\|\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})\|^2 = \sum_{i=1}^{n} [\nabla_{\boldsymbol{\delta}}\mathbf{v}]_i^2 \tag{A.26}$$

$$= [\nabla_{\boldsymbol{\delta}}\mathbf{v}]_1^2 + \sum_{i=1}^{n-1} [\nabla_{\boldsymbol{\delta}}\mathbf{v}]_{i+1}^2 \tag{A.27}$$

$$= \left(\langle \boldsymbol{\delta}, \boldsymbol{\xi}\rangle \|\boldsymbol{\xi}\|^{-1} \sin\|\boldsymbol{\xi}\|\right)^2 + \sum_{i=1}^{n-1} \left(\alpha_i \cos\|\boldsymbol{\xi}\| + \beta_i \sin\|\boldsymbol{\xi}\|\right)^2 \tag{A.28}$$

Let us consider the right-hand term separately:

$$\sum_{i=1}^{n-1} \left(\alpha_i \cos\|\boldsymbol{\xi}\| + \beta_i \sin\|\boldsymbol{\xi}\|\right)^2 \tag{A.29}$$

$$= \sum_{i=1}^{n-1} \alpha_i^2 \cos^2\|\boldsymbol{\xi}\| + \beta_i^2 \sin^2\|\boldsymbol{\xi}\| + 2\alpha\beta \cos\|\boldsymbol{\xi}\| \sin\|\boldsymbol{\xi}\| \tag{A.30}$$

$$= \cos^2\|\boldsymbol{\xi}\| \sum_{i=1}^{n-1} \alpha_i^2 + \sin^2\|\boldsymbol{\xi}\| \sum_{i=1}^{n-1} \beta_i^2 + 2\cos\|\boldsymbol{\xi}\| \sin\|\boldsymbol{\xi}\| \sum_{i=1}^{n-1} \alpha\beta \tag{A.31}$$

Which leads us to examine $\sum_{i=1}^{n-1} \alpha_i^2$, $\sum_{i=1}^{n-1} \beta_i^2$, and $\sum_{i=1}^{n-1} \alpha\beta$. First, we will consider $\sum_{i=1}^{n-1} \alpha_i^2$:

$$\sum_{i=1}^{n-1} \alpha_i^2 = \sum_{i=1}^{n-1} \left(\xi_i \langle \boldsymbol{\delta}, \boldsymbol{\xi}\rangle \|\boldsymbol{\xi}\|^{-2}\right)^2 \tag{A.32}$$

$$= \langle \boldsymbol{\delta}, \boldsymbol{\xi}\rangle^2 \|\boldsymbol{\xi}\|^{-4} \sum_{i=1}^{n-1} \xi_i^2 \tag{A.33}$$

$$= \langle \boldsymbol{\delta}, \boldsymbol{\xi}\rangle^2 \|\boldsymbol{\xi}\|^{-4} \|\boldsymbol{\xi}\|^2 \tag{A.34}$$

$$= \langle \boldsymbol{\delta}, \boldsymbol{\xi}\rangle^2 \|\boldsymbol{\xi}\|^{-2} \tag{A.35}$$

Next, we consider $\sum_{i=1}^{n-1} \beta_i^2$:

$$\sum_{i=1}^{n-1} \beta_i^2 = \sum_{i=1}^{n-1} \left(\|\boldsymbol{\xi}\|^{-1}(\delta_i - \alpha_i) \right)^2 \tag{A.36}$$

$$= \|\boldsymbol{\xi}\|^{-2} \sum_{i=1}^{n-1} \left(\delta_i^2 + \alpha_i^2 - 2\delta_i\alpha_i \right) \tag{A.37}$$

$$= \|\boldsymbol{\xi}\|^{-2} \left(\sum_{i=1}^{n-1} \delta_i^2 + \sum_{i=1}^{n-1} \alpha_i^2 - 2\sum_{i=1}^{n-1} \delta_i\alpha_i \right) \tag{A.38}$$

$$= \|\boldsymbol{\xi}\|^{-2} \left(\|\boldsymbol{\delta}\|^2 + \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle^2 \|\boldsymbol{\xi}\|^{-2} - 2\sum_{i=1}^{n-1} \delta_i\xi_i \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle \|\boldsymbol{\xi}\|^{-2} \right) \tag{A.39}$$

$$= \|\boldsymbol{\xi}\|^{-2} \left(1 + \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle^2 \|\boldsymbol{\xi}\|^{-2} - 2\langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle \|\boldsymbol{\xi}\|^{-2} \sum_{i=1}^{n-1} \delta_i\xi_i \right) \tag{A.40}$$

$$= \|\boldsymbol{\xi}\|^{-2} \left(1 + \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle^2 \|\boldsymbol{\xi}\|^{-2} - 2\langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle^2 \|\boldsymbol{\xi}\|^{-2} \right) \tag{A.41}$$

$$= \|\boldsymbol{\xi}\|^{-2} \left(1 - \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle^2 \|\boldsymbol{\xi}\|^{-2} \right) \tag{A.42}$$

$$\tag{A.43}$$

where we use the fact that $\|\boldsymbol{\delta}\| = 1$. Finally, we consider $\sum_{i=1}^{n-1} \alpha\beta$:

$$\sum_{i=1}^{n-1} \alpha\beta = \sum_{i=1}^{n-1} \alpha_i \|\boldsymbol{\xi}\|^{-1}(\delta_i - \alpha_i) \tag{A.44}$$

$$= \sum_{i=1}^{n-1} \|\boldsymbol{\xi}\|^{-1}(\delta_i\alpha_i - \alpha_i^2) \tag{A.45}$$

$$= \|\boldsymbol{\xi}\|^{-1} \left(\sum_{i=1}^{n-1} \delta_i\alpha_i - \sum_{i=1}^{n-1} \alpha_i^2 \right) \tag{A.46}$$

We saw in our calculation for $\sum_{i=1}^{n-1} \beta_i^2$ that $\sum_{i=1}^{n-1} \delta_i\alpha_i = \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle^2 \|\boldsymbol{\xi}\|^{-2}$. We also computed that $\sum_{i=1}^{n-1} \alpha_i^2 = \langle \boldsymbol{\delta}, \boldsymbol{\xi} \rangle^2 \|\boldsymbol{\xi}\|^{-2}$. Hence $\sum_{i=1}^{n-1} \delta_i\alpha_i = \sum_{i=1}^{n-1} \alpha_i^2$, and the right hand side of equation A.46 cancels to become zero.

$$\sum_{i=1}^{n-1} \alpha\beta = 0 \tag{A.47}$$

Now, returning to the squared norm of $\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})$, we substitute the results of our computations for the terms involving α and β:

$$\|\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})\|^2 = \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| + \sum_{i=1}^{n-1}\left(\alpha_i\cos\|\boldsymbol{\xi}\| + \beta_i\sin\|\boldsymbol{\xi}\|\right)^2 \tag{A.48}$$

$$= \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| + \cos^2\|\boldsymbol{\xi}\|\sum_{i=1}^{n-1}\alpha_i^2 + \sin^2\|\boldsymbol{\xi}\|\sum_{i=1}^{n-1}\beta_i^2 \tag{A.49}$$

$$= \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| + \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\cos^2\|\boldsymbol{\xi}\| + \sin^2\|\boldsymbol{\xi}\|\sum_{i=1}^{n-1}\beta_i^2 \tag{A.50}$$

$$= \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\left(\sin^2\|\boldsymbol{\xi}\| + \cos^2\|\boldsymbol{\xi}\|\right) + \sin^2\|\boldsymbol{\xi}\|\sum_{i=1}^{n-1}\beta_i^2 \tag{A.51}$$

$$= \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2} + \sin^2\|\boldsymbol{\xi}\|\sum_{i=1}^{n-1}\beta_i^2 \tag{A.52}$$

$$= \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2} + \|\boldsymbol{\xi}\|^{-2}\left(1 - \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\right)\sin^2\|\boldsymbol{\xi}\| \tag{A.53}$$

$$= \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2} + \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| - \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-4}\sin^2\|\boldsymbol{\xi}\| \tag{A.54}$$

$$= \langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\left(1 - \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\|\right) + \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| \tag{A.55}$$

We know that $\|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| \leq 1$, so the left hand term in equation A.55 is always positive. Hence, $\|\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})\|^2$ is maximized whenever $\boldsymbol{\delta}$ is in the same direction as $\boldsymbol{\xi}$. When this is the case, $\langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle = \|\boldsymbol{\xi}\|$ (recall that $\|\boldsymbol{\delta}\| = 1$). We then have:

$$\max\|\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})\|^2 = \max\langle\boldsymbol{\delta},\boldsymbol{\xi}\rangle^2\|\boldsymbol{\xi}\|^{-2}\left(1 - \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\|\right) + \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| \tag{A.56}$$

$$= \|\boldsymbol{\xi}\|^2\|\boldsymbol{\xi}\|^{-2}\left(1 - \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\|\right) + \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| \tag{A.57}$$

$$= 1 - \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| + \|\boldsymbol{\xi}\|^{-2}\sin^2\|\boldsymbol{\xi}\| \tag{A.58}$$

$$= 1 \tag{A.59}$$

Hence $\max\|\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})\|^2 = 1$ for all $\boldsymbol{\xi}$, and therefore $\max\|\nabla_{\boldsymbol{\delta}}\mathbf{v}(\boldsymbol{\xi})\| = 1$, which implies that the exponential map on the unit sphere at $\mathbf{e}_1$ is Lipschitz continuous with Lipschitz constant $L_{Exp} = 1$. □

A.2 Smoothness of $\mathbf{Q}(\bar{\mathbf{v}})$

In this section we prove the Lipschitz continuity of $\mathbf{Q}(\bar{\mathbf{v}})$, as stated in theorem 1.

Theorem 1. *Let $\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2 \in \mathbb{R}^{nk}$ be constructed from k stacked unit n-vectors, and $\mathbf{Q}(\bar{\mathbf{v}})$ be the product of the corresponding Householder reflections (as defined in Eq. 4.7, 4.8). Then,*

$$\|\mathbf{Q}(\bar{\mathbf{v}}_1) - \mathbf{Q}(\bar{\mathbf{v}}_2)\| \leq L_Q \|\bar{\mathbf{v}}_1 - \bar{\mathbf{v}}_2\| \tag{4.20}$$

where $L_Q = 2\sqrt{k}$.

We write $\mathbf{H}_i$ as shorthand for $\mathbf{H}(\mathbf{v}_i) = \mathbf{I} - 2\mathbf{v}_i\mathbf{v}_i^\top$. We write $\bar{\mathbf{v}} \in \mathbb{R}^{nk}$ to denote the concatenated column vector of $\mathbf{v}_i \in \mathbb{R}^n$: $\bar{\mathbf{v}} = [\mathbf{v}_1^\top, \mathbf{v}_2^\top, \ldots, \mathbf{v}_k^\top]^\top$. We denote the map from $\bar{\mathbf{v}}$ to the corresponding product of reflections as $Q : \bar{\mathbf{v}} \mapsto \mathbf{Q}(\bar{\mathbf{v}})$, where

$$\mathbf{Q}(\bar{\mathbf{v}}) = \mathbf{H}(\mathbf{v}_1)\mathbf{H}(\mathbf{v}_2)\cdots\mathbf{H}(\mathbf{v}_k) \tag{A.60}$$

We likewise write $\bar{\boldsymbol{\delta}} \in \mathbb{R}^{nk}$ to denote the concatenated vector of perturbations to each $\mathbf{v}_i$

$$\bar{\boldsymbol{\delta}} = \begin{bmatrix} \boldsymbol{\delta}'_1 \\ \boldsymbol{\delta}'_2 \\ \vdots \\ \boldsymbol{\delta}'_k \end{bmatrix} = \begin{bmatrix} c_1\boldsymbol{\delta}_1 \\ c_2\boldsymbol{\delta}_2 \\ \vdots \\ c_k\boldsymbol{\delta}_k \end{bmatrix} \tag{A.61}$$

where $\|\bar{\boldsymbol{\delta}}\| = 1$, with scalars $c_i \in \mathbb{R}$ scaling the unit norm $\boldsymbol{\delta}_i$ vectors that represent the direction of change for each $\mathbf{v}_i$. We consider the directional derivative of $\mathbf{Q}(\bar{\mathbf{v}})$ in the direction of $\bar{\boldsymbol{\delta}}$:

$$\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}(\bar{\mathbf{v}}) \doteq \lim_{\epsilon \to 0} \frac{\mathbf{Q}(\bar{\mathbf{v}} + \epsilon\bar{\boldsymbol{\delta}}) - \mathbf{Q}(\bar{\mathbf{v}})}{\epsilon} \tag{A.62}$$

where $\|\bar{\boldsymbol{\delta}}\| = 1$.

The existence of a positive constant L_Q that bounds $\|\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}(\bar{\mathbf{v}})\|$ implies Lipschitz continuity of $\mathbf{Q}(\bar{\mathbf{v}})$:

$$\|\mathbf{Q}(\bar{\mathbf{v}}_1) - \mathbf{Q}(\bar{\mathbf{v}}_2)\| \leq L_Q \|\bar{\mathbf{v}}_1 - \bar{\mathbf{v}}_2\| \tag{A.63}$$

for all $\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2$ constructed with k stacked unit n-vectors. We explicitly compute such an L_Q below. As a first step, we show in section A.2 that $\|\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v})\| = 2$. We will then use this result to compute an upper bound on L_Q in section A.2.

Lipschitz Continuity of $\mathbf{H}(\mathbf{v})$

The directional derivative of $\mathbf{H}(\mathbf{v})$ in the direction of $\boldsymbol{\delta}$ is defined as:

$$\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v}) \doteq \lim_{\epsilon\to 0} \frac{\mathbf{H}(\mathbf{v}+\epsilon\boldsymbol{\delta}) - \mathbf{H}(\mathbf{v})}{\epsilon} \tag{A.64}$$

where $\boldsymbol{\delta} \in \mathbb{R}^n$. Recall that $\mathbf{v}$ is in the $n-1$ sphere, and thus any instantaneous change to $\mathbf{v}$ must occur in a direction tangent to the sphere at $\mathbf{v}$; that is, $\boldsymbol{\delta} \perp \mathbf{v}$. Furthermore, without loss of generality, we let $\|\boldsymbol{\delta}\| = 1$. Thus, $\boldsymbol{\delta}$ is a unit vector in the direction of the perturbation of $\mathbf{v}$.

We first simplify the first term in the numerator:

$$\begin{aligned}
\mathbf{H}(\mathbf{v}+\epsilon\boldsymbol{\delta}) &= \mathbf{I} - 2(\mathbf{v}+\epsilon\boldsymbol{\delta})(\mathbf{v}+\epsilon\boldsymbol{\delta})^\top && \text{(A.65)}\\
&= \mathbf{I} - 2(\mathbf{v}\mathbf{v}^\top + \epsilon\boldsymbol{\delta}\mathbf{v}^\top + \epsilon\mathbf{v}\boldsymbol{\delta}^\top + \epsilon^2\boldsymbol{\delta}\boldsymbol{\delta}^\top) && \text{(A.66)}\\
&= \mathbf{I} - 2\mathbf{v}\mathbf{v}^\top - 2(\epsilon\boldsymbol{\delta}\mathbf{v}^\top + \epsilon\mathbf{v}\boldsymbol{\delta}^\top + \epsilon^2\boldsymbol{\delta}\boldsymbol{\delta}^\top) && \text{(A.67)}\\
&= \mathbf{H}(\mathbf{v}) - 2(\epsilon\boldsymbol{\delta}\mathbf{v}^\top + \epsilon\mathbf{v}\boldsymbol{\delta}^\top + \epsilon^2\boldsymbol{\delta}\boldsymbol{\delta}^\top) && \text{(A.68)}
\end{aligned}$$

Substituting the result into the definition of $\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v})$, we have:

$$\begin{aligned}
\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v}) &= \lim_{\epsilon\to 0} \frac{-2(\epsilon\boldsymbol{\delta}\mathbf{v}^\top + \epsilon\mathbf{v}\boldsymbol{\delta}^\top + \epsilon^2\boldsymbol{\delta}\boldsymbol{\delta}^\top)}{\epsilon} && \text{(A.69)}\\
&= \lim_{\epsilon\to 0} -2(\boldsymbol{\delta}\mathbf{v}^\top + \mathbf{v}\boldsymbol{\delta}^\top + \epsilon\boldsymbol{\delta}\boldsymbol{\delta}^\top) && \text{(A.70)}\\
&= -2(\boldsymbol{\delta}\mathbf{v}^\top + \mathbf{v}\boldsymbol{\delta}^\top) && \text{(A.71)}
\end{aligned}$$

Now we compute $\|\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v})\|$. Note that the symmetry of the sphere guarantees that $\|\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v})\|$ is invariant with respect to both $\boldsymbol{\delta}$ and $\mathbf{v}$.

$$\|\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v})\| \doteq \max_{\mathbf{x}\neq 0} \frac{\|\nabla_{\boldsymbol{\delta}}\mathbf{H}(\mathbf{v})\mathbf{x}\|}{\|\mathbf{x}\|} \tag{A.72}$$

The numerator is maximized when $\mathbf{x}$ is in the plane spanned by $\mathbf{v}$ and $\boldsymbol{\delta}$. Given this is the case, we can write $\mathbf{x}$ as a linear combination of $\mathbf{v}$ and $\boldsymbol{\delta}$. Let

$$\mathbf{x} = \alpha\mathbf{v} + \beta\boldsymbol{\delta} \tag{A.73}$$

for some $\alpha, \beta \in \mathbb{R}$. We then have the following:

$$\|\nabla_{\boldsymbol{\delta}} \mathbf{H}(\mathbf{v})\| = \max_{\mathbf{x}\neq 0} \frac{\|\nabla_{\boldsymbol{\delta}} \mathbf{H}(\mathbf{v})\mathbf{x}\|}{\|\mathbf{x}\|} \tag{A.74}$$

$$= \max_{\mathbf{x}\neq 0} \frac{\|-2(\boldsymbol{\delta}\mathbf{v}^\top + \mathbf{v}\boldsymbol{\delta}^\top)\mathbf{x}\|}{\|\mathbf{x}\|} \tag{A.75}$$

$$= 2\frac{\|(\boldsymbol{\delta}\mathbf{v}^\top + \mathbf{v}\boldsymbol{\delta}^\top)(\alpha\mathbf{v} + \beta\boldsymbol{\delta})\|}{\|\mathbf{x}\|} \tag{A.76}$$

$$= 2\frac{\|\alpha\boldsymbol{\delta}\mathbf{v}^\top\mathbf{v} + \alpha\mathbf{v}\boldsymbol{\delta}^\top\mathbf{v} + \beta\boldsymbol{\delta}\mathbf{v}^\top\boldsymbol{\delta} + \beta\mathbf{v}\boldsymbol{\delta}^\top\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \tag{A.77}$$

$$= 2\frac{\|\alpha\boldsymbol{\delta}\mathbf{v}^\top\mathbf{v} + \beta\mathbf{v}\boldsymbol{\delta}^\top\boldsymbol{\delta}\|}{\|\mathbf{x}\|} \tag{A.78}$$

$$= 2\frac{\|\alpha\boldsymbol{\delta} + \beta\mathbf{v}\|}{\|\mathbf{x}\|} \tag{A.79}$$

where equation A.78 follows from A.77 by the fact that $\boldsymbol{\delta} \perp \mathbf{v}$. Equation A.79 follows from the fact that both $\boldsymbol{\delta}$ and $\mathbf{v}$ have unit norm.

Now, recall that $\mathbf{x} = \alpha\mathbf{v} + \beta\boldsymbol{\delta}$. The numerator in A.79 represents a simple change of basis for $\mathbf{x}$. Since $\boldsymbol{\delta}$ and $\mathbf{v}$ are orthonormal, this change of basis preserves the norm of $\mathbf{x}$. Hence $\|\alpha\boldsymbol{\delta} + \beta\mathbf{v}\| = \|\mathbf{x}\|$, and we have:

$$\|\nabla_{\boldsymbol{\delta}} \mathbf{H}(\mathbf{v})\| = 2 \tag{A.80}$$

This implies $\mathbf{H}(\mathbf{v})$ is Lipschitz continuous with Lipschitz constant 2.

Lipschitz Continuity of Q

We now consider the directional derivative of $\mathbf{Q}(\bar{\mathbf{v}})$ in the direction of $\bar{\boldsymbol{\delta}}$:

$$\nabla_{\bar{\boldsymbol{\delta}}} \mathbf{Q}(\bar{\mathbf{v}}) \doteq \lim_{\epsilon\to 0} \frac{\mathbf{Q}(\bar{\mathbf{v}} + \epsilon\bar{\boldsymbol{\delta}}) - \mathbf{Q}(\bar{\mathbf{v}})}{\epsilon} \tag{A.81}$$

where $\|\bar{\boldsymbol{\delta}}\| = 1$. Recall:

$$\bar{\boldsymbol{\delta}} = \begin{bmatrix} \boldsymbol{\delta}'_1 \\ \boldsymbol{\delta}'_2 \\ \vdots \\ \boldsymbol{\delta}'_k \end{bmatrix} = \begin{bmatrix} c_1\boldsymbol{\delta}_1 \\ c_2\boldsymbol{\delta}_2 \\ \vdots \\ c_k\boldsymbol{\delta}_k \end{bmatrix} \tag{A.82}$$

Theorem 1. *Let $\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2 \in \mathbb{R}^{nk}$ be constructed from k stacked unit n-vectors, and $\mathbf{Q}(\bar{\mathbf{v}})$ be the product of the corresponding Householder reflections (as defined in Eq. 4.7, 4.8). Then,*

$$\|\mathbf{Q}(\bar{\mathbf{v}}_1) - \mathbf{Q}(\bar{\mathbf{v}}_2)\| \leq L_Q\|\bar{\mathbf{v}}_1 - \bar{\mathbf{v}}_2\| \tag{4.20}$$

where $L_Q = 2\sqrt{k}$.

Proof. We begin by expanding the numerator of $\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}$

$$\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}(\bar{\mathbf{v}}) = \lim_{\epsilon\to 0} \frac{\mathbf{H}(\mathbf{v}_1 + \epsilon\boldsymbol{\delta}'_1)\mathbf{H}(\mathbf{v}_2 + \epsilon\boldsymbol{\delta}'_2)\cdots\mathbf{H}(\mathbf{v}_k + \epsilon\boldsymbol{\delta}'_k) - \mathbf{Q}(\bar{\mathbf{v}})}{\epsilon} \tag{A.83}$$

We now consider the first term in the numerator. In the following we write $\mathbf{H}_i$ as shorthand for $\mathbf{H}(\mathbf{v}_i)$, and $\nabla\mathbf{H}_i$ as shorthand for $\nabla_{\boldsymbol{\delta}_i}\mathbf{H}(\mathbf{v}_i)$, the derivative of $\mathbf{H}(\mathbf{v}_i)$ as defined in equation (A.64).

$$\mathbf{H}(\mathbf{v}_1 + \epsilon\boldsymbol{\delta}'_1)\mathbf{H}(\mathbf{v}_2 + \epsilon\boldsymbol{\delta}'_2)\cdots\mathbf{H}(\mathbf{v}_k + \epsilon\boldsymbol{\delta}'_k) \tag{A.84}$$

$$= (\mathbf{H}_1 + \epsilon c_1\nabla\mathbf{H}_1)(\mathbf{H}_2 + \epsilon c_2\nabla\mathbf{H}_2)\cdots(\mathbf{H}_k + \epsilon c_k\nabla\mathbf{H}_k) + O(\epsilon^2) \tag{A.85}$$

$$= \mathbf{H}_1\mathbf{H}_2\cdots\mathbf{H}_k + \epsilon c_1(\nabla\mathbf{H}_1)\mathbf{H}_2\cdots\mathbf{H}_k + \epsilon c_2\mathbf{H}_1(\nabla\mathbf{H}_2)\mathbf{H}_3\cdots\mathbf{H}_k + \ldots \tag{A.86}$$

$$+ \epsilon c_k\mathbf{H}_1\mathbf{H}_2\cdots\mathbf{H}_{k-1}(\nabla\mathbf{H}_k) + O(\epsilon^2) \tag{A.87}$$

$$= \mathbf{Q}(\bar{\mathbf{v}}) + \epsilon c_1(\nabla\mathbf{H}_1)(\prod_{i=2}^{k}\mathbf{H}_i) + \epsilon c_2\mathbf{H}_1(\nabla\mathbf{H}_2)(\prod_{i=3}^{k}\mathbf{H}_i) + \cdots + \epsilon c_k(\prod_{i=1}^{k-1}\mathbf{H}_i)(\nabla\mathbf{H}_k) + O(\epsilon^2) \tag{A.88}$$

$$= \mathbf{Q}(\bar{\mathbf{v}}) + \epsilon\sum_{j=1}^{k} c_j\left[(\prod_{i=1}^{j-1}\mathbf{H}_i)(\nabla\mathbf{H}_j)(\prod_{l=j+1}^{k}\mathbf{H}_l)\right] + O(\epsilon^2) \tag{A.89}$$

and substitute the result into the definition of $\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}(\bar{\mathbf{v}})$:

$$\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}(\bar{\mathbf{v}}) = \lim_{\epsilon\to 0} \frac{\mathbf{Q}(\bar{\mathbf{v}}) + \epsilon\sum_{j=1}^{k} c_j\left[(\prod_{i=1}^{j-1}\mathbf{H}_i)(\nabla\mathbf{H}_j)(\prod_{l=j+1}^{k}\mathbf{H}_l)\right] + O(\epsilon^2) - \mathbf{Q}(\bar{\mathbf{v}})}{\epsilon} \tag{A.90}$$

$$= \sum_{j=1}^{k} c_j\left[(\prod_{i=1}^{j-1}\mathbf{H}_i)(\nabla\mathbf{H}_j)(\prod_{l=j+1}^{k}\mathbf{H}_l)\right] \tag{A.91}$$

We now work toward bounding the norm of $\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}(\bar{\mathbf{v}})$:

$$\|\nabla_{\bar{\boldsymbol{\delta}}}\mathbf{Q}(\bar{\mathbf{v}})\| = \left\| \sum_{j=1}^{k} c_j \left[(\prod_{i=1}^{j-1} \mathbf{H}_i)(\nabla \mathbf{H}_j)(\prod_{l=j+1}^{k} \mathbf{H}_l) \right] \right\| \tag{A.92}$$

$$\leq \sum_{j=1}^{k} c_j \left\| (\prod_{i=1}^{j-1} \mathbf{H}_i)(\nabla \mathbf{H}_j)(\prod_{l=j+1}^{k} \mathbf{H}_l) \right\| \tag{A.93}$$

$$= \sum_{j=1}^{k} c_j \|\nabla \mathbf{H}_j\| \tag{A.94}$$

$$\leq \left(\sqrt{\sum_{j=1}^{k} c_j^2} \right) \left(\sqrt{\sum_{j=1}^{k} \|\nabla \mathbf{H}_j\|^2} \right) \tag{A.95}$$

$$= \left\|\bar{\boldsymbol{\delta}}\right\| \sqrt{\sum_{j=1}^{k} \|\nabla_{\boldsymbol{\delta}_j} \mathbf{H}(\mathbf{v}_j)\|^2} \tag{A.96}$$

$$= \sqrt{\sum_{j=1}^{k} (2)^2} \tag{A.97}$$

$$= 2\sqrt{k} \tag{A.98}$$

Where equation (A.94) is thanks to the fact that each of the $\mathbf{H}_i$ in the preceding equation are orthogonal, and equation (A.95) follows by the Cauchy-Schwarz inequality.

Hence, the norm of the directional derivative of $\mathbf{Q}(\bar{\mathbf{v}})$ is bounded by $2\sqrt{k}$; that is:

$$\|\nabla \mathbf{Q}(\bar{\mathbf{v}})\| \leq 2\sqrt{k} \tag{A.99}$$

which implies

$$\|\mathbf{Q}(\bar{\mathbf{v}}_1) - \mathbf{Q}(\bar{\mathbf{v}}_2)\| \leq L_Q \|\bar{\mathbf{v}}_1 - \bar{\mathbf{v}}_2\| \tag{A.100}$$

with Lipschitz constant $L_Q = 2\sqrt{k}$. □